Topics

Check, tuple-based

HQ_7_1

Given the relations Students(id, name, major), Courses(id, title, instructor), and Enrolls(sID, cID, term, grade), for all the students who have taken "CS411" and received a grade of 4.0, we want to use INSERT to enroll those students in "CS511" for the "Spring 2018" term. Select the statement that meets the requirement.

✔

```
INSERT INTO Enrolls
SELECT id, "CS511", "Spring 2018", Null
FROM Students
WHERE id IN (
    SELECT sID
    FROM Enrolls
    WHERE cID = "CS411" AND grade = 4.0
)
```

- Inserted tuples must conform to the schema of Enrolls. Here, since we can't predetermine the grade for those future enrollments, we should use Null as a placeholder.

✖

```
INSERT INTO Enrolls
SELECT id, "CS511", "Spring 2018"
FROM Students
WHERE id IN (
    SELECT sID
    FROM Enrolls
    WHERE cID = "CS411" AND grade = 4.0
)
```

- Inserted tuples must conform to the schema of Enrolls. Here, the constructed tuples are missing the value for "grade".

✖

```
INSERT ON Enrolls
```

```
SELECT id, "CS511", "Spring 2018", Null
FROM Students
WHERE id IN (
    SELECT sID
    FROM Enrolls
    WHERE cID = "CS411" AND grade = 4.0
)
```

- INSERT INTO Table is the correct syntax. (INSERT ON Table is used in Triggers.)

HQ_7_2

Given the relations Students(id, name, major), Courses(id, title, instructor), and Enrolls(sID, cID, term, grade), we need to delete, from the Courses table, those courses that were taken by fewer than 10 students in Fall 2016. For now, do not worry about the foreign key constraints.

✔

```
DELETE FROM Courses
WHERE id IN (
    SELECT cID
    FROM Enrolls
    WHERE term = "Fall 2016"
    GROUP BY cID
    HAVING COUNT(sID) < 10
)
```

- In the subquery we find all the courses (by cID) offered in Fall 2016 that were taken by fewer than 10 students. Then we simply delete these tuples whose course IDs are in the results of the subquery from Courses.

✖

```
DELETE FROM Courses
SELECT cID
FROM Enrolls
WHERE term = "Fall 2016"
GROUP BY cID
HAVING COUNT(sID) < 10
```

- The DELETE statement must have a WHERE clause to specify the condition under which a tuple should be deleted.

HQ_7_3

Given the relations Students(id, name, major), Courses(id, title, instructor), and Enrolls(sID, cID, term, grade), for each Mathematics major who has taken CS473 and received a grade of 4.0 (possibly among multiple grades if the student has taken the course more than once), use the UPDATE statement to change the major of those students to CS. Select all the queries that meet the requirement.

✔

```
UPDATE Students
SET major = "CS"
WHERE major = "Mathematics" and 4.0 = ANY (
    SELECT grade
    FROM Enrolls
    WHERE id = sID AND cID = "CS473"
)
```

- A student can possibly take the same course multiple times (in different semesters), and it counts as long as the student received one grade of 4.0. Therefore, the subquery can return multiple grade values, and we use ANY to check if at least one grade equals 4.0.

✔

```
UPDATE Students
SET major = "CS"
WHERE major = "Mathematics" AND  EXISTS (
    SELECT sID
    FROM Enrolls
    WHERE id = sID AND cID = "CS473" AND grade = 4.0
)
```

- The subquery finds the set of students, by ID, who have taken the course and got a grade of 4.0, by joining Enrolls and Students (i.e. Students.id = Enrolls.sID). Note that since the student can take the course more than once, the set can have multiple sIDs. We just need to check if the set is empty or not.

✖

```
UPDATE Students
SET major = "CS"
WHERE major = "Mathematics" AND  EXISTS (
    SELECT sID
```

```
    FROM Enrolls
    WHERE cID = "CS473" AND grade = 4.0
)
```

- In the subquery's WHERE clause, the equi-join between Students.id and Enrolls.sID is mising.

HQ_7_4

Given the relations Students(id, name, major), Courses(id, title, instructor), and Enrolls(sID, cID, term, grade), how can you create a view, named "CS411Students", for all the students (by ID) who have taken CS411, along with the terms and grades? Now using that view, how can you find all the Bioengineering majors in the view (by ID and name) who got a grade of 4.0 in CS411?

✔

```
CREATE VIEW CS411Students AS
SELECT sID, term, grade
FROM Enrolls
WHERE cID = "CS411"


SELECT S.id, S.name
FROM Students S, CS411Students S411
WHERE S.id = S411.sID AND S.major = "Bioengineering" AND S411.grade = 4.0
```

- The view essentially extracts every tuple from Enrolls whose cID attribute is "CS411". This view can then be used for queries as if it were a real table, and we no longer have to specify the course in the join condition, because it is a constant value "CS411" in this view.

