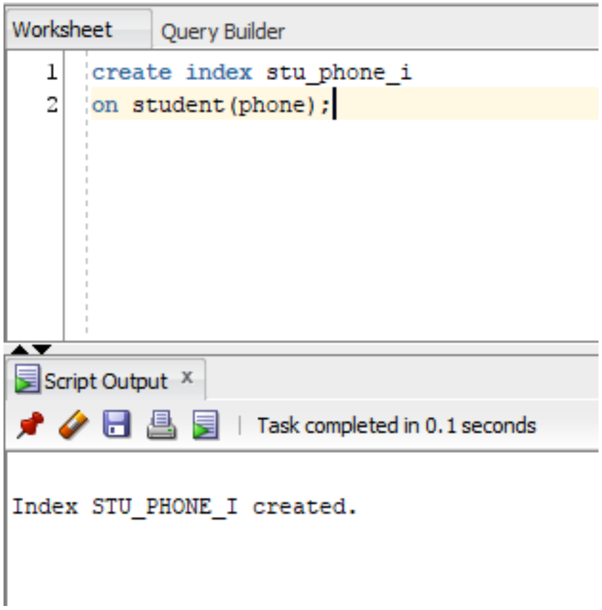
Ryan Paw
Week 4, Report
ANA 660

I, Ryan Paw, finished and completed this week's reading requirements and lab activities. I created and ran the code in SQL Plus myself.

Lab 13.1 Exercises (all the questions)
Lab 18.1 Exercises (questions b, c, d, e, f, g, h, I, and j)

13.1, a)
Code/Output:

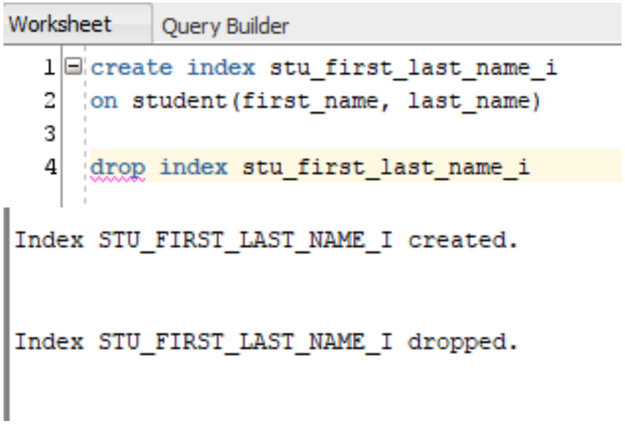


```
create index stu_phone_i
on student(phone);
```

Task completed in 0.1 seconds

```
Index STU_PHONE_I created.
```

13.1, b)
Code/Output:

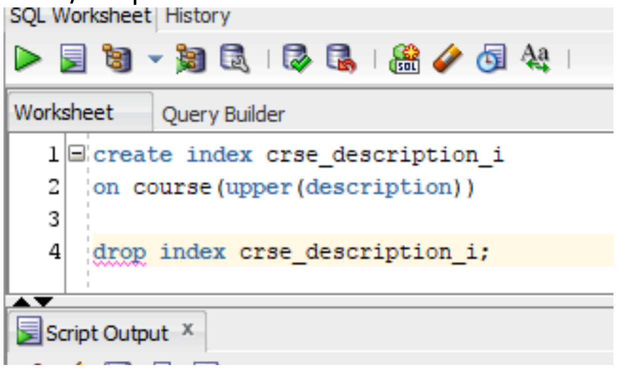


```
create index stu_first_last_name_i
on student(first_name, last_name)

drop index stu_first_last_name_i
```

```
Index STU_FIRST_LAST_NAME_I created.


Index STU_FIRST_LAST_NAME_I dropped.
```

13.1, c)

Code/Output:



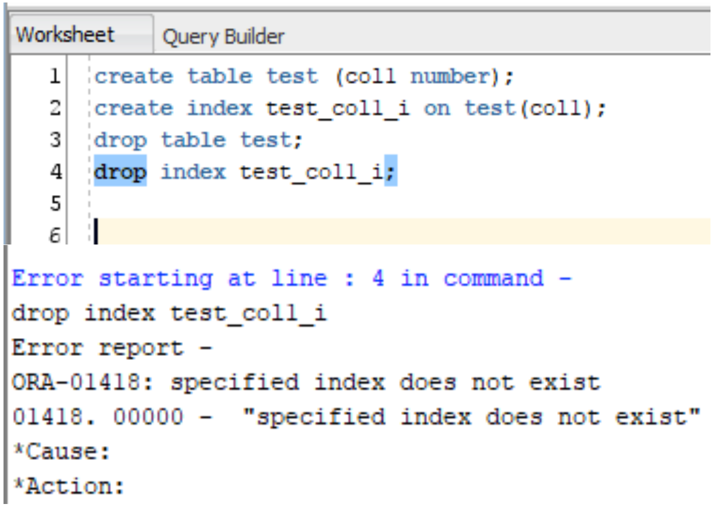
```
create index crse_description_i
on course(upper(description))

drop index crse_description_i;
```

```
Index CRSE_DESCRIPTION_I created.


Index CRSE_DESCRIPTION_I dropped.
```

13.1, d)

The code is receiving an error because when DROP function is used for dropping tables, it automatically removes the index too. Therefore, we are seeing an error that the specified index does not exist because the index was already dropped.

Code/Output:



```
1 create table test (coll number);
2 create index test_coll_i on test(coll);
3 drop table test;
4 drop index test_coll_i;
5
6
```

```
Error starting at line : 4 in command -
drop index test_coll_i
Error report -
ORA-01418: specified index does not exist
01418. 00000 -  "specified index does not exist"
*Cause:
*Action:
```

13.1, e)

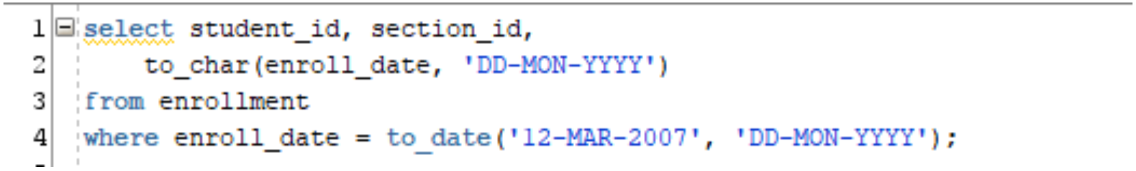
Yes, a B-tree index is a type of indexing that stores data where each node creates a tree structure. It’s a more organized form of indexing in a form of a “search tree”. This makes indexing faster, and could be helpful for columns that are frequently accessed with few distinct values.

13.1, f)

Indexing is advantageous because it improves SQL's performance. Columns that are used frequently in queries can create faster outputs. However, if only a small percentage of rows are accessed, then a typical full table scan would likely be better. Too many indexes could also be a problem because every time there a insert/update/delete code is ran that relates to your indexes, SQL also updates the indexes too. Therefore, it could these functions could decrease performance and require additional disk space on your computer.

13.1, g)

The example query can edited to include the index by the following code:

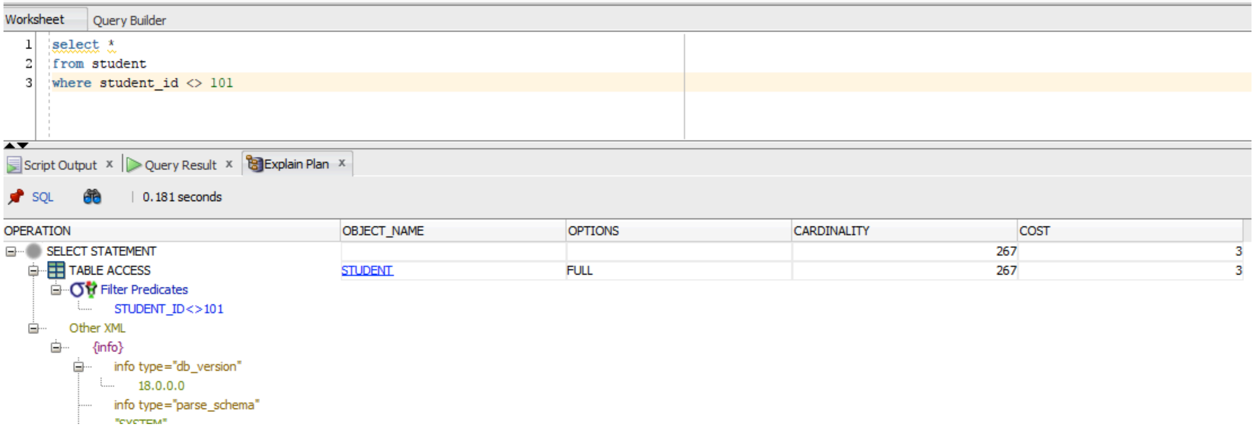


```
select student_id, section_id,
       to_char(enroll_date, 'DD-MON-YYYY')
from enrollment
where enroll_date = to_date('12-MAR-2007', 'DD-MON-YYYY');
```

18.1, b)

After running the example code, the index was not used as shown in the "Explain Plan" function in SQL. A full table scan was used instead. Since this query has a negation (<>) in it, an index is not typically used.

Code/Output:



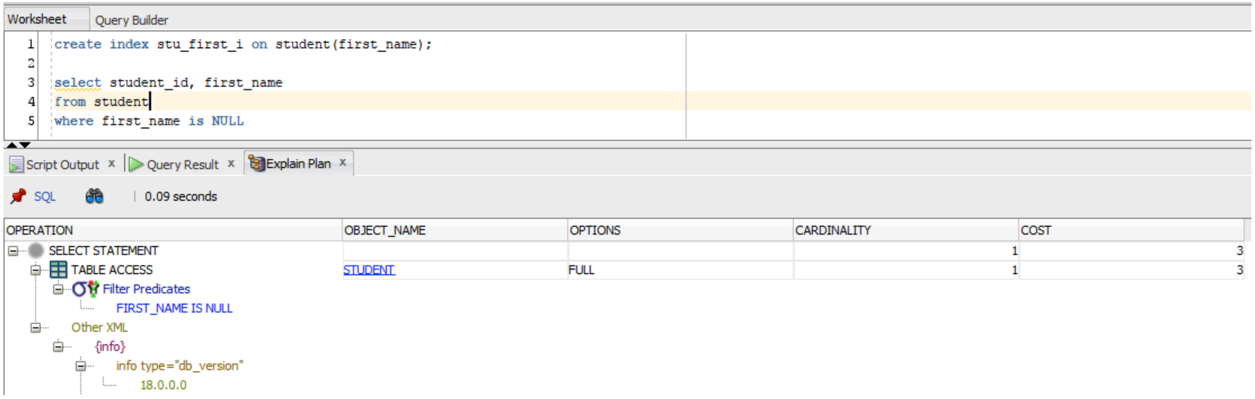
```
select *
from student
where student_id <> 101
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			267	3
TABLE ACCESS	STUDENT	FULL	267	3
Filter Predicates				
STUDENT_ID<>101				
Other XML				
{info}				
info type="db_version"				
18.0.0.0				
info type="parse_schema"				
"SYSTEM"				

18.1, c)

An index was not utilized as shown in the "Explain Plan" result in SQL. This is because NULL values are not stored in the index.

Code/Output:

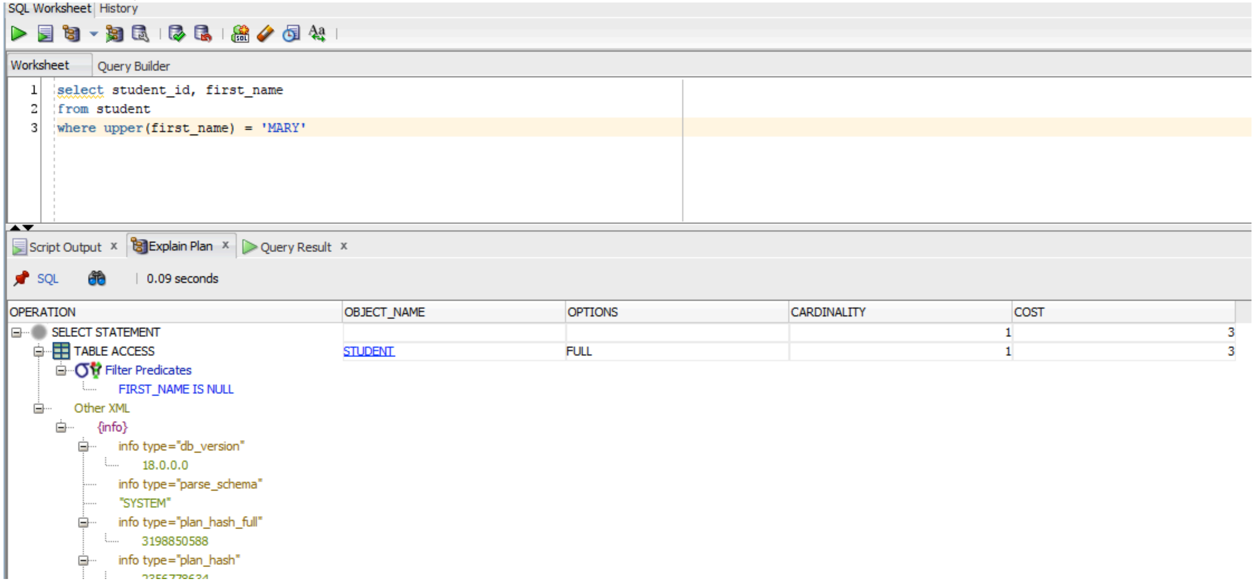


