ANSWERS:

Q1. Write an SQL query: **[2 Marks]**

An 'important' course is defined as the one which consists of more than 2 units and has capacity more than or equal to 100. Find all the 'important' ECONOMICS courses, that is, those that have the term 'ECONOMICS' in it. Your table should contain course code and name of the respective course in this order.

```
SELECT CODE, NAME,UNITS,CAPACITY FROM COURSES WHERE (UNITS>2) AND
(CAPACITY>=100) AND (NAME LIKE '%ECONOMICS%') ;
```

Q2. Write two SQL statements: **[2 Marks]**

Create a new table named 'STUDENTS_CHANGED' having all the fields same as STUDENTS table except for the id field. The new table has the ids suffixed with a '0'. You can make use of CONVERT() function. Its format is described below – `CONVERT( NUMERIC, string_name)` converts string_name to Numeric format. Similarly, `CONVERT( VARCHAR, number)` changes `'number'` to `VARCHAR`. Note that the STUDENTS_CHANGED table should have the same order of columns as in STUDENTS table.

```
SELECT * INTO STUDENTS_CHANGED FROM STUDENTS;
UPDATE STUDENTS_CHANGED
SET ID= CONVERT(NUMERIC, CONVERT(VARCHAR, ID)+'0') ;
```

Q3. Write a single SQL query: **[3 Marks]**

Find the highest CGPA among the students belonging to **each course** and studying under **each lecturer** and born or after the first of January 1998. Sort the result in descending order of CGPA.
The resulting query should have course code, lecturer ID and highest CGPA columns in this order.

```
SELECT CODE, LECTID, MAX(CGPA) AS MAX_CG
FROM ENROLLEDIN T1, STUDENTS T2
WHERE (T2.ID=T1.STUDID AND (DATEDIFF( DAY, 1998-01-01,T2.DOB)>=0))
GROUP BY CODE, LECTID
ORDER BY MAX_CG DESC;
```

Q4. Write a single SQL query: **[5 Marks]**

Make the "merit list" - fetch all the details of the top 10 students with high CGPA. You can assume that all students have distinct CGPAs.

```
SELECT * FROM STUDENTS S1
WHERE (SELECT COUNT( CGPA ) FROM STUDENTS S2 WHERE S2.CGPA>S1.CGPA) <
10
ORDER BY CGPA DESC;
```

Q5. Write a single SQL query: **[4 Marks]**

Find the course name and full name of the lecturer for all “multi-section” courses. A multi-section course is defined as a course lectured by two or more lecturers.

```
SELECT C.NAME, L.FIRSTNAME+' '+COALESCE(L.LASTNAME,'') AS 'FULL NAME'
FROM COURSES C, LECTUREDBY B, LECTURERS L
WHERE L.ID=B.ID AND C.CODE=B.CODE AND C.CODE IN
(SELECT CODE FROM LECTUREDBY GROUP BY CODE HAVING COUNT(ID)>1 );
```

Q6. Write a single SQL query: **[4 Marks]**

Find all the details of students who are enrolled in all the courses lectured by Professor Kamal Kapoor.

```
SELECT * FROM STUDENTS WHERE ID IN
      (SELECT E.STUDID FROM ENROLLEDIN E WHERE NOT EXISTS
            (SELECT CODE FROM LECTUREDBY WHERE ID IN
                  (SELECT ID FROM LECTURERS WHERE FIRSTNAME+'
            '+LASTNAME LIKE 'KAMAL KAPOOR')
            EXCEPT
            SELECT E1.CODE FROM ENROLLEDIN E1 WHERE E1.STUDID=E.STUDID
            )
      );
```

Q7. Create a view: **[2 Marks]**

Create a view "teacher_lecture_prereq" which is a table that contains three tuple rows of the following information

- id of a teacher
- the code of a course he/she teaches
- the code of the prerequisite course needed for the above taught course.

```
CREATE VIEW TEACHER_LECTURE_PREREQ AS
SELECT LECTUREDBY.ID, COURSES.CODE, PREREQ.REQ
FROM COURSES INNER JOIN PREREQ ON COURSES.CODE = PREREQ.CODE
INNER JOIN LECTUREDBY ON COURSES.CODE=LECTUREDBY.CODE;
```

Q8. Design a trigger: **[8 Marks]**

Write a trigger that allows only ***certain*** kinds of updates (details are mentioned below) on the created view "`teacher_lecture_prereq`".

The following are the constraints that exist on the kinds of updates -

A) **Updating Course Code** - allow only if all occurrences of one single course are changed.

B) **Updating Prerequisites** - allow any number of occurrences to be changed.

