CSE 180: Lab Section 1

- Recap (Students, Courses, Enrolments)
- Subquery
- IN & NOT IN
- EXISTS vs NOT EXISTS
- Difference between IN and EXISTS
- UNION
- INTERSECT
- EXCEPT
- ALL
- ANY

Tuesday, October 24th

Utkarsh Gupta

Student, Courses, & Enrolment

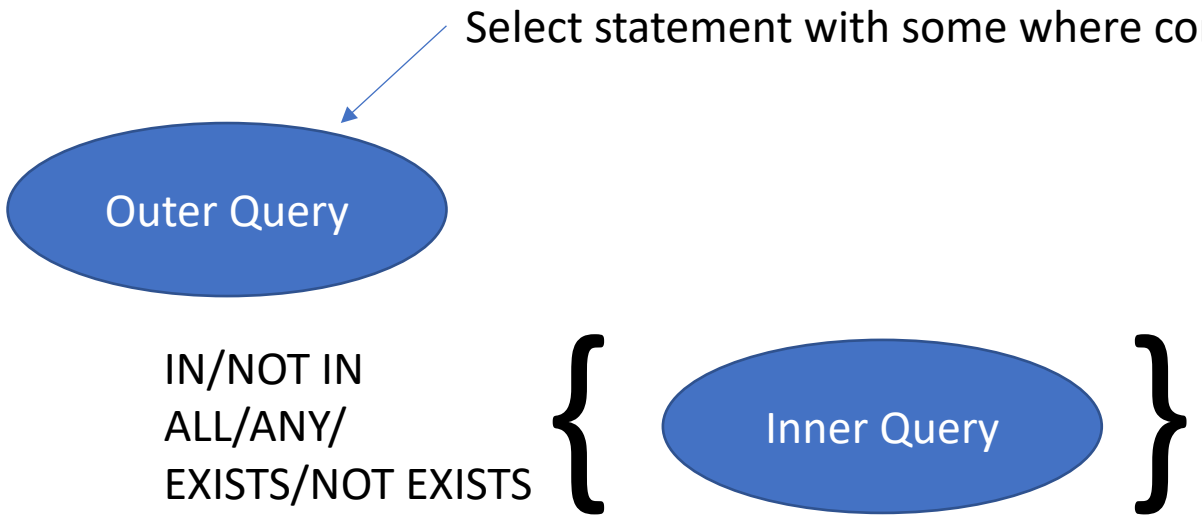
```
utgupta=# select * from student;
 s_id | s_name |    ssn     |    dob     | gpa  | has_grad |   phone    |      email
------+--------+------------+------------+------+----------+------------+-----------------
    1 | DAVID  | 1234       | 1997-01-01 | 3.80 | f        | 0123456    | david@pqrs.com
    2 | JULIA  | 4567       | 2000-02-18 | 3.90 | f        | 1234589    | julia@pqrs.com
    3 | DAVID  | 2468       | 2000-02-01 | 3.80 | f        | 9827123    | david1@pqrs.com
    4 | JOEL   | 3412       | 2002-03-23 | 3.34 | f        |            | joel@pqrs.com
    5 | ABY    | 4321       | 2000-01-01 | 3.32 | t        |            |
(5 rows)
```

```
utgupta=# select * from courses;
 c_id |   c_name    | credits
------+-------------+---------
  180 | DB 1        |       5
  181 | DB 2        |       5
  215 | DB GRAD 1   |       5
  160 | CG          |       5
  280 | CV SEM      |       2
(5 rows)
```

```
utgupta=# select * from enrolment;
 s_id | c_id | dropped
------+------+---------
    1 |  180 | f
    5 |  160 | f
    5 |  215 | f
    1 |  181 | t
    2 |  160 | f
    5 |  180 | f
    2 |  280 | f
    5 |  181 | f
    2 |  181 | t
    3 |  180 | f
    5 |  280 | f
(11 rows)
```

