Scalar Q2 Execution

Students

SID	Name	Class	Major
123	John	3	CS
124	Mary	3	CS
126	Sam	2	CS
999	Newman	1	CS
132	Erin	2	EE

Enroll

SID	CID	Term	Grade
123	CS1520	Fall 10	3.75
124	CS1520	Fall 10	4
126	CS1520	Fall 10	3
123	CS1555	Fall 10	4
124	CS1555	Fall 10	NULL
126	CS1550	Spring 11	NULL

Q2 RESULT

SID	LG
123	4
124	4
126	3
999	

- Q2 is equivalent to a single-level SQL query; which one?

Equivalent Query to Q2

Q2 Equivalent Query:

```
SELECT SID, MAX(Grade) AS LG
FROM STUDENT S NATURAL LEFT JOIN ENROLL E
WHERE S.Major = 'CS'
GROUP BY SID;
```

Set Membership $x \in A$

- The IN and NOT IN operators check for simple membership.
- LIBRARIAN (SSN, Name, City, Gender, Salary, SNO);
  SECTION (SNO, Name, HeadSSN);

  Q1: List each head librarian's SSN, along with their section, except those of the Science & Art sections.

```
SELECT HeadSSN, Name
FROM   SECTION
WHERE  Name NOT IN ('Science', 'Art ');
```

Set Membership

LIBRARIAN (<u>SSN</u>, Name, City, Gender, Salary, SNO);
SECTION (<u>SNO</u>, Name, HeadSSN);
DEPENDENT (<u>Name, LIBSSN</u>, DSSN, Gender, DoB);

- Q2:. ? Lists librarian's name who have dependents of the same gender

```
SELECT L.Name
FROM   LIBRARIAN AS L
WHERE  L.SSN IN (SELECT LIBSSN
                 FROM   DEPENDENT D
                 WHERE  D.LIBSSN = L.SSN AND
                        D.Gender = L.Gender);
```

Set Membership & Comparisons

- Test for membership on other comparisons:
  (=, <>, >, >=, <, <=)
- They can be quantified using ANY (i.e., SOME) or ALL.
- Q: List the SSN of all head librarians whose salary is lower than that of any librarian who is not a head librarian.

```
SELECT H.SSN
FROM   LIBRARIAN H JOIN SECTION ON SSN = HeadSSN
WHERE  H.Salary < ANY (SELECT L.Salary
                       FROM   LIBRARIAN L
                       WHERE  L.SSN NOT IN  (SELECT HeadSSN
                                             FROM SECTION ));
```

Set Comparisons: Unique & Empty

- The UNIQUE and NOT UNIQUE operators test for duplicates in a result (i.e, tests for a set and a bag, respectively)
- The EXISTS (not empty) and NOT EXISTS (empty) operators test for emptiness of a result
- Q: ?   Lists non-head librarian

```
SELECT L.SSN, L.Name
FROM   LIBRARIAN L
WHERE  NOT EXISTS (SELECT *
                   FROM SECTION
                   WHERE L.SSN = HeadSSN);
```

Set Comparisons: Unique & Empty…

- Q: ?   Lists the students with dual major CS and Math

```
SELECT S.SID
FROM   STUDENT S
WHERE  NOT UNIQUE ( SELECT *
                    FROM (SELECT  SID
                          FROM    STUDENT
                          WHERE   Major = 'CS' ) UNION ALL
                         (SELECT  SID
                          FROM    STUDENT
                          WHERE   Major = 'Math' )
                    WHERE  S.SID = SID
                    );
```

Unique & Not Unique

- UNIQUE in POSTGRES & Oracle is used for constraints or instead of DISTINCT
  - Always use DISTINCT
- UNIQUE & Not UNIQUE can be expressed using JOIN
- Q: ?   Lists students with dual major only once

```
SELECT C.SID
FROM   Student C JOIN student M ON
         (C.SID = M.SID AND C.Major > M.Major);
```

Unique in Postgres/Oracle

```
SELECT S.SID
FROM STUDENT S
WHERE NOT EXISTS(SELECT S.SID
                 FROM ((SELECT SID
                        FROM STUDENT
                        WHERE Major = 'CS')
                       UNION ALL
                       (SELECT SID FROM STUDENT WHERE
                        Major = 'Math')
                       ) A
                 WHERE S.SID = SID
                 GROUP BY S.SID
                 HAVING count(*) > 1
  );
```

Removing Duplicates

- Q: List all the students who are double majors with their majors only once.
- Note that (1, CS, Math) and (1, Math, CS) are duplicates
- A:
```
SELECT C.SID, C.Major, M.Major
  FROM Student C JOIN student M ON
       (C.SID = M.SID AND C.Major > M.Major);
```
- Example:

Student

SID	*Major*
123	CS
123	Math

Student C JOIN Student M

C.SID	*C.Major*	*M.SID*	*M. Major*
~~123~~	~~CS~~	~~123~~	~~CS~~
~~123~~	~~CS~~	~~123~~	~~Math~~
123	**Math**	123	**CS**
~~123~~	~~Math~~	~~123~~	~~Math~~

⇐ Result in bold

Challenging yet Common Query

- Assume ENROLL(SID, CID, score)
- Find the ranking of students in CS2550 according to their scores. Your results should consider the case of tie.
  - E.g., output when 007 & 009 both received the highest score.

