Database Systems

Instructor: Bilal Khalid Dar

Relational Algebra

DBMS Architecture

How does a SQL engine work ?

- SQL query $\rightarrow$ relational algebra plan
- Relational algebra plan $\rightarrow$ Optimized plan
- Execute each operator of the plan

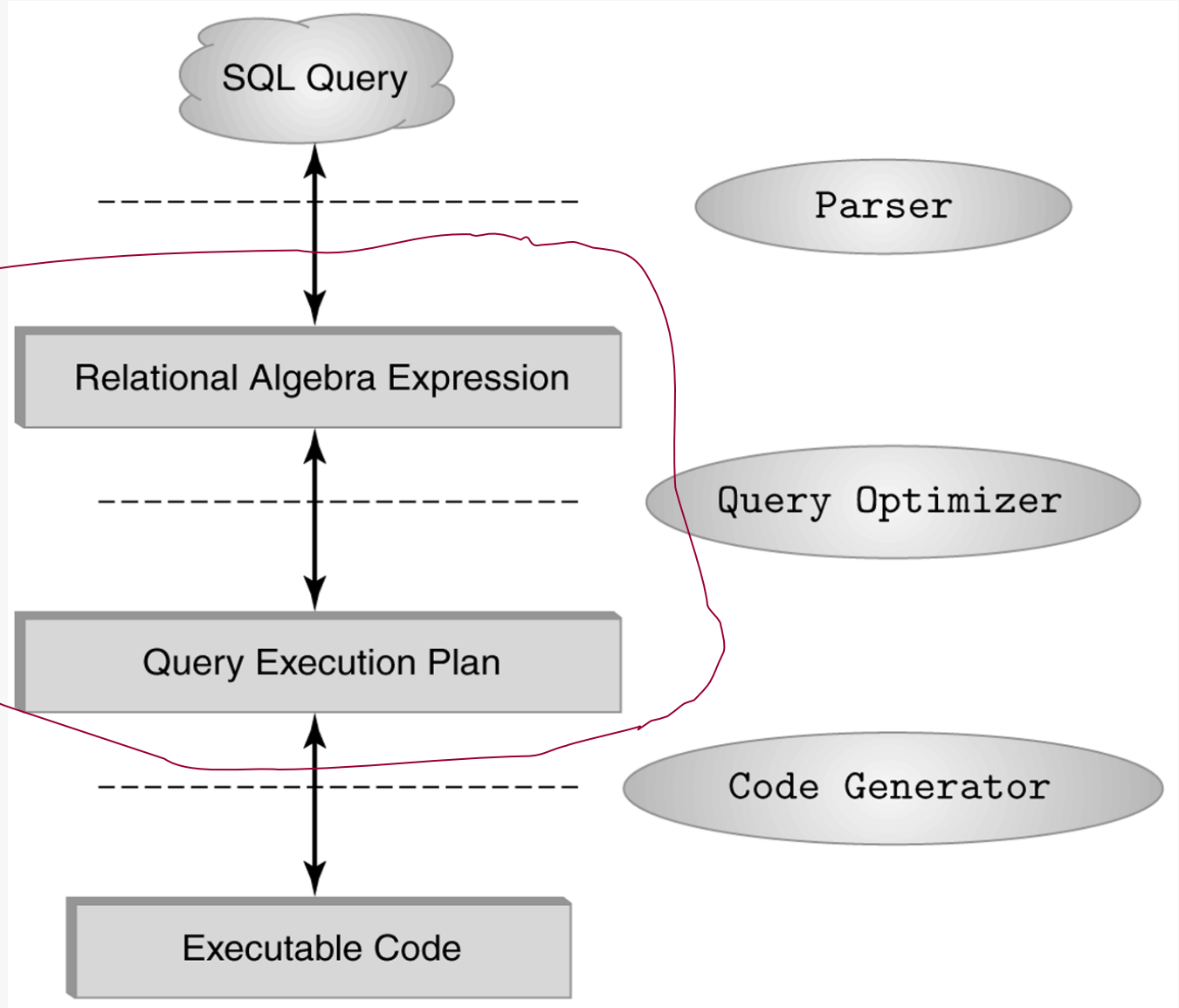
What is an Algebra?