Subquery

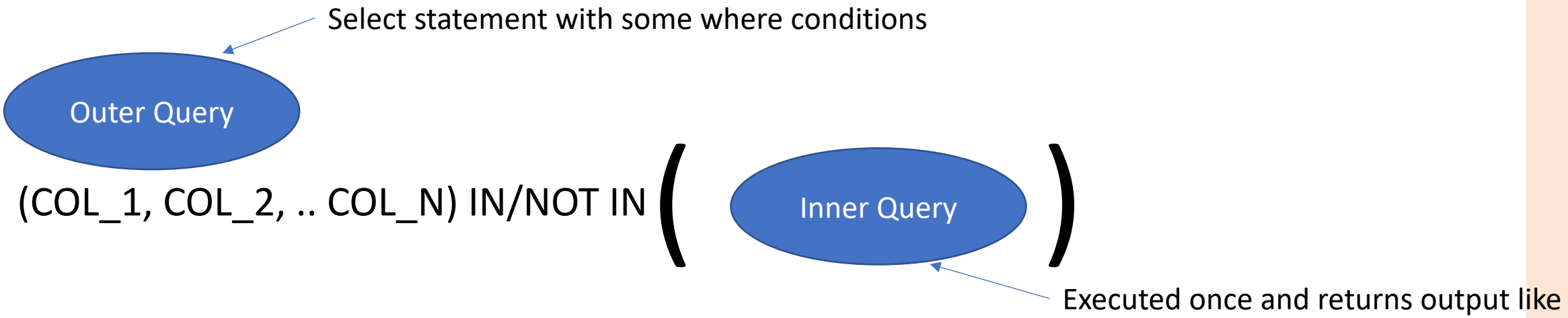
- A query that is run inside a query is know as subquery.
- Similar to nested for loops in the procedural programming languages, for instance, C, C++, etc.



- It is possible to have multiple nested subqueries.

IN & NOT IN operators

- Tells whether a row from the outer table is INCLUDE/NOT INCLUDED in the result of the inner query.
- Execution of IN operator



- After the execution of Inner query, every row from the tables of the outer query is matched against the result of the inner query.

IN & NOT IN operators

- Find the name of the students who are enrolled in a course.

SELECT s_name FROM student S, enrolment E

WHERE S.s_id = E.s_id;

```
utgupta=# SELECT s_name FROM student S, enrolment E
utgupta-# WHERE S.s_id = E.s_id;
 s_name
--------
 DAVID
 ABY
 ABY
 DAVID
 JULIA
 ABY
 JULIA
 ABY
 JULIA
 DAVID
 ABY
(11 rows)
```

IN & NOT IN operators (cont.)

- Find the name of the students who are enrolled in a course.

```
SELECT DISTINCT s_name
FROM student S, enrolment E
WHERE S.s_id = E.s_id;
```

```
utgupta=# SELECT DISTINCT s_name
utgupta-# FROM student S, enrolment E
utgupta-# WHERE S.s_id = E.s_id;
 s_name
--------
 ABY
 JULIA
 DAVID
(3 rows)
```

Be careful when you are using DISTINCT

IN & NOT IN operators (cont.)

- Find the name of the students who are enrolled in a course.

```
SELECT s_name
FROM student
WHERE s_id IN (SELECT s_id FROM enrolment);
```

```
utgupta=# SELECT s_name
utgupta-# FROM student
utgupta-# WHERE s_id IN (SELECT s_id FROM enrolment);
 s_name
--------
 DAVID
 JULIA
 DAVID
 ABY
(4 rows)
```

IN & NOT IN operators (cont.)

- Find the name of the students who are **NOT** enrolled in a course.

SELECT s_name

FROM student

WHERE s_id **NOT IN** (SELECT s_id FROM enrolment);

```
utgupta=# SELECT s_name
utgupta-# FROM student
utgupta-# WHERE s_id NOT IN (SELECT s_id FROM enrolment);
 s_name
--------
 JOEL
(1 row)
```

IN & NOT IN operators (cont.)