No other kind of update is allowed. Please note that a single update statement can perform ***either B or A and not both***. Make sure to account for wrong ***inputs that violate the consistency*** of the database. ***Rollback in such situations***.

```
CREATE TRIGGER TR_COURSES_UPDATE
ON TEACHER_LECTURE_PREREQ
INSTEAD OF UPDATE
AS
-- BEGIN TRANSACTION
BEGIN TRANSACTION TRANS;

-- CHECKPOINT IN CASE OF ERROR
SAVE TRANSACTION CHECK_1;

-- VARIABLES (INSERT)
DECLARE @INSERT_TEACHER_ID NUMERIC(9),@INSERT_COURSE_CODE
NUMERIC(9),@INSERT_PREREQ_CODE NUMERIC(9);

-- VARIABLES (DELETE)
DECLARE @DELETE_COURSE_CODE NUMERIC(9),@DELETE_TEACHER_ID
NUMERIC(9),@DELETE_PREREQ_CODE NUMERIC(9);

-- VARIABLES FOR COURSES.CODE AND PREREQ.REQ CHANGES
DECLARE @NAME VARCHAR(30);
DECLARE @UNITS NUMERIC(1);
DECLARE @CAPACITY NUMERIC(3);

-- INSERT_CURSOR
DECLARE INSERT_CURSOR CURSOR FOR SELECT * FROM INSERTED;
OPEN INSERT_CURSOR;
FETCH NEXT FROM INSERT_CURSOR INTO
@INSERT_TEACHER_ID,@INSERT_COURSE_CODE,@INSERT_PREREQ_CODE;

-- CHECK IF THE INSERT TABLE IS EMPTY
IF ( @@FETCH_STATUS != 0 ) BEGIN
     CLOSE INSERT_CURSOR;
     DEALLOCATE INSERT_CURSOR;
     ROLLBACK TRANSACTION CHECK_1;
     print('Inserted table is Empty.')
END

ELSE BEGIN
     -- DELETE CURSOR
     DECLARE DELETE_CURSOR CURSOR FOR SELECT * FROM DELETED;
     OPEN DELETE_CURSOR;
     FETCH NEXT FROM DELETE_CURSOR INTO
     @DELETE_TEACHER_ID,@DELETE_COURSE_CODE,@DELETE_PREREQ_CODE;

     -- CHECK IF THE DELETE CURSOR IS EMPTY
     IF ( @@FETCH_STATUS != 0 ) BEGIN
```

```
      CLOSE INSERT_CURSOR;
      CLOSE DELETE_CURSOR;
      DEALLOCATE INSERT_CURSOR;
      DEALLOCATE DELETE_CURSOR;
      ROLLBACK TRANSACTION CHECK_1;
      print('Deleted table is Empty');
END

ELSE BEGIN

      -- CHECKING WHAT HAS CHANGED
      DECLARE @FLAG_TEACHER NUMERIC(1),@FLAG_COURSE
      NUMERIC(1),@FLAG_PREREQ NUMERIC(1);
      SET @FLAG_TEACHER = 0;
      SET @FLAG_COURSE = 0;
      SET @FLAG_PREREQ = 0;

      -- COUNTING HOW MANY THINGS HAVE CHANGED
      DECLARE @COUNT NUMERIC(3);
      SET @COUNT = 0;

      IF( @INSERT_TEACHER_ID != @DELETE_TEACHER_ID ) BEGIN
            SET @FLAG_TEACHER = 1;
            SET @COUNT = @COUNT + 1;
            print('Teacher ID changed');
      END

      IF( @INSERT_COURSE_CODE != @DELETE_COURSE_CODE ) BEGIN
            SET @FLAG_COURSE = 1;
            SET @COUNT = @COUNT + 1;
            print('Course code changed');
      END

      IF( @INSERT_PREREQ_CODE != @DELETE_PREREQ_CODE ) BEGIN
            SET @FLAG_PREREQ = 1;
            SET @COUNT = @COUNT + 1;
            print('Prereq changed');
      END

      IF ( (@COUNT > 1) OR (@FLAG_TEACHER = 1) ) BEGIN
            -- CLOSE AND DEALLOCATE ALL CURSORS AND ROLLBACK
            CLOSE INSERT_CURSOR;
            CLOSE DELETE_CURSOR;
            DEALLOCATE INSERT_CURSOR;
            DEALLOCATE DELETE_CURSOR;
            ROLLBACK TRANSACTION CHECK_1;
            print('More than 1 variable changed, Rolling Back');
      END
```

```
ELSE IF(
-- CHECK IF NEW CODE IS STILL A PART OF PREVIOUS
COURSES.CODE
(@FLAG_COURSE=1 AND  (@INSERT_COURSE_CODE IN (SELECT CODE
FROM COURSES)))
OR
-- CHECK IF THE PREREQ COURSE CODE IS NOT A PART OF THE
COURSES.CODE
(@FLAG_PREREQ=1 AND  (@INSERT_PREREQ_CODE NOT IN (SELECT
CODE FROM COURSES))))  BEGIN
-- CLOSE AND DEALLOCATE ALL CURSORS
CLOSE INSERT_CURSOR;
CLOSE DELETE_CURSOR;
DEALLOCATE INSERT_CURSOR;
DEALLOCATE DELETE_CURSOR;
ROLLBACK TRANSACTION CHECK_1;
print('Primary Constraint or Foreign Key Constraint
Violated');
END

-- CHECK IF THE COURSE OF ONLY ONE CODE HAS BEEN UPDATED
ELSE IF( @FLAG_COURSE=1 AND (select count(*) from (select
distinct code from deleted) d1) != 1 )
BEGIN
     -- CLOSE AND DEALLOCATE ALL CURSORS
     CLOSE INSERT_CURSOR;
     CLOSE DELETE_CURSOR;
     DEALLOCATE INSERT_CURSOR;
     DEALLOCATE DELETE_CURSOR;
     ROLLBACK TRANSACTION CHECK_1;
     print('More than once course code trying to be
     changed');
END

-- CHECK IF ALL THE OCCURRENCES OF THAT CODE ARE DELETED
ELSE IF( @FLAG_COURSE=1 AND (select count(*) from deleted)