```
create index stu_first_i on student(first_name);

select student_id, first_name
from student
where first_name is NULL
```

```
OPERATION                     OBJECT_NAME     OPTIONS     CARDINALITY     COST
SELECT STATEMENT                                                    1        3
  TABLE ACCESS                STUDENT         FULL                  1        3
    Filter Predicates
        FIRST_NAME IS NULL
  Other XML
    {info}
      info type="db_version"
        18.0.0.0
```

18.1, d)
The query runs, but it does do an index scan, even though there is an index for first_name. It does a full table scan instead. This is because we're modifying the index (UPPER), which makes the index disabled.

Code/Output:



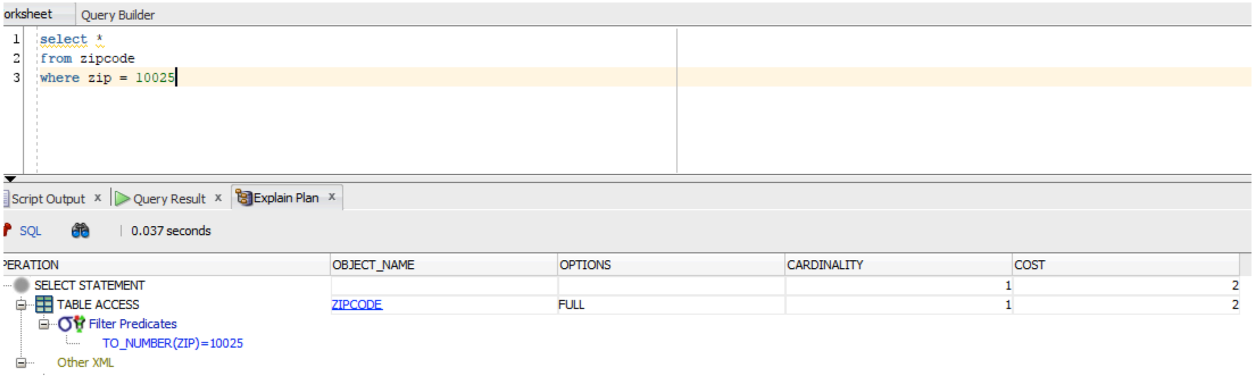
```
select student_id, first_name
from student
where upper(first_name) = 'MARY'
```