- Get the names of students who enrolled in DB 1 but not DB 2:

```
SELECT s_name
  FROM Student
WHERE s_id IN ( SELECT e.s_id
                  FROM Enrolment e, Courses c
                 WHERE e.c_id = c.c_id
                   AND c.c_name= 'DB 1'
                   AND e.dropped = FALSE )
    AND s_id NOT IN ( SELECT e.s_id
                        FROM Enrolment e, Courses c
                       WHERE e.c_id = c.c_id
                         AND c.c_name = 'DB 2' );
```

```
utgupta=# SELECT s_name
utgupta-#   FROM Student
utgupta-# WHERE s_id IN ( SELECT e.s_id
utgupta(#                   FROM Enrolment e, Courses c
utgupta(#                  WHERE e.c_id = c.c_id
utgupta(#                    AND c.c_name= 'DB 1'
utgupta(#                    AND e.dropped = FALSE )
utgupta-#       AND s_id NOT IN ( SELECT e.s_id
utgupta(#
utgupta(#                   FROM Enrolment e, Courses c
utgupta(#
utgupta(#                  WHERE e.c_id = c.c_id
utgupta(#                    AND c.c_name = 'DB 2' );
 s_name
--------
 DAVID
(1 row)
```

IN & NOT IN operators (cont.)

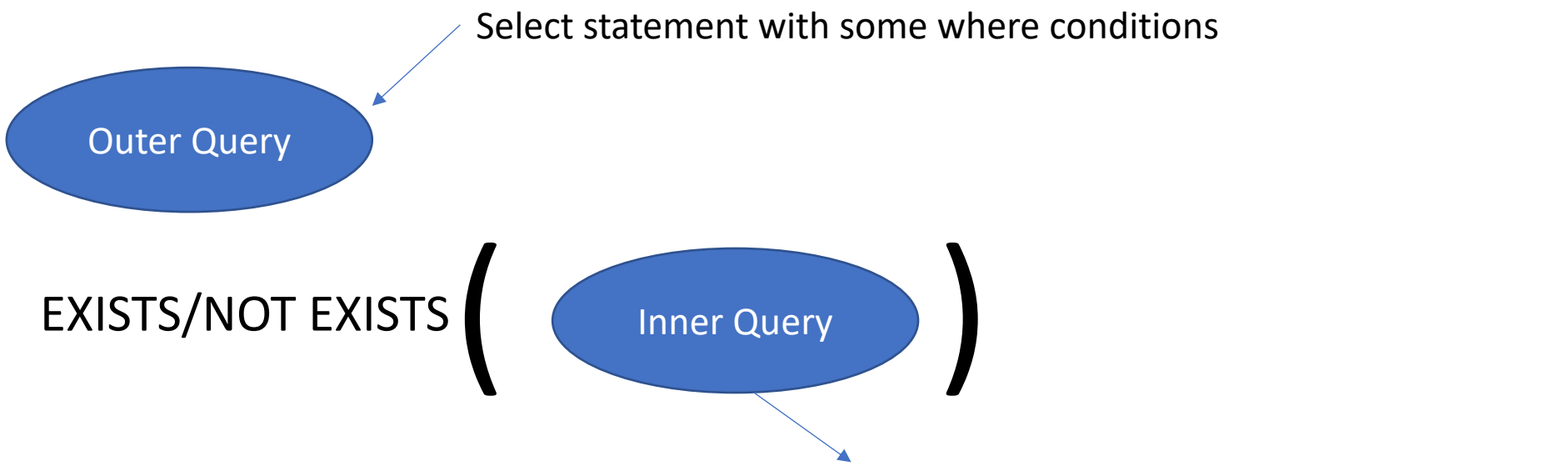
- Get the names of students who have not enrolled in any DB courses:

```
SELECT s_name
FROM Student
WHERE s_id NOT IN ( SELECT s_id
                    FROM Enrolment e, Courses c
                   WHERE e.c_id = c.c_id
                     AND c.c_name LIKE 'DB%');
```

```
utgupta=# SELECT s_name
utgupta-# FROM Student
utgupta-# WHERE s_id NOT IN ( SELECT s_id
utgupta(#                       FROM Enrolment e, Courses c
utgupta(#                      WHERE e.c_id = c.c_id
utgupta(#                        AND c.c_name LIKE 'DB%');
 s_name
--------
 JOEL
(1 row)
```

EXISTS & NOT EXISTS operators

- The EXISTS and NOT EXISTS operators will return true and false if there is **AT LEAST** one row returned by the inner query.
- Specifically used in **correlated** (outer query tables are used in the inner query) nested queries.
- Execution of EXISTS/NOT EXISTS operators



EXISTS & NOT EXISTS operators

- Find the name of the students who are enrolled in a course.