- A language based on operators and a domain of values
- Operators map values taken from the domain into other domain values
- Hence, an expression involving operators and arguments produces a value in the domain
- When the domain is a set of all relations (and the operators are as described later), we get the *relational algebra*
- We refer to the expression as a *query* and the value produced as the *query result*

Relational Algebra

- Five operators:
  - Union: $\cup$
  - Difference: -
  - Selection: σ
  - Projection: Π
  - Cartesian Product: $\times$
- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
  - Renaming: ρ

The Role of Relational Algebra in a DBMS



1. Union and 2. Difference

- R1 $\cup$ R2
- Example:
  - ActiveEmployees $\cup$ RetiredEmployees

- R1 – R2
- Example:
  - AllEmployees -- RetiredEmployees

What about Intersection ?

- It is a derived operator
- $R1 \cap R2 = R1 - (R1 - R2)$
- Also expressed as a join (will see later)
- Example
  - UnionizedEmployees $\cap$ RetiredEmployees

Union Compatible Relations

- Two relations are *union compatible* if
  - Both have same number of columns
  - Names of attributes are the same in both
  - Attributes with the same name in both relations have the same domain
- Union compatible relations can be combined using *union*, *intersection*, and *set difference*

Example

Tables:

Person (*SSN, Name, Address, Hobby*)

Professor (*Id, Name, Office, Phone*)

are <u>not</u> union compatible.

But

π_{Name} (Person) and π_{Name} (Professor)

<u>are</u> union compatible so

π_{Name} (Person) - π_{Name} (Professor)

makes sense.

3. Selection

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
  - $\sigma_{\text{Salary} > 40000}$ (Employee)
  - $\sigma_{\text{name} = \text{“Smith”}}$ (Employee)
- The condition c can be =, <, ≤, >, ≥, <>

Select Operator

- Produce table containing subset of rows of argument table satisfying condition

$$\sigma_{condition}(relation)$$

- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\sigma_{Hobby='stamps'}(Person)$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
9876	Bart	5 Pine St	stamps

Selection Condition

- Operators: $<$, $\leq$, $\geq$, $>$, $=$, $\neq$
- Simple selection condition:
  - <*attribute*> *operator* <*constant*>
  - <*attribute*> *operator* <*attribute*>
- <*condition*> AND <*condition*>
- <*condition*> OR <*condition*>
- NOT <*condition*>

Selection Condition - Examples

- $\sigma_{Id>3000 \text{ OR } Hobby=\text{'hiking'}}(\text{Person})$
- $\sigma_{Id>3000 \text{ AND } Id<3999}(\text{Person})$
- $\sigma_{\text{NOT}(Hobby=\text{'hiking'})}(\text{Person})$
- $\sigma_{Hobby\neq\text{'hiking'}}(\text{Person})$

SSN	Name	Salary
1234545	John	200000
5423341	Smith	600000
4352342	Fred	500000

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

SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

4. Projection

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A1,\ldots,An}(R)$
- Example: project social-security number and names:
  - $\Pi_{SSN,\ Name}$ (Employee)
  - Output schema: Answer(SSN, Name)

SSN	Name	Salary
1234545	John	200000
5423341	John	600000
4352342	John	200000

$\Pi_{Name,Salary}$ (Employee)

Name	Salary
John	20000
John	60000

Project Operator

- Produces table containing subset of columns of argument table

$$\pi_{attribute\ list}(relation)$$

- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\pi_{Name,Hobby}$(Person)

Name	*Hobby*
John	stamps
John	coins
Mary	hiking
Bart	stamps

Project Operator

- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\pi_{Name,Address}$(Person)

Name	*Address*
John	123 Main
Mary	7 Lake Dr
Bart	5 Pine St

Result is a table (no duplicates); can have fewer tuples than the original

Expressions

$\pi_{Id,\ Name}\ (\sigma_{Hobby='stamps'\ \text{OR}\ Hobby='coins'}\ (\text{Person})\ )$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Person