```
OPERATION                     OBJECT_NAME     OPTIONS     CARDINALITY     COST
SELECT STATEMENT                                                    1        3
  TABLE ACCESS                STUDENT         FULL                  1        3
    Filter Predicates
        FIRST_NAME IS NULL
  Other XML
    {info}
      info type="db_version"
        18.0.0.0
      info type="parse_schema"
      "SYSTEM"
      info type="plan_hash_full"
        3198850588
      info type="plan_hash"
```

18.1, e)
Between the 2 example queries, the first query did a full table scan, while the second query did an index scan. The first query likely did not use an index scan because the first_name index could not be determined from the index entries. The second query was able to use the stu_first_i index for the index scan.

18.1, f)
The example query does not use the index for the ZIP column. This is because the data types for the ZIP columns do not align. The ZIP column is VARCHAR2 type, the literal is a NUMBER type.

Code/Output:

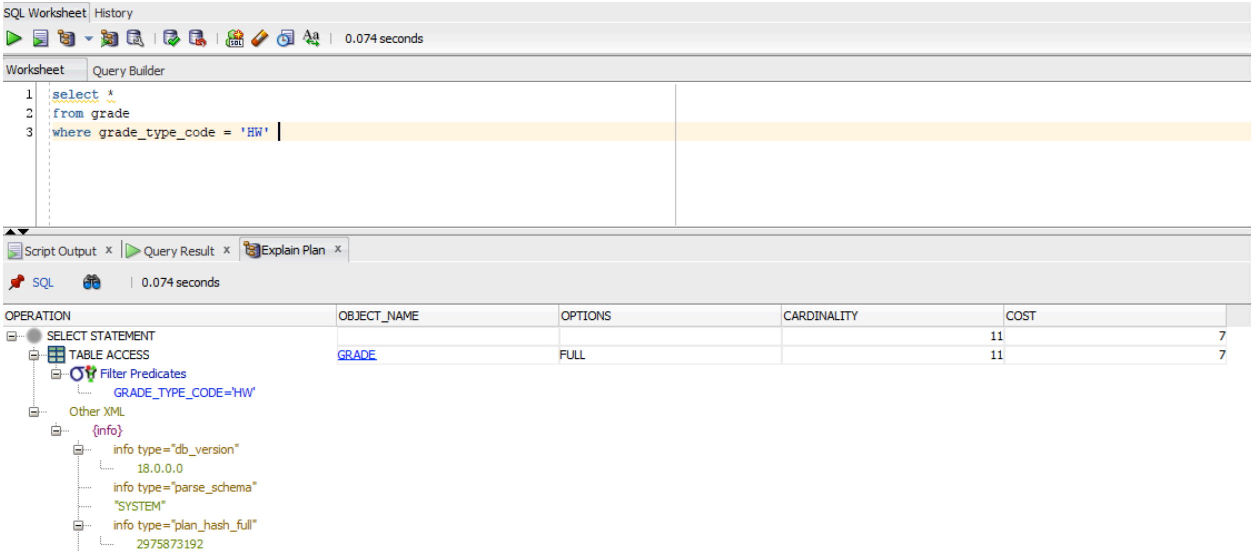