!= (select count(*) from TEACHER_LECTURE_PREREQ where code
= (select distinct code from deleted)) )BEGIN
     -- CLOSE AND DEALLOCATE ALL CURSORS
     CLOSE INSERT_CURSOR;
     CLOSE DELETE_CURSOR;
     DEALLOCATE INSERT_CURSOR;
     DEALLOCATE DELETE_CURSOR;
     ROLLBACK TRANSACTION CHECK_1;
     print('All occurrences of the code not being
     changed');
END
```

```
ELSE BEGIN
      DECLARE @CHECK_FLAG NUMERIC(1);
      SET @CHECK_FLAG = 1;

      -- CASE 1
      IF( @FLAG_COURSE = 1) BEGIN
            -- MODIFY THE CODE FOR A COURSE
            SELECT @NAME = NAME FROM COURSES WHERE
            CODE=@DELETE_COURSE_CODE;
            SELECT @UNITS = UNITS FROM COURSES WHERE
            CODE=@DELETE_COURSE_CODE;
            SELECT @CAPACITY = CAPACITY FROM COURSES WHERE
            CODE=@DELETE_COURSE_CODE;

            -- INSERT NEW TUPLE
            INSERT INTO COURSES
            VALUES(@INSERT_COURSE_CODE,@NAME,@UNITS,@CAPACI
            TY);

            --UPDATE THREE TABLES
            UPDATE LECTUREDBY SET CODE=@INSERT_COURSE_CODE
            WHERE CODE=@DELETE_COURSE_CODE;
            UPDATE PREREQ SET CODE=@INSERT_COURSE_CODE
            WHERE CODE=@DELETE_COURSE_CODE;
            UPDATE PREREQ SET REQ=@INSERT_COURSE_CODE WHERE
            REQ=@DELETE_COURSE_CODE
            UPDATE ENROLLEDIN SET CODE=@INSERT_COURSE_CODE
            WHERE CODE=@DELETE_COURSE_CODE;

            --DELETE OLD TUPLE
            DELETE FROM COURSES WHERE
            CODE=@DELETE_COURSE_CODE;
      END

      -- CASE 2
      ELSE IF ( @FLAG_PREREQ = 1 ) BEGIN

            while @@FETCH_STATUS = 0 BEGIN

                  -- CHECK FOR PRIMARY KEY CONSTRAINT
                  if ( (select count(*) from (select * from
                  PREREQ where code=@INSERT_COURSE_CODE and
                  req=@INSERT_PREREQ_CODE)d12 ) != 0 ) begin
                        print('No row changed')
                        -- Delete that row
```

```
                                    DELETE FROM PREREQ WHERE
                                    CODE=@DELETE_COURSE_CODE AND
                                    REQ=@DELETE_PREREQ_CODE;
                                    -- FETCH NEXT
                                    FETCH NEXT FROM INSERT_CURSOR INTO
                                    @INSERT_TEACHER_ID,@INSERT_COURSE_COD
                                    E,@INSERT_PREREQ_CODE;
                                    FETCH NEXT FROM DELETE_CURSOR INTO
                                    @DELETE_TEACHER_ID,@DELETE_COURSE_COD
                                    E,@DELETE_PREREQ_CODE;
                               end
                               else begin
                                    print('Row changed');
                                    -- MODIFY THE PREREQ OF A COURSE CODE
                                    CHANGE
                                    UPDATE PREREQ SET REQ=
                                    @INSERT_PREREQ_CODE WHERE CODE=
                                    @INSERT_COURSE_CODE and req =
                                    @DELETE_PREREQ_CODE;
                                    -- FETCH NEXT
                                    FETCH NEXT FROM INSERT_CURSOR INTO
                                    @INSERT_TEACHER_ID,@INSERT_COURSE_COD
                                    E,@INSERT_PREREQ_CODE;
                                    FETCH NEXT FROM DELETE_CURSOR INTO
                                    @DELETE_TEACHER_ID,@DELETE_COURSE_COD
                                    E,@DELETE_PREREQ_CODE;
                               end
                          END
                     END
               END

               IF @CHECK_FLAG=1 BEGIN
                    -- CLOSE AND DEALLOCATE BOTH CURSORS
                    CLOSE INSERT_CURSOR;
                    CLOSE DELETE_CURSOR;
                    DEALLOCATE INSERT_CURSOR;
                    DEALLOCATE DELETE_CURSOR;
               END

               -- COMMIT THE TRANSACTIONS
               COMMIT TRANSACTION TRANS;
          END
     END
     GO
```