✖

```
CREATE VIEW (
SELECT sID, term, grade
FROM Enrolls
WHERE cID = "CS411"
) AS CS411Students


SELECT S.id, S.name
FROM Students S, CS411Students S411
WHERE S.id = S411.sID AND S.major = "Bioengineering" AND S411.grade = 4.0
```

- The correct syntax for view creation is CREATE VIEW ViewName AS

(...Subquery...). The answer wrongly uses the "(...Subquery...) AS Name" syntax in the FROM clause.

✖

```
CREATE VIEW CS411Students AS
SELECT sID, term, grade
FROM Enrolls
WHERE cID = "CS411"


SELECT S.id, S.name
FROM Students S, CS411Students S411
WHERE S.major = "Bioengineering" AND S411.grade = 4.0
```

- The second SQL SELECT is missing equi-join between Students and the view.

HQ_7_5

Given the relations Students(id, name, major), Courses(id, title, instructor), and Enrolls(sID, cID, term, grade), if we declare that Enrolls.cID is a foreign key that references Courses.id, then what types of database manipulations can possibly violate this foreign key constraint? Check all that apply.

✔ INSERT INTO Enrolls

- Foreign key constraints are directional, here the constraint is on Enrolls referencing Courses. You can use the notion of pointers to visualize under what situations you may create invalid pointers that reference undefined targets. For instance, a newly inserted Enrolls.cID might not have a corresponding record in Courses, and deleting a Course.id record can create a dangling pointer in Enrolls. The same goes for updating either tables. However, inserting new Course tuples or deleting Enrolls tuples will not violate this constraint, which follows from how the pointers are directed in our constraint declaration.

✔ DELETE FROM Courses

✔ UPDATE Enrolls

✔ UPDATE Courses

✖ INSERT INTO Courses

✖ DELETE FROM Enrolls

HQ_7_6

Given the relations Students(id, name, major), Courses(id, title, instructor), and Enrolls(sID, cID, term, grade), how can we use CHECK() to enforce the foreign key constraint on Enrolls.sID referencing Students.id? Does this provide the same level of referential integrity guarantee as using FOREIGN KEY declaration?

✔ Add `CHECK(sID IN (SELECT id FROM Students))` to the declaration of the sID attribute when creating the Enrolls table. No, FOREIGN KEY declaration provides more thorough referential integrity guarantee than CHECK() does.

- CHECK() is activated only when the Enrolls table itself is modified (via INSERT or UPDATE). However, when the Students table is modified, which can potentially cause a foreign key constraint violation, CHECK() is not aware of the changes taking place outside the Enrolls table, and hence will not be run.

✖ Add `CHECK(sID IN (SELECT id FROM Students))` to the declaration of the sID attribute when creating the Enrolls table. No, CHECK() provides more thorough referential integrity guarantee than FOREIGN KEY declaration does.

✖ Add `CHECK(sID IN (SELECT id FROM Students))` to the declaration of the sID attribute when creating the Enrolls table. Yes, the two approaches provide the same level of referential integrity guarantee.

HQ_7_7

Given the relations Students(id, name, major), Courses(id, title, instructor), and Enrolls(sID, cID, term, grade), when we are creating the Courses table, how can you use CHECK() to enforce the UNIQUE (or PRIMARY KEY) constraint on the Courses.id attribute? Answer the question with the following assumptions: (1) the SQL system fully supports subqueries in a CHECK condition statement; (2) the SQL system has been designed to allow the table that is being created (i.e. Courses) to be used in the CHECK condition for that table's own attributes; (3) a CHECK is performed *after* an INSERT or UPDATE modification involving the id attribute is applied. Select all that apply.

✔ Add `CHECK((SELECT COUNT(*) FROM Courses) = (SELECT COUNT(DISTINCT id) FROM Courses))` to the declaration of the id attribute when creating the Courses table.

- Checking whether the number of distinct values of the key is equal to the total number of tuples after the modification is applied, is essentially checking if each key attribute's value remains unique in the table, and that indeed is the UNIQUE constraint.

✖ Add `CHECK(id NOT IN (SELECT id FROM Courses))` to the declaration of the id attribute when creating the Courses table.

- Since the check is run after the modification is applied, the check condition will always evaluate to false, and thus any INSERT or UPDATE involving the id attribute will always be rejected! This works only if the check happens BEFORE the table is modified,

✖ With the two assumptions above, there is no reliable way to enforce the UNIQUE constraint using CHECK().

HQ_7_8

Given the relations Students(id, name, major), Courses(id, title, instructor), and Enrolls(sID, cID, term, grade), how can you use ASSERTION to enforce the foreign key constraint on Enrolls.sID referencing Students.id?

