CSE 480, Spring 2003

Database Systems

SQL: Structured Query Language

S. Pramanik
σ and π into a Single Statement

• Get student id’s and corresponding names of all students in department with dno=01.

<table>
<thead>
<tr>
<th>Student</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Sname</td>
</tr>
<tr>
<td>s1</td>
<td>john</td>
</tr>
<tr>
<td>s2</td>
<td>Mary</td>
</tr>
<tr>
<td>s3</td>
<td>Rick</td>
</tr>
<tr>
<td>s4</td>
<td>David Maple</td>
</tr>
</tbody>
</table>

• Algebra:

\[ \pi(Sid, Sname) \sigma S\text{dno}='01' (\text{Student}) \]

• SQL:

Select Sid, Sname
From Student
Where Sdno='01'
$\sigma, \pi$ and $\times$ into a Single Statement

• Get student ids and corresponding course numbers and grades for all students in department 01.

<table>
<thead>
<tr>
<th>Student</th>
<th>StudentCourse</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Sname</td>
<td>Saddr</td>
</tr>
<tr>
<td>s1</td>
<td>john</td>
<td>2nd st.</td>
</tr>
<tr>
<td>s2</td>
<td>Mary</td>
<td>4th st.</td>
</tr>
<tr>
<td>s3</td>
<td>Rick</td>
<td>Elm St.</td>
</tr>
<tr>
<td>s4</td>
<td>David Maple</td>
<td></td>
</tr>
</tbody>
</table>

• Algebra:

$$\pi(Sid, cno, Grade)^{\sigma Sdno='01'}(Student \times StudentCourse)$$

• SQL:

Select Sid, cno, Grade
From Student, StudentCourse
Where Sdno='01'
General Structure of SQL Statement

- SQL is the standard for relational query languages.
- Uses the terms
  *table, row, column* instead of relation, tuple and attribute.
- Basic form of sql statements:

  ```
  SELECT <attribute names>
  
  FROM <table List separated by comma>
  
  WHERE <condition>
  ```

- Similarities with relational algebra:
  - SELECT clause in SQL is similar to project \( (\pi) \) in algebra.
  - WHERE clause in SQL is similar to select \( (\sigma) \)
  - FROM clause in SQL is similar to cross product \( \times \) in algebra.
Example SQLs

Tables for Example Queries

Student(Sid, Sname, Saddr, Sdno) : S
StudentCourse(sid, cno, Instr, Grade) : SC
Department(Dno, Dname, Daddr, Chair) : D
Course(Cno, Cname, Cdno) : C

Query: Get names and id’s for those students whose address is East Lansing.

Algebra: \( \pi_{Sname,Sid} \sigma_{Saddr='EastLansing'}(Student) \)

SQL:

```
SELECT Sname, Sid
FROM Student
WHERE Saddr='East Lansing'
```
More Examples

- θ-join: Two table names in the FROM clause, θ condition in the WHERE clause.
  Get all Student names who are in CSE dept.
  \[ \pi_{Sname} \sigma_{Dname="CSE"}(Student \bowtie_{Sdno=Dno} Department) \]
  Select Sname
  from Student, Department
  where Sdno=Dno and Dname="CSE"

- Conceptually cross product.

- Optimized implementation based on θ-join.

- Unspecified WHERE-Clause
  Get all student names in the database
  \[ \pi_{Sname}(Student) \]
  SELECT Sname
  FROM Student

- Projection on all attributes:
  Get all student information in the student table: Student
  SELECT *
  FROM Student
Duplicates in SQL

- Basic relational algebra is set theoretic and assumes no duplicates.
- Tables created by SQL with keys cannot have duplicates.
- Temporary tables created by SQL Select statements may have duplicates. Therefore, these tables may not be treated as sets.

1. Projections create duplicates.
2. Duplicate elimination expensive.
3. Keeping duplicates may be necessary (e.g., computing aggregates).
4. Use DISTINCT to remove duplicates when necessary.

```
SELECT DISTINCT Sname
FROM Student
```

All duplicate names is removed from the result.
Set Operations in SQL

1. SQL standard provides set operation UNION

2. SQL 2 standard provides in addition

   • EXCEPT (set difference).
   • INTERSECT (set intersection).

