Q)Add a LIMIT 10 ROWS clause at the end of the previous query, and see what algorithm method is used. (LIMIT n ensures only n rows are output.) Explain what happened, if the join algorithm changes; if the plan does not change, create a different query where the join algorithm changes as a result of adding the LIMIT clause.

Solution:

```
    SELECT *
    FROM student
    JOIN takes ON student.ID = takes.ID
    JOIN course ON takes.course_id = course.course_id
    ORDER BY student.ID, takes.semester, takes.year limit 10;
```

Initially was merge-join then went to nested-loop join. Merge join is generally more efficient when joining large tables, as it requires sorting the input tables and then merging the sorted results. However, if one or both of the tables being joined are small, a nested loop join may be more efficient.
The planner may have decided that the overhead of sorting and merging the larger result set using a merge-join was not worth the potential speedup.

```
explain SELECT *
    FROM student
    JOIN takes ON student.ID = takes.ID
    JOIN course ON takes.course_id = course.course_id
    ORDER BY student.ID, takes.semester, takes.year limit 10;

                                                        QUERY PLAN
------------------------------------------------------------------------------
---------------------------------
 Limit  (cost=2.87..4.46 rows=10 width=83)
   ->  Incremental Sort  (cost=2.87..4755.26 rows=30000 width=83)
         Sort Key: student.id, takes.semester, takes.year
         Presorted Key: student.id
         ->  Nested Loop  (cost=0.72..3330.33 rows=30000 width=83)
               ->  Merge Join  (cost=0.56..2568.55 rows=30000 width=48)
                     Merge Cond: ((student.id)::text = (takes.id)::text)
                     ->  Index Scan using student_pkey on student
(cost=0.28..130.27 rows=2000 width=24)
                     ->  Index Scan using takes_pkey on takes
(cost=0.29..2058.28 rows=30000 width=24)
               ->  Memoize  (cost=0.15..0.17 rows=1 width=35)
                     Cache Key: takes.course_id
                     Cache Mode: logical
                     ->  Index Scan using course_pkey on course
(cost=0.14..0.16 rows=1 width=35)
                           Index Cond: ((course_id)::text =
(takes.course_id)::text)
(14 rows)
```

```
explain analyze SELECT *
    FROM student
    JOIN takes ON student.ID = takes.ID
    JOIN course ON takes.course_id = course.course_id
    ORDER BY student.ID, takes.semester, takes.year limit 10;
```

QUERY PLAN

```
------------------------------------------------------------------------
------------------------------------------------------------------------
--
 Limit  (cost=2.87..4.46 rows=10 width=83) (actual time=0.263..0.269
rows=10 loops=1)
   ->  Incremental Sort  (cost=2.87..4755.26 rows=30000 width=83) (actual
time=0.261..0.265 rows=10 loops=1)
         Sort Key: student.id, takes.semester, takes.year
         Presorted Key: student.id
         Full-sort Groups: 1  Sort Method: quicksort  Average Memory:
26kB  Peak Memory: 26kB
         ->  Nested Loop  (cost=0.72..3330.33 rows=30000 width=83)
(actual time=0.069..0.195 rows=14 loops=1)
               ->  Merge Join  (cost=0.56..2568.55 rows=30000 width=48)
(actual time=0.045..0.094 rows=14 loops=1)
                     Merge Cond: ((student.id)::text = (takes.id)::text)
                     ->  Index Scan using student_pkey on student
(cost=0.28..130.27 rows=2000 width=24) (actual time=0.019..0.021 rows=2
loops=1)
                     ->  Index Scan using takes_pkey on takes
(cost=0.29..2058.28 rows=30000 width=24) (actual time=0.010..0.046
rows=14 loops=1)
               ->  Memoize  (cost=0.15..0.17 rows=1 width=35) (actual
time=0.006..0.006 rows=1 loops=14)
                     Cache Key: takes.course_id
                     Cache Mode: logical
                     Hits: 0  Misses: 14  Evictions: 0  Overflows: 0
Memory Usage: 2kB
                     ->  Index Scan using course_pkey on course
(cost=0.14..0.16 rows=1 width=35) (actual time=0.004..0.004 rows=1
loops=14)
                           Index Cond: ((course_id)::text =
(takes.course_id)::text)
 Planning Time: 1.407 ms
 Execution Time: 0.391 ms
(18 rows)
```