SID	Rank
007	1
009	1
003	3
005	4

Ranking Query using of Scalar Subquery

- ENROLL(SID, CID, score)
- Q: Find the ranking of students in CS1555 according to their scores. Your results should consider the case of tie.

```
SELECT S.SID,  (1 + (SELECT COUNT(*)
                     FROM ENROLL E
                     WHERE E.CID = 'CS 1555' AND
                           E.score > S.score)
                ) AS Rank
FROM ENROLL S
WHERE S.CID = 'CS 1555'
ORDER BY Rank;
```

Limiting result rows

- Save resources, speed-up result
- FETCH clause (SQL 2008)

  OFFSET *start* { ROW | ROWS }
  FETCH { FIRST | NEXT } [*count*] { ROW | ROWS } ONLY

  - the OFFSET clause must come before the FETCH clause
  - *start:* number of rows to skip [default 0]
  - *count:* maximum number of rows to return [default 1]
  - *ROW, ROWS and NEXT is "noise"* keywords – have no effect
- E.g.,

```
SELECT *
FROM   STUDENT
FETCH FIRST 10 ROWS ONLY;
```

Top-K Queries

- Q1: ? Ten "oldest" students

```
SELECT *
FROM   STUDENT
ORDER BY SID ASC
FETCH FIRST 10 ROWS ONLY;
```

- Q2: ? Ten students with lowest QPA

```
SELECT *
FROM   STUDENT
ORDER BY QPA ASC
FETCH FIRST 10 ROWS ONLY;
```

- How many tuples does Q2 return?

Next-K Queries

- Q3: ? List students with lowest QPA rank between 7-16

```
SELECT *
FROM   STUDENT
ORDER BY QPA ASC
OFFSET 6 ROW
FETCH NEXT 10 ROWS ONLY;
```

- Q3: (alternative that seems to work)

```
SELECT *
FROM   STUDENT
ORDER BY QPA ASC
OFFSET 6 ROW
FETCH FIRST 10 ROWS ONLY;
```

Limits (No Standard Syntax)

- `SELECT * FROM T WHERE ROWNUM <= 10;`
  - Oracle (also supports the standard ?)
- `SELECT * FROM T LIMIT 10 OFFSET 20;`
  - MySQL, PostgreSQL (supports the standard), SQLite
- `SELECT * FROM T`
  `WHERE ID_T > 10 FETCH FIRST 10 ROWS ONLY;`
  - IBM DB2
- `SELECT TOP 10 * FROM T;`
  - MS SQL Server, Sybase ASE

Top-K in Oracle

- Q: Select the top 3 students with the highest QPA

```
SELECT *
FROM (SELECT * FROM STUDENT ORDER BY QPA DESC) S
WHERE rownum <= 3
ORDER BY rownum;
```

- ROWNUM: pseudocolumn which returns a number showing the order in which Oracle selects a row from a table
- Q: Select top students with offset = 3, next = 3

```
??
```

Top-K in Oracle

- Q: Select top students with offset = 3, next = 3
- What about using **BETWEEN**?

```
SELECT *
FROM (SELECT * FROM STUDENT ORDER BY QPA DESC)
WHERE rownum BETWEEN 3 and 6;
```

Or

```
SELECT *
FROM (SELECT * FROM STUDENT ORDER BY QPA DESC)
WHERE rownum > 3 AND rownum <=6;
```

- Does not work because `rownum` is assigned dynamically

Next Top-K in Oracle

- Q: Select top students with offset = 3, next = 3

```
SELECT *
FROM (
  SELECT SID, Name, Major, QPA, rownum AS snum
  FROM (SELECT * FROM STUDENT ORDER BY QPA DESC)
  WHERE rownum <=6)
 WHERE snum > 3;
```

- Consider QPAs: 1, 3, 6, 8, 2, 12, 19, 9

Top-K in Postgres

```
SELECT *
FROM (
       SELECT *,
              ROW_NUMBER() OVER (ORDER BY sid)
       FROM cs1555.student
     ) x
WHERE ROW_NUMBER <=3;
```

Structured Query Language
SQL – DML

◆ Update Operations

Query Language with Update Statements?

❑ Is it a historical accident?

or Select is in the hard of updates!

- *Selections* are expressed by the WHERE clause!

"They're sure not making Champaign bottles the way they used to."

Update Tuples

❑ Update can apply to a single relation

❑ Updates all the selected tuples by the condition in the WHERE-clause

❑ Examples:

```
UPDATE STUDENT
SET Name = 'Kathy Jones'
WHERE SID = 165;
```

```
UPDATE STUDENT
SET Major =  'CS'
WHERE  DNO IN
      ( SELECT DNUM
        FROM DEPT
        WHERE Dname = 'CS' );
```

Delete Tuple

❑ Delete removes all selected tuples by the condition in the WHERE-clause

❑ Examples:

```
DELETE  FROM STUDENT
WHERE SID = 165;
```

```
DELETE  FROM STUDENT
WHERE Name =  'John' ;
```

```
DELETE  FROM STUDENT;
```

```
DELETE  FROM  STUDENT
WHERE  DNO IN
      ( SELECT DNUM
        FROM DEPT
        WHERE Dname = 'CS' );
```

Insert

STUDENT(SID,Name,Major);

- Two forms: **Implicit** (list) and **Explicit** (set)
- **Implicit**:

```
INSERT INTO STUDENT
VALUES (165, 'Susan', 'CS');
```

- **Explicit**:

```
INSERT  INTO  STUDENT (SID, Name)
VALUES (165, 'Susan Jones');

INSERT  INTO  STUDENT (Name, SID)
VALUES ( 'Susan Jones', 165);
```

Derived Insert Values

- Tuples are derived using SELECT
- Useful to populate a table in the database from data already in the database
- E.g.,

```
INSERT  INTO Dept_Info (Dept_Name, Num_Students)
     SELECT Major, Count(*)
     FROM STUDENT
     GROUP  BY  Major;
```