3. These are set operators and no duplicates in the result.

4. Key word ALL in SELECT keeps duplicates (default is ALL).

5. For set operators, tables have to be UNION compatible.

   Example: Get department numbers for those departments that have no students.

   ALGEBRA: $\pi_{Dno}(Department) - \pi_{Sdno}(Student)$

   ```sql
   SELECT Dno
   FROM Department
   EXCEPT
   SELECT DISTINCT Dno
   FROM Student
   ```
Example: Get names of all students (including duplicates).
Algebra: ??
SQL:

```
SELECT ALL Sname
FROM   Student
```
OR

```
SELECT Sname
FROM   Student
```
Substring Comparisons and Ordering

1. Part of an attribute value can be specified in the WHERE clause. The results can be ordered by a set of attribute values.

\textit{LIKE, \%, -, ORDER BY}

Example: Retrieve all student names whose address is East Lansing and the last 4 digits of their tel. no. are 4341. Retrieve the names in alphabetical order.

ALGEBRA: ?

SQL:

\begin{verbatim}
SELECT Sname
FROM Student
WHERE Saddr LIKE '%East Lansing%' and TelNO=- -4341
ORDER BY Sname
\end{verbatim}
Renaming Table Names and Attribute Names

- Renaming table names and column names on the fly.
- Useful for
  - Self Joins
  - Give different column names for result table.
  - Short names for tables.

Example:

```sql
SELECT Sid AS StudentID,  
      Sname as studentName
FROM Student AS S
WHERE S.Addr="East Lansing"
```
OUTER Join

• Rows not participating in the join are also included in the result.

• Missing values are nulls

<table>
<thead>
<tr>
<th>Faculty</th>
<th>Department</th>
<th>Join</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fid</td>
<td>Fname</td>
<td>worksfor</td>
</tr>
<tr>
<td>F1</td>
<td>John</td>
<td>D1</td>
</tr>
<tr>
<td>F2</td>
<td>Mary</td>
<td>D1</td>
</tr>
<tr>
<td>F3</td>
<td>Rick</td>
<td>null</td>
</tr>
<tr>
<td>F4</td>
<td>David</td>
<td>D3</td>
</tr>
</tbody>
</table>

• Join:

Select Fname as Faculty, Dname as Department
From Faculty, Department
where worksfor = Dno

• Left Outer Join:

Select Fname as Faculty, Dname as Department
From Faculty, Department
where worksfor (+)= Dno
- **Right Outer Join:**

  Select Fname as Faculty, Dname as Department
  From Faculty, Department
  where worksfor = Dno (+)

- **Outer Join (both left and right):**

  Select Fname as Faculty, Dname as Department
  From Faculty, Department
  where worksfor (+)= Dno (+)

<table>
<thead>
<tr>
<th>LeftOuterJoin Faculty</th>
<th>Department</th>
<th>RightOuterJoin Faculty</th>
<th>Department</th>
<th>OuterJoin Faculty</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>CSE</td>
<td>John</td>
<td>CSE</td>
<td>John</td>
<td>CSE</td>
</tr>
<tr>
<td>Mary</td>
<td>CSE</td>
<td>Mary</td>
<td>CSE</td>
<td>Mary</td>
<td>CSE</td>
</tr>
<tr>
<td>David</td>
<td>ME</td>
<td>David</td>
<td>ME</td>
<td>David</td>
<td>ME</td>
</tr>
<tr>
<td>Rick</td>
<td>null</td>
<td>null</td>
<td>EE</td>
<td>Rick</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>EE</td>
</tr>
</tbody>
</table>
Aggregate Functions, ORDER BY and GROUP BY

• None in Relational Algebra.

• Aggregate functions: SUM, MAX, MIN, AVG on selected columns.

• COUNT counts the number of rows.

• Get count, minimum, maximum and average grade for CSE 480.

    SELECT COUNT(*), MIN(Grade),
          MAX(Grade), AVG(Grade)
    FROM StudentCourse
    WHERE cno=’CSE480’

• Find the same for all courses???