SELECT s_name

FROM student S

WHERE **EXISTS** (SELECT s_id FROM enrolment E where S.s_id=E.s_id);

```
utgupta=# SELECT s_name
utgupta-# FROM student S
utgupta-# WHERE EXISTS (SELECT s_id FROM enrolment E where S.s_id=E.s_id);
 s_name
--------
 DAVID
 JULIA
 DAVID
 ABY
(4 rows)
```

EXISTS & NOT EXISTS operators (cont.)

- Find the name of the students who are **NOT** enrolled in a course.

SELECT s_name

FROM student S

WHERE **NOT EXISTS** (SELECT s_id FROM enrolment E where S.s_id=E.s_id);

```
utgupta=# SELECT s_name
utgupta-# FROM student S
utgupta-# WHERE NOT EXISTS (SELECT s_id FROM enrolment E where S.s_id=E.s_id);
 s_name
--------
 JOEL
(1 row)
```

EXISTS & NOT EXISTS operators (cont.)

- Get the names of students who enrolled in DB 1 but not DB 2:

```
SELECT s_name
FROM Student s
WHERE EXISTS ( SELECT *
                 FROM Enrolment e, Courses c
                WHERE s.s_id = e.s_id
                  AND e.c_id = c.c_id
                  AND e.dropped = False
                  AND c.c_name = 'DB 1')
AND NOT EXISTS ( SELECT *
                  FROM Enrolment e, Courses c
                 WHERE s.s_id = e.s_id
                   AND e.c_id = c.c_id
                   AND c.c_name = 'DB 2');
```

```
utgupta=# SELECT s_name
utgupta-#   FROM Student s
utgupta-#  WHERE EXISTS ( SELECT *
utgupta(#                   FROM Enrolment e, Courses c
utgupta(#                  WHERE s.s_id = e.s_id
utgupta(#                    AND e.c_id = c.c_id
utgupta(#                    AND e.dropped = False
utgupta(#                    AND c.c_name = 'DB 1')
utgupta-#    AND NOT EXISTS ( SELECT *
utgupta(#                      FROM Enrolment e, Courses c
utgupta(#                     WHERE s.s_id = e.s_id
utgupta(#                       AND e.c_id = c.c_id
utgupta(#                       AND c.c_name = 'DB 2');
 s_name
--------
 DAVID
(1 row)
```

Differences between IN and EXISTS operators

	IN Operator	EXISTS Operator
1.	IN can be used as a replacement for multiple OR operators.	To determine if any values are returned or not, we use EXISTS.
2.	IN works faster than the EXISTS Operator when If the sub-query result is small.	If the sub-query result is larger, then EXISTS works faster than the IN Operator.
3.	In the IN-condition SQL Engine compares all the values in the IN Clause.	Once true is evaluated in the EXISTS condition then the SQL Engine will stop the process of further matching.
4.	To check against only a single column, IN operator can be used.	For checking against more than one single column, you can use the EXISTS Operator.
5.	The IN operator cannot compare anything with NULL values.	The EXISTS clause can compare everything with NULLs.
6.	A direct set of values can be given for comparison.	Cannot compare directly the values, sub-query needs to be given.

Reference Link

UNION

- Combines the results of two queries:

Example

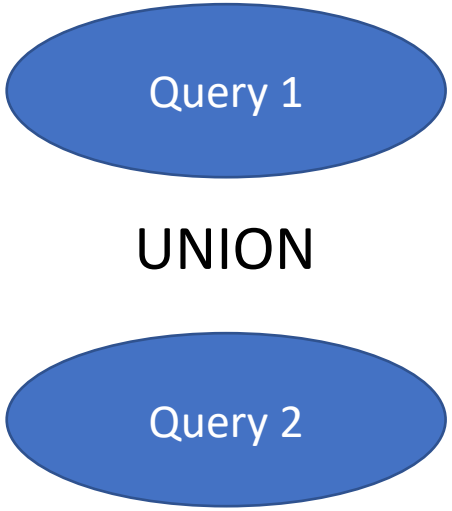
Query 1:

SELECT s_name, dob FROM student where dob > DATE '2000-01-15';

Query 2:

SELECT s_name, dob FROM student S

WHERE EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id);



UNION (cont.)