Id	*Name*
1123	John
9876	Bart

Result

5. Cartesian Product

- Each tuple in R1 with each tuple in R2
- Notation: R1 $\times$ R2
- Example:
  - Employee $\times$ Dependents
- Very rare in practice; mainly used to express joins

Cartesian Product Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

EmployeeSSN	Dname
999999999	Emily
777777777	Joe

Employee x Dependents

Name	SSN	EmployeeSSN	Dname
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

Cartesian Product

- If *R* and *S* are two relations, $R \times S$ is the set of all concatenated tuples <*x*,*y*>, where *x* is a tuple in *R* and *y* is a tuple in *S*
  - *R* and *S* need not be union compatible.
  - *But R* and *S* must have distinct attribute names. Why?
- $R \times S$ is expensive to compute. But why?

A	*B*
x1	x2
x3	x4

R

C	*D*
y1	y2
y3	y4

S

A	*B*	*C*	*D*
x1	x2	y1	y2
x1	x2	y3	y4
x3	x4	y1	y2
x3	x4	y3	y4

$R \times S$

Relational Algebra

- Five operators:
  - Union: $\cup$
  - Difference: -
  - Selection: σ
  - Projection: Π
  - Cartesian Product: $\times$
- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
  - Renaming: ρ

Renaming

- Changes the schema, not the instance
- Notation: $\rho_{B1,\ldots,Bn}(R)$
- Example:
  - $\rho_{LastName,\ SocSocNo}$ (Employee)
  - Output schema:
    Answer(LastName, SocSocNo)

Renaming Example

Employee

Name	SSN
John	999999999
Tony	777777777

$$\rho_{LastName,\ SocSocNo}\ (\textbf{Employee})$$

LastName	SocSocNo
John	999999999
Tony	777777777

Example – Second Method

Transcript (*StudId, CrsCode, Semester, Grade*)

Teaching (*ProfId, CrsCode, Semester*)

$\pi_{StudId,\ CrsCode}$ (Transcript)[*StudId, CrsCode1*]

$\times$ $\pi_{ProfId,\ CrsCode}$(Teaching) [*ProfId, CrsCode2*]

This is a relation with 4 attributes:

StudId, *CrsCode1*, *ProfId*, *CrsCode2*

Natural Join

- Notation: R1 $|\times|$ R2
- Meaning: R1 $|\times|$ R2 = $\Pi_A(\sigma_C(R1 \times R2))$
- Where:
  - The selection σ_C checks equality of all common attributes
  - The projection eliminates the duplicate common attributes

Natural Join Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

SSN	Dname
999999999	Emily
777777777	Joe

Employee $\bowtie$ **Dependents =**

$$\Pi_{\text{Name, SSN, Dname}}(\sigma_{\text{SSN=SSN2}}(\text{Employee} \times \rho_{\text{SSN2, Dname}}(\text{Dependents}))$$

Name	SSN	Dname
John	999999999	Emily
Tony	777777777	Joe

Theta Join

- A join that involves a predicate
- $R1 \mid\times\mid_{\theta} R2 = \sigma_{\theta}(R1 \times R2)$
- Here θ can be any condition

Eq-join

- A theta join where θ is an equality
- $R1 \,|\times|_{A=B}\, R2 \;=\; \sigma_{A=B}(R1 \times R2)$
- Example:
  - Employee $|\times|_{SSN=SSN}$ Dependents