• Needs a GROUP BY
GROUP BY & ORDER BY Clause

• Get count, maximum, minimum, and average grades for all courses.

    SELECT cno,COUNT(*) ,SUM(Grade), MIN(Grade) , AVG(Grade)
    FROM StudentCourse
    GROUP BY cno

• Syntax of Group By Clause:

    SELECT ???
    FROM ???
    WHERE ???
    GROUP BY col name
    HAVING condition
    ORDER BY Col name ASC , Col Name DESC
Use of WHERE and HAVING

- WHERE clause selects the larger group.
- HAVING clause selects particular groups.
- Get count, maximum, minimum, and the average grades of those courses offered by CSE Department and enrollment > 40.

```sql
SELECT cno, COUNT(*), SUM(Grade),
      MIN(Grade), AVG(Grade)
FROM SC, C, D
WHERE cno=Cno & Cdno=Dno & Dname='CSE'
GROUP BY cno
HAVING COUNT(*) > 40
```

- Can GROUP BY be implemented by SQL without GROUP BY?
Nested Queries (Sub Queries)

- Multiple **SQL Selects** in a single SQL statement.
- One Outer Select
- Subselects inside WHERE or HAVING of the Outer Select.
- Result of subselect can be used
  - as a single value
    Comparison operators $=, >, <, etc.$
  - as a single row
    Comparing a single resulting row.
  - as a set
    Set operators: IS IN, NOT IN.
● Example:

Select sid  
From StudentCourse  
Where cno=cse480 and grade>(Select AVG(grade)  
From StudentCourse  
where cno=cse480)

Select Sid, Sname  
From Student  
Where Sid IS IN (Select sid  
From StudentCourse  
Where cno=CSE480)

● Outer select? Subselect?

● Single valued, set valued subselect?

● Relational algebra for the second:

$$\pi_{Sid,Sname}\sigma_{Sid=sid\&cno='CSE480'}(Student \times StudentCourse)$$

● Select Sid, Sname  
From Student, StudentCourse  
Where Sid=sid \& cno='CSE480'
More Examples:

- Names of students in CSE and those in CSE480, not necessarily in CSE.

- Algebra:

\[
\left( \pi sname \sigma Dname = "CSE" \right) \cup \\
\left( \pi sname \sigma cno = "CSE480" \right) 
\]
• SQL:

1. Without Nesting:

   SELECT DISTINCT Sname
   FROM Student, Department
   WHERE Sdno=Dno and Dname="CSE"
   UNION
   SELECT DISTINCT Sname
   FROM Student, StudentCourse
   WHERE Sno=sno and cno="CSE480"

2. With Nesting:

   SELECT DISTINCT Sname
   FROM S
   WHERE Sno IN
     (SELECT Sno
      FROM S,D
      WHERE Sdno=Dno and Dname="CSE")
   OR Sno IN
     (SELECT Sno
      FROM S,SC
      WHERE Sno=sno and cno="CSE480")
Another example:

- Get all students in CSE480 and not in CSE department.

- Algebra:

\[ \pi_{\text{name}} \sigma_{\text{cno}="CSE480"}(\text{Student} \bowtie \text{StudentCourse}) - \]
\[ \pi_{\text{name}} \sigma_{\text{Dname}="CSE"}(\text{Student} \bowtie \text{Department}) \]
**SQL:**

1. **Without nesting:**

   ```sql
   SELECT DISTINCT Sname
   FROM Student, StudentCourse
   WHERE Sno=sno and cno="CSE480"
   DIFFERENCE
   SELECT DISTINCT Sname
   FROM Student, Department
   WHERE Sdno=Dno and Dname="CSE"
   ```

2. **With Nesting:**

   ```sql
   SELECT DISTINCT Sname
   FROM Student
   WHERE Sno IN
   (SELECT Sno
   FROM Student, StudentCourse
   WHERE Sno=sno and cno="CSE480")
   AND Sno NOT IN
   (SELECT Sno
   FROM Student, Department
   WHERE Sdno=Dno and Dname="CSE")
   ```

**Why Nesting?** Can we do the same without nesting?
Correlated Nested Queries

- Tables of outer queries referenced in nested query.
- Conceptually, nested query is evaluated each time WHERE clause of Outer query is evaluated.
- Get names of those students who have taken at least five courses.