```
select *
from zipcode
where zip = 10025
```

```
PERATION                      OBJECT_NAME   OPTIONS   CARDINALITY   COST
SELECT STATEMENT                                                1      2
  TABLE ACCESS                ZIPCODE       FULL                1      2
    Filter Predicates
      TO_NUMBER(ZIP)=10025
  Other XML
```

18.1, g)
SQL did a full table scan on the grade_type_code column. This is because grade_type_code column is not a leading column. SQL has a skip scan features that improves index scans by skipping index scans if the leading portion of the index is not specified

Code/Output:

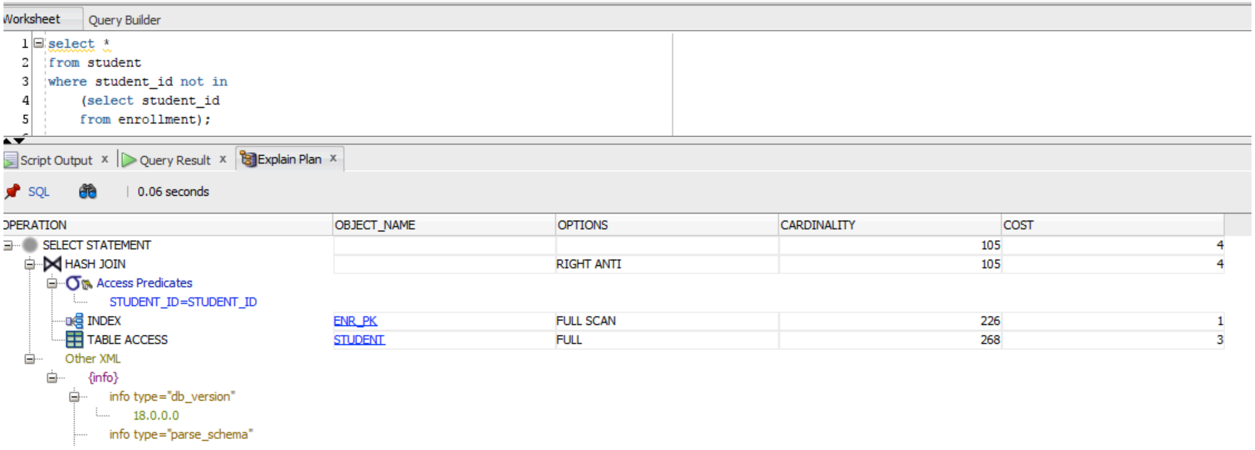


```
select *
from grade
where grade_type_code = 'HW'
```

```
OPERATION                     OBJECT_NAME   OPTIONS   CARDINALITY   COST
SELECT STATEMENT                                               11      7
  TABLE ACCESS                GRADE         FULL               11      7
    Filter Predicates
      GRADE_TYPE_CODE='HW'
  Other XML
    {info}
      info type="db_version"
        18.0.0.0
      info type="parse_schema"
      "SYSTEM"
      info type="plan_hash_full"
        2975873192
```

18.1, h)
In the example code, the driving table is the section table in line 3. The driving table is the table that is joined to other in tables in a SQL query and the first table accessed when a query will run. The type of join is a nested-loop join. A nested-loop join gets a row from the outer table and searches for the row in the inner table.

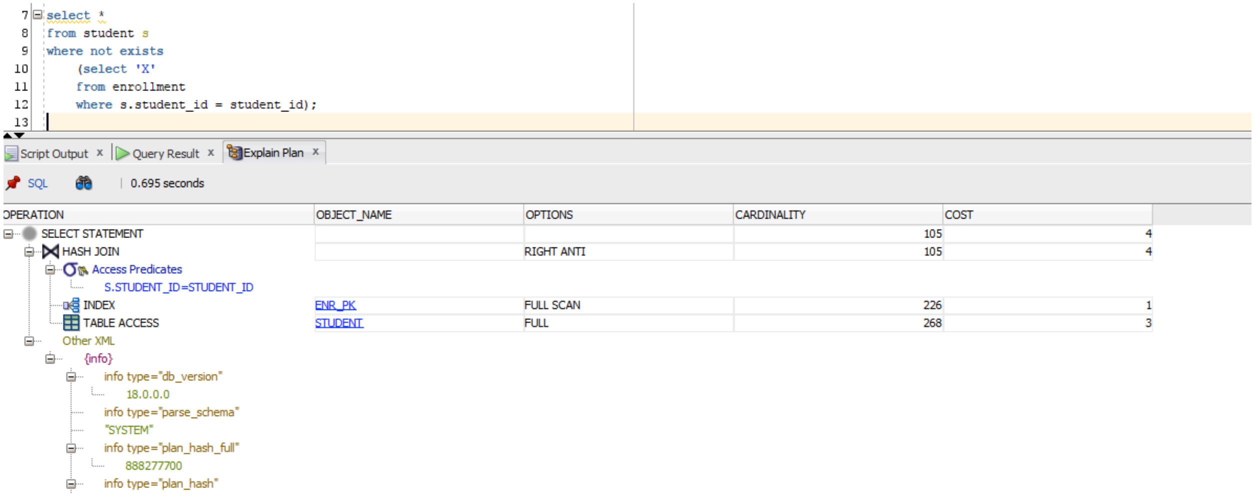
18.1, i)
The first query does a full table scan with a total cost of 4, and does not do an index scan.
Code/Output:



```
select *
from student
where student_id not in
    (select student_id
    from enrollment);
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			105	4
HASH JOIN		RIGHT ANTI	105	4
Access Predicates				
STUDENT_ID=STUDENT_ID				
INDEX	ENR_PK	FULL SCAN	226	1
TABLE ACCESS	STUDENT	FULL	268	3
Other XML				
{info}				
info type="db_version"				
18.0.0.0				
info type="parse_schema"				

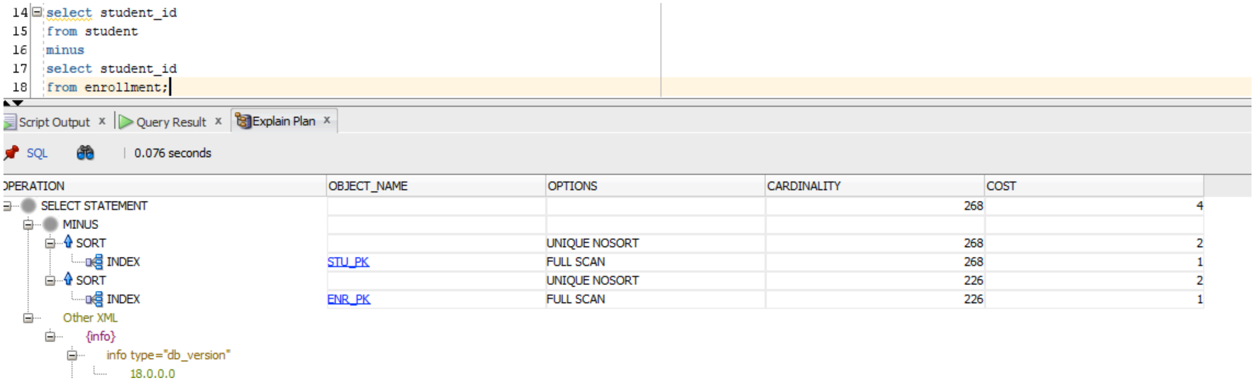
The second query also does a full table scan with the same total cost of 4. No index scan.



```
select *
from student s
where not exists
    (select 'X'
    from enrollment
    where s.student_id = student_id);
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			105	4
HASH JOIN		RIGHT ANTI	105	4
Access Predicates				
S.STUDENT_ID=STUDENT_ID				
INDEX	ENR_PK	FULL SCAN	226	1
TABLE ACCESS	STUDENT	FULL	268	3
Other XML				
{info}				
info type="db_version"				
18.0.0.0				
info type="parse_schema"				
"SYSTEM"				
info type="plan_hash_full"				
888277700				
info type="plan_hash"				

The third query does a full table scan with a total cost of 4. Unlike the other 2 queries, it only has a cost of 1 for each primary key, and then a cost of 2 for a unique nosort.



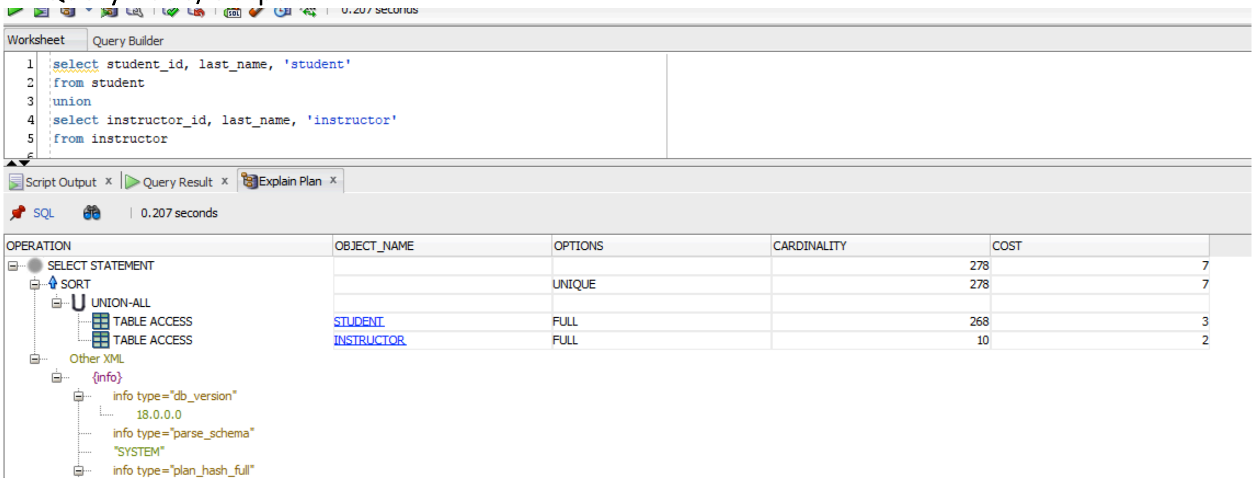
```
select student_id
from student
minus
select student_id
from enrollment;
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			268	4
MINUS				
SORT		UNIQUE NOSORT	268	2
INDEX	STU_PK	FULL SCAN	268	1
SORT		UNIQUE NOSORT	226	2
INDEX	ENR_PK	FULL SCAN	226	1
Other XML				
{info}				
info type="db_version"				
18.0.0.0				