Example

Query 1:

SELECT s_name, dob FROM student where dob > DATE '2000-01-15';

```
utgupta=# SELECT s_name, dob FROM student where dob > DATE '2000-01-15';
 s_name |    dob
--------+------------
 JULIA  | 2000-02-18
 DAVID  | 2000-02-01
 JOEL   | 2002-03-23
(3 rows)
```

Query 2:

SELECT s_name, dob FROM student S

WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id);

```
utgupta=# SELECT s_name, dob FROM student S
utgupta-# WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id);
 s_name |    dob
--------+------------
 JOEL   | 2002-03-23
(1 row)
```

UNION (cont.)

Example

(SELECT s_name, dob FROM student where dob > DATE '2000-01-15')

UNION

(SELECT s_name, dob FROM student S

WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id));

```
utgupta=# (SELECT s_name, dob FROM student where dob > DATE '2000-01-15')
utgupta-# UNION
utgupta-# (SELECT s_name, dob FROM student S
utgupta(# WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id));
 s_name |    dob
--------+------------
 DAVID  | 2000-02-01
 JOEL   | 2002-03-23
 JULIA  | 2000-02-18
(3 rows)
```

UNION (cont.)

Example (Bags)

(SELECT s_name, dob FROM student where dob > DATE '2000-01-15')

UNION ALL

(SELECT s_name, dob FROM student S

WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id));

```
utgupta=# (SELECT s_name, dob FROM student where dob > DATE '2000-01-15')
UNION ALL
(SELECT s_name, dob FROM student S
WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id));
 s_name |    dob
--------+------------
 JULIA  | 2000-02-18
 DAVID  | 2000-02-01
 JOEL   | 2002-03-23
 JOEL   | 2002-03-23
(4 rows)
```

INTERSECT

- Displays the common results from the two queries.

Example

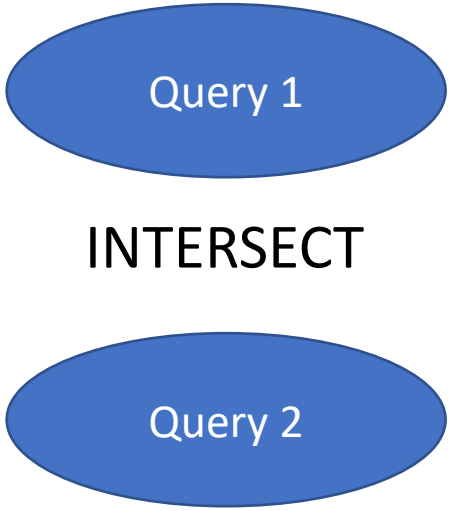
Query 1:

SELECT s_name, dob FROM student where dob > DATE '2000-01-15';

Query 2:

SELECT s_name, dob FROM student S

WHERE EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id);



INTERSECT (cont.)

Example

Query 1:

SELECT s_name, dob FROM student where dob > DATE '2000-01-15';

```
utgupta=# SELECT s_name, dob FROM student where dob > DATE '2000-01-15';
 s_name |    dob
--------+------------
 JULIA  | 2000-02-18
 DAVID  | 2000-02-01
 JOEL   | 2002-03-23
(3 rows)
```

Query 2:

SELECT s_name, dob FROM student S

WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id);

```
utgupta=# SELECT s_name, dob FROM student S
utgupta-# WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id);
 s_name |    dob
--------+------------
 JOEL   | 2002-03-23
(1 row)
```

INTERSECT (cont.)

Example

(SELECT s_name, dob FROM student where dob > DATE '2000-01-15')

INTERSECT

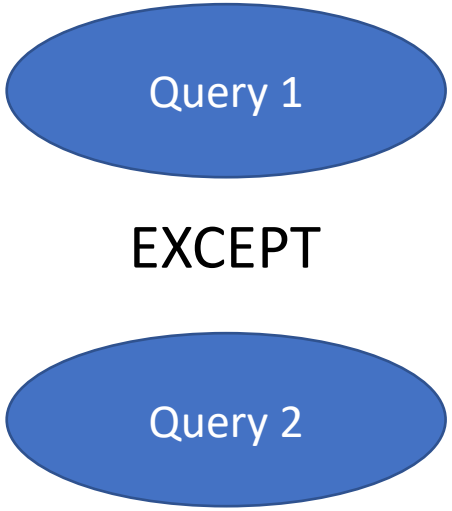
(SELECT s_name, dob FROM student S

WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id));

```
utgupta=# (SELECT s_name, dob FROM student where dob > DATE '2000-01-15')
utgupta-# INTERSECT
utgupta-# (SELECT s_name, dob FROM student S
utgupta(# WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id));
 s_name |    dob
--------+------------
 JOEL   | 2002-03-23
(1 row)
```

EXCEPT

- Displays the rows from the query 1 that are not in the result of query 2.