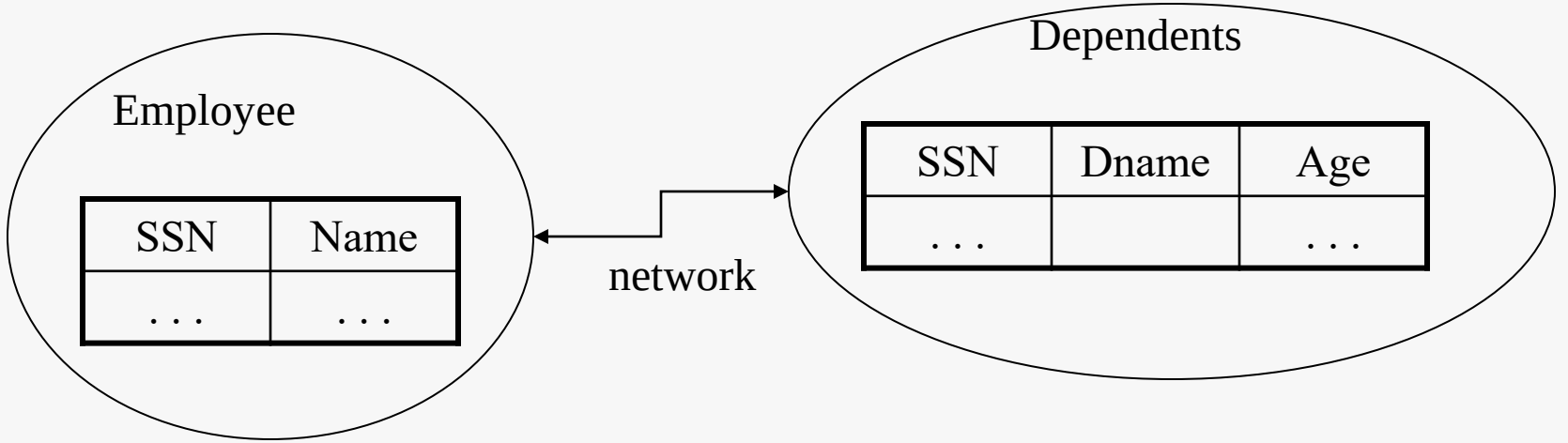
- Most useful join in practice

Semijoin

- $R \mid\times S = \Pi_{A1,\ldots,An} (R \mid\times\mid S)$
- Where $A_1, \ldots, A_n$ are the attributes in R
- Example:
  - Employee $\mid\times$ Dependents

Semijoins in Distributed Databases

- Semijoins are used in distributed databases



$$\text{Employee} \mid\times_{ssn=ssn} (\sigma_{age>71} (\text{Dependents}))$$

$$T = \Pi_{SSN}\, \sigma_{age>71} (\text{Dependents})$$

$$R = \text{Employee} \mid\times T$$

$$\text{Answer} = R \mid\times\mid \text{Dependents}$$

-- Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

  $$_{G1, G2, \ldots, Gn}\mathcal{g}_{F1(A1), F2(A2), \ldots, Fn(An)}(E)$$

  - E is any relational-algebra expression
  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

--- Aggregate Operation – Example 1

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

r

$$\Pi_{g\,\mathrm{sum}(c)}(r) \longrightarrow$$

sum-C
27

--- Aggregate Operation – Example 2

- **Relation *account* grouped by *branch-name*:**

branch-name	*account-number*	*balance*
Dammam	A-102	400
Dammam	A-201	900
Khobar	A-217	750
Khobar	A-215	750
Hafuf	A-222	700

$$T = \textit{branch-name}\ \mathcal{g}\ \textit{sum(balance)}\ (\textit{account})$$

branch-name	*balance*
Dammam	1300
Khobar	1500
Hafuf	700

--- Aggregate Functions: Renaming

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

$$\textit{branch-name}\ \mathcal{G}\ \textbf{\textit{sum}}\textit{(balance)}\ \textbf{\textit{as}}\ \textit{sum-balance}\ (\textit{account})$$

Finally: RA has Limitations !

- Cannot compute “transitive closure”

Name1	Name2	Relationship
Fred	Mary	Father
Mary	Joe	Cousin
Mary	Bill	Spouse
Nancy	Lou	Sister

- Find all direct and indirect relatives of Fred
- Cannot express in RA !!! Need to write C program

SQL to Relational Algebra Conversation

Schema for Student Registration System

Student (*Id, Name, Addr, Status*)
Professor (*Id, Name, DeptId*)
Course (*DeptId, CrsCode, CrsName, Descr*)
Transcript (*StudId, CrsCode, Semester, Grade*)
Teaching (*ProfId, CrsCode, Semester*)
Department (*DeptId, Name*)