✔

```
CREATE ASSERTION FKConstraint
CHECK (NOT EXISTS
    SELECT *
    FROM Enrolls
    WHERE sID NOT IN (
        SELECT id
        FROM Students
    )
)
```

- The foreign key constraint we want to impose is on Enrolls.sID referencing Students.id; that is, for every sID in Enrolls, there must be a corresponding id in Students. However, using SQL we must check that the violation (sID NOT IN…) does not happen (NOT EXISTS), since the SQL syntax does not support the expression of "for every sID in Enrolls, it exists in Students). Alternatively, we can use set difference to assert that the set of sIDs must be a subset of the IDs in Students, using the fact that R - S is the empty set iff R is a subset of S.

✖

```
CREATE ASSERTION FKConstraint
CHECK (NOT EXISTS
    SELECT *
    FROM Students
    WHERE id NOT IN (
        SELECT sID
        FROM Enrolls
    )
)
```

- This asserts the foreign key constraint in the opposite direction to what we declared, hence it does not meet the requirement.

HQ_7_9

Given the relations Students(id, name, major), Courses(id, title, instructor), and Enrolls(sID, cID, term, grade), how can you define a trigger to implement the foreign key constraint on Enrolls.sID referencing Students.id, by cascading DELETE events that violate the constraint? Select all triggers that meet the requirement.

✔

```
CREATE TRIGGER FKConstraint
AFTER DELETE ON Students
REFERENCING OLD ROW AS DeletedStudent
FOR EACH ROW
DELETE FROM Enrolls WHERE sID = DeletedStudent.id
```

- Delete events that can violate the constraint happens only in the Students table, not in the Enrolls table. Since we allow such deletions, we choose the event timing to be AFTER, and by referencing each deleted row in Students, we can remove the corresponding rows in Enrolls whose sID matches Students.id.

✔

```
CREATE TRIGGER FKConstraint
AFTER DELETE ON Students
REFERENCING OLD TABLE AS AllDeletedStudents
DELETE FROM Enrolls WHERE sID IN (SELECT id FROM AllDeletedStudents)
```

- We can use statement-level trigger by referencing the entire set of deleted rows (as OLD TABLE), and delete every corresponding row in Enrolls that matches a Student by ID in the old table of deleted Students.

✖

```
CREATE TRIGGER FKConstraint
AFTER DELETE ON Enrolls
REFERENCING OLD ROW AS DeletedEnroll
FOR EACH ROW
DELETE FROM Students WHERE id = DeletedEnroll.sID
```

- This places FK constraint on Students.id referencing Enrolls.sID (enforcing that every student must take at least one course), which is not the requirement.

HQ_7_10

Given the relations Students(id, name, major), Courses(id, title, instructor), and Enrolls(sID, cID, term, grade), we have created a view CS411F16Students(sID, grade), which shows all the students (by sID) who took CS411 in Fall 2016, along with their grades. Now we intend to update a particular student's grade using this view only, e.g., via `UPDATE CS411F16Students SET grade = 4.0 WHERE sID = "ab12"`; however, as we have learned, a view is not materialized and thus its "modification" must be properly translated into the modification of the underlying base table(s). How can you define a trigger to make this translation work?

✔

```
CREATE TRIGGER CS411F16ChangeGrade
INSTEAD OF UPDATE OF grade ON CS411F16Students
REFERENCING
    OLD ROW AS Old
    NEW ROW AS New
FOR EACH ROW
BEGIN
    UPDATE Enrolls
    SET grade = New.grade
    WHERE sID = Old.sID AND cID = "CS411" AND term = "Fall 2016"
END
```

- A trigger for a view should be activated using the INSTEAD OF event timing keyword, and since the underlying base table of CS411F16Students is Enrolls, we need to translate updates of grade on the view into updates on Enrolls. Note that the view implicitly has the cID attribute held constant as "CS411", and the term attribute held constant as "Fall 2016". Therefore, in the action part of the trigger, we need to make sure that we select only the Enrolls tuples about CS411 and Fall 2016 only.

✖

```
CREATE TRIGGER CS411F16ChangeGrade
INSTEAD OF UPDATE OF grade ON CS411F16Students
REFERENCING
    OLD ROW AS Old
    NEW ROW AS New
FOR EACH ROW
BEGIN
    UPDATE Enrolls
    SET grade = New.grade
    WHERE sID = Old.sID AND cID = Old.cID AND term = Old.term
END
```

- The updates on the view only specify sID and grade, without having to specify the course (cID) and term (and that's one of the benefits of using the view instead of Enrolls). Therefore, when referring to an OLD ROW, we do not have access to its cID or term attribute.

✖

```
CREATE TRIGGER CS411F16ChangeGrade
BEFORE UPDATE OF grade ON CS411F16Students
REFERENCING
    OLD ROW AS Old
    NEW ROW AS New
FOR EACH ROW
BEGIN
    UPDATE Enrolls
    SET grade = New.grade
    WHERE sID = Old.sID AND cID = Old.cID AND term = Old.term
END
```

- In addition to mistakenly accessing Old.cID and Old.term, the event timing keyword "BEFORE" is not suitable for declaring a trigger for views, because BEFORE / AFTER will still allow the modification attempt on the view to occur, which would be illegal.