Example

Query 1:

SELECT s_name, dob FROM student where dob > DATE '2000-01-15';

Query 2:

SELECT s_name, dob FROM student S

WHERE EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id);

EXCEPT (cont.)

Example

Query 1:

SELECT s_name, dob FROM student where dob > DATE '2000-01-15';

```
utgupta=# SELECT s_name, dob FROM student where dob > DATE '2000-01-15';
 s_name |    dob
--------+------------
 JULIA  | 2000-02-18
 DAVID  | 2000-02-01
 JOEL   | 2002-03-23
(3 rows)
```

Query 2:

SELECT s_name, dob FROM student S

WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id);

```
utgupta=# SELECT s_name, dob FROM student S
utgupta-# WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id);
 s_name |    dob
--------+------------
 JOEL   | 2002-03-23
(1 row)
```

EXCEPT (cont.)

Example

(SELECT s_name, dob FROM student where dob > DATE '2000-01-15')

EXCEPT

(SELECT s_name, dob FROM student S

WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id));

```
utgupta=# (SELECT s_name, dob FROM student where dob > DATE '2000-01-15')
utgupta-# EXCEPT
utgupta-# (SELECT s_name, dob FROM student S
utgupta(# WHERE NOT EXISTS (SELECT * FROM enrolment E where S.s_id=E.s_id));
 s_name |    dob
--------+------------
 DAVID  | 2000-02-01
 JULIA  | 2000-02-18
(2 rows)
```

ANY and ALL operators

- The ANY and ALL operators helps you to compare a value picked from a column with a range of other values.

Example: Find the students who are younger than ALL the students enrolled in the CSE 180 course.

SELECT s_name, dob FROM student

WHERE dob **> ALL** (SELECT dob FROM student S, enrolment E

WHERE c_id = 180 AND S.s_id=E.s_id);

```
utgupta=# SELECT s_name, dob FROM student
utgupta-# WHERE dob > ALL (SELECT dob FROM student S, enrolment E
utgupta(#
utgupta(#    WHERE c_id = 180 AND S.s_id=E.s_id);
 s_name |    dob
--------+------------
 JULIA  | 2000-02-18
 JOEL   | 2002-03-23
(2 rows)
```

ANY and ALL operators

- The ANY and ALL operators helps you to compare a value picked from a column with a range of other values.

Example: Find the students who are younger than ANY one of the students enrolled in the CSE 180 course.

SELECT s_name, dob FROM student

WHERE dob **> ANY** (SELECT dob FROM student S, enrolment E

WHERE c_id = 180 AND S.s_id=E.s_id);

```
utgupta=# SELECT s_name, dob FROM student
utgupta-# WHERE dob > ANY (SELECT dob FROM student S, enrolment E
utgupta(#
utgupta(#    WHERE c_id = 180 AND S.s_id=E.s_id);
 s_name |    dob
--------+------------
 JULIA  | 2000-02-18
 DAVID  | 2000-02-01
 JOEL   | 2002-03-23
 ABY    | 2000-01-01
(4 rows)
```

Create commands (Students)

```
CREATE TABLE STUDENT (
        S_ID INT,
        S_NAME VARCHAR(30),
        SSN CHAR(9),
        DOB DATE DEFAULT '2000-01-01' NOT NULL,
        GPA NUMERIC(3,2),
        HAS_GRAD BOOL,
        PHONE CHAR(10),
        EMAIL VARCHAR(50),
        PRIMARY KEY (S_ID),
        UNIQUE (EMAIL),
        UNIQUE (PHONE, S_NAME),
        UNIQUE (SSN)
);
```

```
CREATE TABLE COURSES (
        C_ID INT PRIMARY KEY,
        C_NAME CHAR(9) UNIQUE,
        CREDITS INT DEFAULT 5 NOT NULL
);
```

```
CREATE TABLE ENROLMENT (
        S_ID INT REFERENCES STUDENT,
        C_ID INT REFERENCES COURSES,
        DROPPED BOOL NOT NULL,
        PRIMARY KEY(S_ID, C_ID)
);
```