  - Algebra: ??
  - SQL:

    ```
    SELECT Sname
    FROM S
    WHERE (SELECT COUNT(*)
             FROM SC
             WHERE sno=S.Sno) >= 5
    ```

    - S.Sno refers to table S in the outer Query.
EXISTS and NOT EXISTS

- EXISTS (Select From Where): True when result of Select–From not empty.
- EXISTS(\(Q\)): True for subquery, \(Q\), result not empty.
Names of students taking at least one CSE course.

- Algebra:

\[ \pi_{sno} \sigma_{Dname="CSE"}(S \bowtie_{S.Sno=SC.sno} SC \bowtie_{SC.cno=C.Cno}
\quad C \bowtie_{C.dno=D.Dno} D) \]

- SQL:

```
SELECT Sname
FROM S
WHERE EXISTS (
    SELECT *
    FROM SC
    WHERE Sno=sno and
    Cno IN
    (SELECT Cno
    FROM C,D
    WHERE Dno=Cdno
    and Dname="CSE")
)
```

- Without NESTING:

```
SELECT Sname
FROM S, SC, C, D
WHERE Sno=sno and cno=Cno and Cdno=dno
    and Dname="CSE"
```
• NOT EXISTS(Q):
  TRUE for empty subquery result.
Example: Names of all students not taking any CSE courses

• Algebra:

\[ \pi_{Sno}(S) - \pi_{sno} \sigma_{\text{Sname}="CSE"}(SC \bowtie_{SC.cno=C.cno} C \bowtie_{C.dno=D.dno} D) \]

• SQL:

```sql
SELECT Sname
FROM S
WHERE NOT EXISTS
  ( SELECT *
    FROM SC
    WHERE Sno=sno and
    Cno IN
      (SELECT Cno
       FROM C,D
       WHERE Dname="CSE" ) )
```
• Can EXISTS and NOT EXISTS be implemented by other operations discussed before (e.g. IN, NOT IN)?
Comparing Two Sets

CONTAINS: set $Q_1$ CONTAINS set $Q_2$?

where $Q_1$ and $Q_2$ represent results of two SQL statements.

$Q_1$ CONTAINS $Q_2 \Rightarrow setQ_1 \subseteq setQ_2$

CONTAIN is not supported by sql (it was proposed in the original SQL in SYSTEM R of IBM)

1. Example: Get all students who are taking at least all those courses that John is taking

Use CONTAINS

SELECT Sname
FROM S AS Sx
WHERE (SELECT Cno
     FROM SC
     WHERE Sx.Sno=SC.sno)

CONTAINS

(SELECT Cno
 FROM SC, S
 WHERE Sno=sno and Sname="John" )
2. Intuitive justification: There are three SELECT statements (A), (B) and (C). (C) gets all courses taken by John. (B) gets all courses taken by Student Sx. At this point you have to realize that Sx is a variable. (A) gets all students in Sx such that set B CONTAINS set (C).

Since SQL does not support CONTAINS, we will implement the above by EXISTS and NOT EXISTS.
Comparing Two Sets Using EXISTS and NOT EXISTS

1. Rephrase the above query as follows:
   Select each student such that there does not exit a course that John is taking and the student does not take the course.

   SELECT Sname                           (A)
   FROM S AS Sx
   WHERE NOT EXISTS (SELECT cno           (B)
                    FROM SC   AS SCx
                    WHERE cno IN
                    (SELECT cno   (C)
                     FROM SC, S
                     WHERE Sno=sno and
                     Sname="John")
                    AND

   NOT EXISTS
   (SELECT *                           (D)
    FROM SC
    WHERE Sx.Sno=sno and
    cno=SCx.cno)

2. Intuitive justification: There are four SQL statements,
(A), (B), (C) and (D). (C) is getting all courses John is taking. (D) is getting tuples with SCX courses that Student Sx (a variable on rows of Student table) is taking. (B) is getting those courses that John is taking but the student Sx is not. If that is the case NOT EXISTS is false and that Sx is not in the result, otherwise Sx is in the result.
Insert, Delete, Update

Insert:
Three Types:

- Single tuple insert:

  \[
  \text{INSERT INTO Student(Sid, Sname) VALUES (12345, "Richard")}
  \]

- Multiple Tuple Insert:
  Must be from another table.

