Week 11 Manipulating Databases (1)

Assignment Solutions

Question 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.

```
*A:

 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 )

B:

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

C:

 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 )
```

Solution: A

Explanation: This question can be solved using the process of elimination. When we look at option b, we notice that it is missing the grade field for inserting into Enrolls, which is wrong. For option c, we notice that it uses the statement ON rather than INTO, which is also wrong. We are left with option a, which is the correct option.

Question 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.

`*A:`

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

`B:`

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

Solution: A

Explanation: For A, 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

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

Question 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.

`*A:`

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

`*B:`

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

`C:`

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

Solution: AB

Explanation: For A, 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,

For B, 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

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

Question 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?

```
*A:

 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

B:

 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

C:

 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
```

Solution: A

Explanation: For A, 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

For B, 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

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

Question 5

In one of the video lectures, Prof. Chang talked about a feature of databases called indexes. Indexes are used to speed up queries. In essence, indexes facilitate the retrieval of a subset of a table's tuples and attributes *without* the need to traverse every single record in the table. However, in the "Food for Thought" question, he suggests that creating an index for every single attribute in the table is a bad idea. Which of the following statements is/are true in the context of indexes? You don't need to know balanced trees (B-trees) to answer this question.

*A: Every time a row is updated, all the indexes on the column(s) affected need to be modified as well. As we have a higher number of indexes, the load at the server rises to keep all the schema objects up to date and this tends to slow things down.

B: Columns that are rarely used to retrieve data should be indexed more and columns with higher activity should be left out of indexing.

C: Using indexes speeds up data retrieval in smaller datasets rather than larger datasets.

*D: An index on an attribute that is not frequently used for querying can waste extra storage space to store the index.

Solution: AD

Explanation: For A, An index is a persistent data structure stored on disk and needs to be kept in sync with the database. Therefore, every modification on the main table should be written through into the index-- and thus it puts extra load on the server.

For D, An index needs to be stored permanently for future querying and therefore it imposes storage overhead.