Insert Commands (Students)

Student

```
INSERT INTO STUDENT VALUES (1, 'DAVID', '1234', '1997-01-01', 3.8, false, '0123456', 'david@pqrs.com');
INSERT INTO STUDENT VALUES (2, 'JULIA', '4567', '2000-02-18', 3.9, false, '1234589', 'julia@pqrs.com');
INSERT INTO STUDENT VALUES (3, 'DAVID', '2468', '2000-02-01', 3.8, false, '9827123', 'david1@pqrs.com');
INSERT INTO STUDENT VALUES (4, 'JOEL', '3412', '2002-03-23', 3.34, false, NULL, 'joel@pqrs.com');
INSERT INTO STUDENT (S_ID, S_NAME, SSN, GPA, HAS_GRAD, PHONE, EMAIL) VALUES (5, 'ABY', '4321', 3.32, true, NULL, NULL);
```

Courses

```
INSERT INTO COURSES VALUES (180, 'DB 1');
INSERT INTO COURSES VALUES (181, 'DB 2');
INSERT INTO COURSES VALUES (215, 'DB GRAD 1');
INSERT INTO COURSES VALUES (160, 'CG');
INSERT INTO COURSES VALUES (280, 'CV SEM', 2);
```

Enrolment

```
INSERT INTO ENROLMENT VALUES (1, 180, false);
INSERT INTO ENROLMENT VALUES (5, 160, false);
INSERT INTO ENROLMENT VALUES (5, 215, false);
INSERT INTO ENROLMENT VALUES (1, 181, true);
INSERT INTO ENROLMENT VALUES (2, 160, false);
INSERT INTO ENROLMENT VALUES (5, 180, false);
INSERT INTO ENROLMENT VALUES (2, 280, false);
INSERT INTO ENROLMENT VALUES (5, 181, false);
INSERT INTO ENROLMENT VALUES (2, 181, true);
INSERT INTO ENROLMENT VALUES (3, 180, false);
INSERT INTO ENROLMENT VALUES (5, 280, false);
```

Create commands (Lecture 4)

```
CREATE TABLE customers (
    cid INT PRIMARY KEY,
    cname VARCHAR(20),
    level CHAR(20),
    type CHAR(20),
    age INT);
```

```
CREATE TABLE slopes (
    slopeid CHAR(3) PRIMARY KEY,
    name VARCHAR(20),
    color VARCHAR(20)
);
```

```
CREATE TABLE activities (
    cid INT,
    slopeid CHAR(3),
    day DATE,
    PRIMARY KEY (cid, slopeid, day),
    FOREIGN KEY (cid) REFERENCES customers,
    FOREIGN KEY (slopeid) REFERENCES slopes
);
```

Insert commands (Lecture 4)

```
INSERT INTO customers VALUES (36, 'Cho', 'Beginner', 'snowboard', 18);
INSERT INTO customers VALUES (34, 'Luke', 'Inter', 'snowboard', 25);
INSERT INTO customers VALUES (87, 'Ice', 'Advanced', 'ski', 20);
INSERT INTO customers VALUES (39, 'Paul', 'Beginner', 'ski', 33);

INSERT INTO slopes VALUES ('s1', 'Mountain Run', 'blue');
INSERT INTO slopes VALUES ('s2', 'Olympic Lady', 'black');
INSERT INTO slopes VALUES ('s3', 'Magic Carpet', 'green');
INSERT INTO slopes VALUES ('s4', 'KT-22', 'black');

INSERT INTO activities VALUES (36, 's3', '01/05/21');
INSERT INTO activities VALUES (36, 's1', '01/06/21');
INSERT INTO activities VALUES (36, 's1', '01/07/21');
INSERT INTO activities VALUES (87, 's2', '01/07/21');
INSERT INTO activities VALUES (87, 's1', '01/07/21');
INSERT INTO activities VALUES (34, 's2', '01/05/21');
```