Create Table GoodStudents(
  Sid Number(10),
  Sname varchar(30));
Insert into GoodStudents(Sid,Sname)
  Select Sid, Sname
  From Student X
  Where 3.0 < (Select AVG(Grade)
              From StudentCourse Y
              Where X.Sid=Y.Sid);

- Data Loading from Files:
  Bulk data loaded from a file using OS commands

Note: Columns Defined NOT NULL Must Have Values.
Delete:

- Deletes Tuples From a Table.
  Example: Delete all students from Student table whose address is "East Lansing".

  \[
  \text{DELETE FROM Student} \\
  \text{WHERE Saddr="East Lansing"}
  \]

Update:

- Update Statement:
  Increase grades of every student in CSE480 by 10%.

  \[
  \text{UPDATE StudentCourse} \\
  \text{SET Grade=1.1*Grade} \\
  \text{WHERE sno IN} \\
  \text{(SELECT sno} \\
  \text{FROM StudentCourse, Course} \\
  \text{WHERE cno=Cno and Cname="CSE480"}}
  \]

- Assign Values from Another Table:
  Set Student Grade to the grades in table tmp(Sid,Grade)

  \[
  \text{Update StudentCourse X} \\
  \text{Set Grade=(Select y.Grade} \\
  \text{From Tmp Y} \\
  \text{Where X.Sid=Y.Sid)}
  \]
PL/SQL

- PL/SQL: Oracle’s Procedural Language Extension of SQL
- It is a Block-structured Language:

  [Declare
    ---- declarations ]
BEGIN
    ---- statements
[EXCEPTION
    ---- handlers]

- Following selects the name and address of the student with student id=s1234, and outputs them using PL/SQL dbms_output.put_line.

```
declare
    v_sid Student.sid%type;
    v_sname Student.sname%type;
    v_saddr Student.saddr%type;
begin
    select sname, saddr
    into v_sname, v_saddr
    from Student
```
where sid='s1234';
dbms_output.put_line('NAME:' || v_sname);
dbms_output.put_line('Address:' || v_saddr);

exception

when others then

dbms_output.put_line('Error detected');
end;
/

Following checks in PL/SQL if the average grade of cse 880 is greater than 3.5. If it is, outputs it using PL/SQL dbms_output.put_line.

declare

AverageGrade number;
begin

update StudentCourse

set grade = grade * 1.1
where cno = 'CSE880';

select avg(grade)
into AverageGrade
from StudentCourse
where cno = 'CSE880';

if AverageGrade>3.5 then

dbms_output.put_line('Average grade is ' || AverageGrade);

end if;
end;
/
end if;
commit;
exception
when others then
    dbms_output.put_line('Error in grade update');
end;
/

• When the result of a select statement contains more than one tuple, use of cursor is necessary.

Following outputs within a PL/SQL LOOP the Course numbers and Average Grades of CSE courses with average grades ≥ 3.5.

declare
    v_AverageGrade number(10);
    v_CourseNo varchar2(10);
cursor AverageGrade is
    select cno, avg(grade)
    from StudentCourse
    where cno like '%CSE%
    group by cno
    having avg(grade)>3.5;
begin
    open AverageGrade;

36
loop
    fetch AverageGrade into v_CourseNo, v_AverageGrade;
    exit when AverageGrade%notfound;
    dbms_output.put_line(’Course number is ’||v_CourseNo);
    dbms_output.put_line(’Average grade is ’||AverageGrade);
end loop;

close AverageGrade;
exception
    when others then
        dbms_output.put_line(’Error detected’);
end;