Query Sublanguage of SQL

```
SELECT  C.CrsName
FROM  Course C
WHERE  C.DeptId = 'CS'
```

- *Tuple variable* C ranges over rows of Course.
- Evaluation strategy:
  - FROM clause produces Cartesian product of listed tables
  - WHERE clause assigns rows to C in sequence and produces table containing only rows satisfying condition
  - SELECT clause retains listed columns
- Equivalent to: $\pi_{CrsName}\sigma_{DeptId='CS'}(\text{Course})$

Join Queries

```
SELECT  C.CrsName
FROM  Course C, Teaching T
WHERE  C.CrsCode=T.CrsCode AND T.Semester='S2000'
```

- List CS courses taught in S2000
- Tuple variables clarify meaning.
- Join condition "C.*CrsCode*=T.*CrsCode*"
  - relates facts to each other
- Selection condition " T.*Semester*='S2000' "
  - eliminates irrelevant rows
- Equivalent (using natural join) to:

$$\pi_{CrsName}(\text{Course} \bowtie \sigma_{Semester='S2000'}(\text{Teaching}))$$

$$\pi_{CrsName}(\sigma_{Sem='S2000'}(\text{Course} \bowtie \text{Teaching}))$$

Relational Algebra and SQL Exercises

- Professor(ssn, profname, status)
- Course(crscode, crsname, credits)
- Taught(crscode, semester, ssn)

Query 1

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those professors who have taught ‘csc6710’ but never ‘csc7710’.

Relational Algebra Solution

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

$\pi_{ssn}(\sigma_{crscode=`csc6710'}(Taught))-$
$\pi_{ssn}(\sigma_{crscode=`csc7710'}(Taught))$

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
(SELECT ssn
From Taught
Where crscode = 'CSC6710')
EXCEPT
(SELECT ssn
From Taught
Where crscode = 'CSC7710'))
```

Query 2

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those professors who have taught both ‘csc6710’ and ‘csc7710’.

Relational Algebra Solution

Professor(<u>ssn</u>, profname, status)
Course(<u>crscode</u>, crsname, credits)
Taught(<u>crscode, semester</u>, ssn)

$\pi_{ssn}(\sigma_{crscode=\text{`csc6710'} \wedge crscode=\text{`csc7710'}}(\text{Taught})$, wrong!

$\pi_{ssn}(\sigma_{crscode=\text{`csc6710'}}(\text{Taught})) \cap$
$\pi_{ssn}(\sigma_{crscode=\text{`csc7710'}}(\text{Taught}))$, correct!

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT T1.ssn
From Taught T1, Taught T2,
Where T1.crscode = 'CSC6710' AND T2.crscode='CSC7710' AND
T1.ssn=T2.ssn
```

Query 3

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those professors who have never taught ‘csc7710’.

Relational Algebra Solution

Professor(<u>ssn</u>, profname, status)
Course(<u>crscode</u>, crsname, credits)
Taught(<u>crscode, semester</u>, ssn)

$\pi_{ssn}(\sigma_{crscode<>'csc7710'}(Taught))$,

wrong answer!

$\pi_{ssn}(Professor)-\pi_{ssn}(\sigma_{crscode='csc7710'}(Taught))$,

correct answer!

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
(SELECT ssn
From Professor)
EXCEPT
(SELECT ssn
From Taught T
Where T.crscode = 'CSC7710')
```

Query 4

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those professors who taught ‘CSC6710’ and ‘CSC7710” in the same semester

Relational Algebra Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

Relational Algebra Solution

$\pi_{ssn}(\sigma_{crscode1='csc6710'}(Taught[crscode1, ssn, semester]) \bowtie$

$\sigma_{crscode2='csc7710'}(Taught[crscode2, ssn, semester]))$

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT T1.ssn
From Taught T1, Taught T2,
Where T1.crscode = 'CSC6710' AND T2.crscode='CSC7710' AND
T1.ssn=T2.ssn AND T1.semester=T2.semester
```

Query 5

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those professors who taught ‘CSC6710’ or ‘CSC7710” but not both.