18.1, j)

The cost for the 1st query (cost: 7) is greater than the 2nd query (cost: 5). The UNION statement adds a UNIQUE sort that adds the cost +2, which the UNION ALL statement does not have.

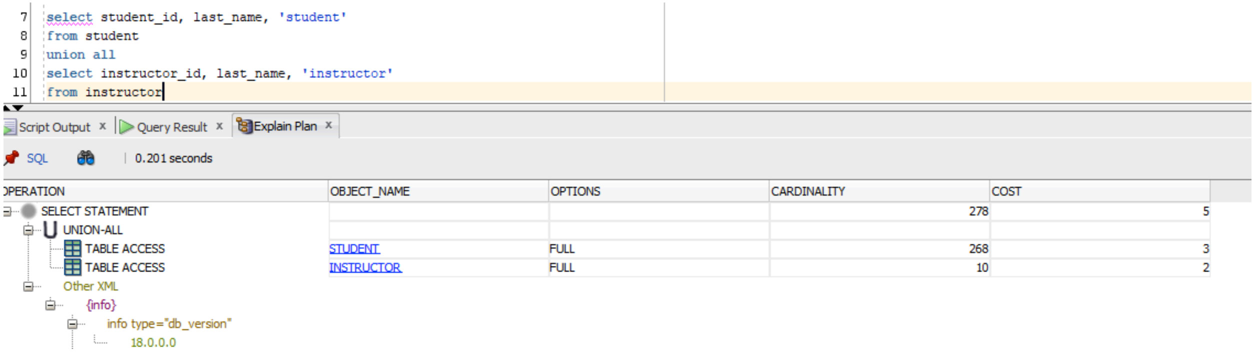
1st Query Code/Output:



```
select student_id, last_name, 'student'
from student
union
select instructor_id, last_name, 'instructor'
from instructor
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			278	7
SORT		UNIQUE	278	7
UNION-ALL				
TABLE ACCESS	STUDENT	FULL	268	3
TABLE ACCESS	INSTRUCTOR	FULL	10	2
Other XML				
{info}				
info type="db_version"				
18.0.0.0				
info type="parse_schema"				
"SYSTEM"				
info type="plan_hash_full"				

2nd Query Code/Output:



```
select student_id, last_name, 'student'
from student
union all
select instructor_id, last_name, 'instructor'
from instructor
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			278	5
UNION-ALL				
TABLE ACCESS	STUDENT	FULL	268	3
TABLE ACCESS	INSTRUCTOR	FULL	10	2
Other XML				
{info}				
info type="db_version"				
18.0.0.0				