Relational Algebra Solution

Professor(<u>ssn</u>, profname, status)
Course(<u>crscode</u>, crsname, credits)
Taught(<u>crscode, semester</u>, ssn)

$$\pi_{ssn}(\sigma_{crscode<>\text{'csc7710'} \vee crscode=\text{'csc7710'}}(Taught)) - (\pi_{ssn}(\sigma_{crscode=\text{'csc6710'}}(Taught)) \cap \pi_{ssn}(\sigma_{crscode=\text{'csc7710'}}(Taught)))$$

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
(SELECT ssn
FROM Taught T
WHERE T.crscode='CSC6710' OR T.crscode='CSC7710')
Except
(SELECT T1.ssn
From Taught T1, Taught T2,
Where T1.crscode = 'CSC6710') AND T2.crscode='CSC7710' AND
T1.ssn=T2.ssn)
```

Query 6

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those courses that have never been taught.

Relational Algebra Solution

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

$\pi_{crscode}(Course) - \pi_{crscode}(Taught)$

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
(SELECT crscode
FROM Course)
EXCEPT
(SELECT crscode
FROM TAUGHT
)
```

Query 7

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those courses that have been taught at least in two semesters.

Relational Algebra Solution

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

$\pi_{crscode}(\sigma_{semester1 <> semester2}($

Taught[crscode, ssn1, semester1] $\bowtie$
Taught[crscode, ssn2, semester2]))

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT T1.crscode
FROM Taught T1, Taught T2
WHERE T1.crscode=T2.crscode AND T1.semester <> T2.semester
```

Query 8

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those courses that have been taught at least in 10 semesters.

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT crscode
FROM Taught
GROUP BY crscode
HAVING COUNT(*) >= 10
```

Query 9

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those courses that have been taught by at least 5 different professors.

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT crscode
FROM (SELECT DISTINCT crscode, ssn FROM TAUGHT)
GROUP BY crscode
HAVING COUNT(*) >= 5
```

Query 10

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return the names of professors who ever taught ‘CSC6710’.

Relational Algebra Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

$\pi_{profname}(\sigma_{crscode='csc6710'}(Taught) \bowtie Professor)$

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT P.profname
FROM Professor P, Taught  T
WHERE P.ssn = T.ssn AND T.crscode = 'CSC6710'
```

Query 11

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return the names of full professors who ever taught ‘CSC6710’.

Relational Algebra Solution

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

$\pi_{profname}(\sigma_{crscode='csc6710'}(Taught) \bowtie \sigma_{status='full'}(Professor))$

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT P.profname
FROM Professor P, Taught  T
WHERE P.status = 'full' AND P.ssn = T.ssn AND T.crscode =
'CSC6710'
```

Query 12

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return the names of full professors who ever taught more than two courses in one semester.

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT P.profname
FROM Professor P
WHERE ssn IN(
SELECT ssn
FROM Taught
GROUP BY ssn, semester
HAVING COUNT(*) > 2
)
```

Query 13

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Delete those professors who never taught a course.

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
DELETE FROM Professor
WHERE ssn NOT IN
(SELECT ssn
FROM Taught
)
```

Query 14

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Change all the credits to 4 for those courses that are taught in f2006 semester.

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
UPDATE Course
SET credits = 4
WHERE crscode IN
(
    SELECT crscode
    FROM Taught
    WHERE semester = 'f2006'
)
```

Query 15

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return the names of the professors who have taught more than 30 credits of courses.

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT profname
FROM Professor
WHERE ssn IN
(
      SELECT T.ssn
      FROM Taught  T, Course C
      WHERE T.crscode = C.crscode
      GROUP BY T.ssn
      HAVING SUM(C.credits) > 30
)
```

Query 16

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return the name(s) of the professor(s) who taught the most number of courses in S2006.

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT profname
FROM Professor
WHERE ssn IN(
    SELECT ssn FROM Taught
    WHERE semester = 'S2006'
    GROUP BY ssn
     HAVING COUNT(*) =
          (SELECT MAX(Num)
           FROM
               (SELECT ssn, COUNT(*) as Num
               FROM Taught
               WHERE semester = 'S2006'
               GROUP BY ssn)
          )
)
```

Query 17

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

List all the course names that professor ‘Smith” taught in Fall of 2007.

Relational Algebra Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

$\pi_{crsname}(\sigma_{profname='Smith'}(Professor) \bowtie \sigma_{semester='f2007'}(Taught) \bowtie Course)$

SQL Solution

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

```
SELECT crsname
FROM Professor P, Taught T, Course C
WHERE P.profname = 'Smith' AND P.ssn = T.ssn AND
T.semester = 'F2007' AND T.crscode = C.crscode
```

Query 18

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

In chronological order, list the number of courses that the professor with ssn ssn = 123456789 taught in each semester.

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT semester, COUNT(*)
FROM Taught
WHERE ssn = '123456789'
GROUP BY semester
ORDER BY semester ASC
```

Query 19

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

In alphabetical order of the names of professors, list the name of each professor and the total number of courses she/he has taught.

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT P.profname, COUNT(*)
FROM Professor P, Taught T
WHERE P.ssn = T.ssn
GROUP BY P.ssn, P.profname
ORDER BY P.profname ASC
```

Query 20

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Delete those professors who taught less than 10 courses.

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
DELETE FROM Professor
WHERE ssn IN(
        SELECT ssn
        FROM Taught
        GROUP BY ssn
        HAVING COUNT(*) < 10
)
```

Query 21

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Delete those professors who taught less than 40 credits.

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
DELETE FROM Professor
WHERE ssn IN(
        SELECT T.ssn
        FROM Taught T, Course C
        WHERE T.crscode = C.crscode
        GROUP BY ssn
        HAVING SUM(C.credits) < 40
)
```

Query 22

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

List those professors who have not taught any course in the past three semesters (F2006, W2007, F2007).

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT *
FROM Professor P
WHERE NOT EXISTS(
        SELECT *
        FROM Taught
        WHERE P.ssn = T.ssn AND (T.semester = 'F2006' OR
        T.semester = 'W2007' OR T.semester='F2007'))
)
```

Query 23

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

List the names of those courses that professor Smith have never taught.

Relational Algebra Solution

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

$\pi_{crsname}$(Course)-
$\pi_{crsname}$($\sigma_{profname='Smith'}$(Professor) ⋈
(Taught) ⋈

Course)

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT crsname
FROM Course C
WHERE NOT EXISTS
      SELECT *
      FROM Professor P, Taught T
      WHERE P.profname='Smith' AND P.ssn = T.ssn AND
T.crscode = C.crscode
)
```

Query 24

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those courses that have been taught by all professors.

Relational Algebra Solution

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

$\pi_{crscode,\ ssn}(Taught) / \pi_{ssn}(Professor)$

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT crscode
FROM Taught T1
WHERE NOT EXISTS(
        (SELECT ssn
        FROM Professor)
        EXCEPT
        (SELECT ssn
         FROM Taught T2
         WHERE T2.crscode = T1.crscode)
)
```

Query 25

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those courses that have been taught in all semesters.

Relational Algebra Solution

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

$\pi_{\text{crscode, semester}}(\text{Taught}) / \pi_{\text{semester}}(\text{Taught})$

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT crscode
FROM Taught T1
WHERE NOT EXISTS(
        (SELECT semester
        FROM Taught)
        EXCEPT
        (SELECT semester
         FROM Taught T2
         WHERE T2.crscode = T1.crscode)
)
```

Query 25

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

Return those courses that have been taught ONLY by junior professors.

Relational Algebra Solution

Professor(ssn, profname, status)
Course(crscode, crsname, credits)
Taught(crscode, semester, ssn)

$\pi_{crscode}(Course) - \pi_{crscode}(\sigma_{status \neq \text{'Junior'}}(Professor) \bowtie Taught)$

SQL Solution

Professor(ssn, profname, status)

Course(crscode, crsname, credits)

Taught(crscode, semester, ssn)

```
SELECT crscode
FROM  Course C
WHERE c.crscode NOT IN(
        (SELECT crscode
         FROM Taught T, Professor P
         WHERE T.ssn = P.ssn AND P.status='Junior'
)
```


Thank
You