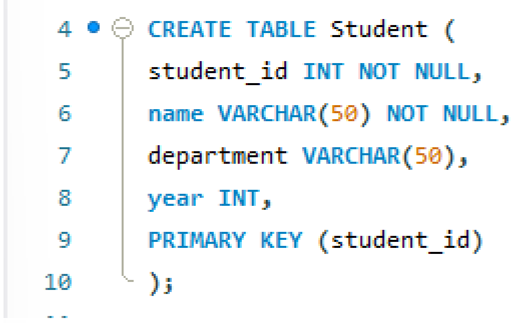
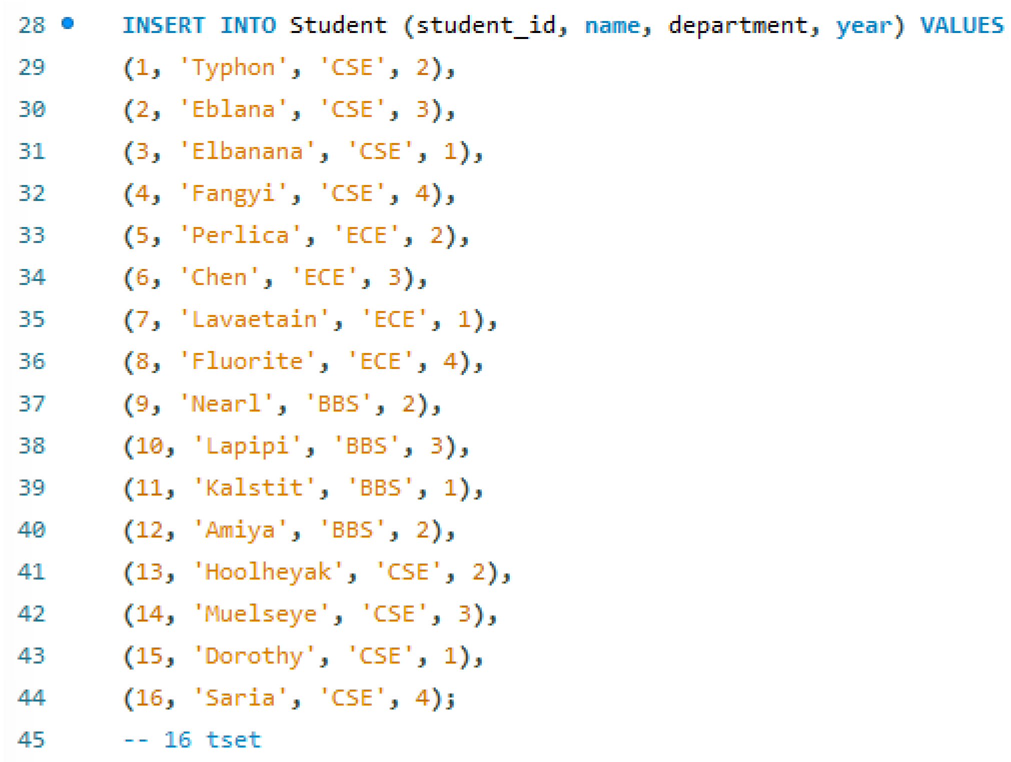
LAB 2: 2331200153

1. Create a table named STUDENT, COURSE, ENROLLMENT

Student

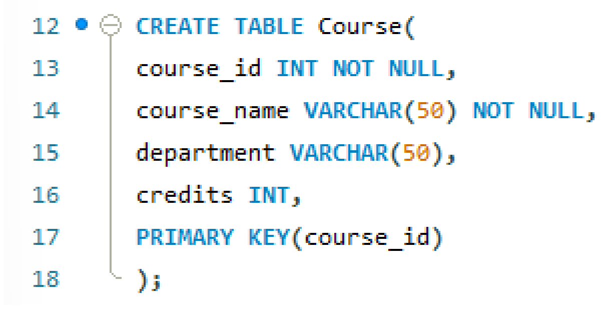


```sql
CREATE TABLE Student (
student_id INT NOT NULL,
name VARCHAR(50) NOT NULL,
department VARCHAR(50),
year INT,
PRIMARY KEY (student_id)
);
```

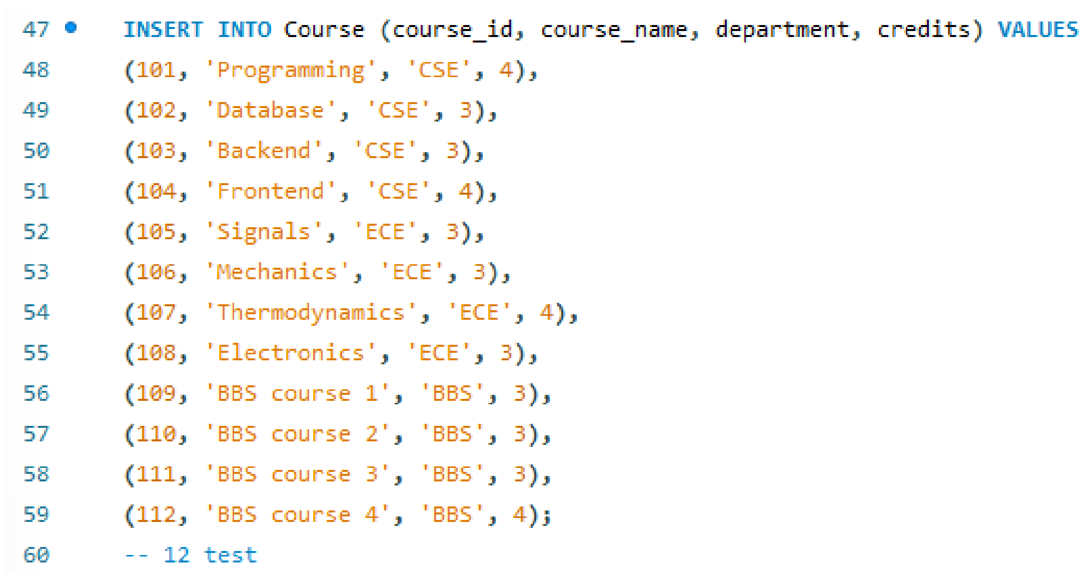


```sql
INSERT INTO Student (student_id, name, department, year) VALUES
(1, 'Typhon', 'CSE', 2),
(2, 'Eblana', 'CSE', 3),
(3, 'Elbanana', 'CSE', 1),
(4, 'Fangyi', 'CSE', 4),
(5, 'Perlica', 'ECE', 2),
(6, 'Chen', 'ECE', 3),
(7, 'Lavaetain', 'ECE', 1),
(8, 'Fluorite', 'ECE', 4),
(9, 'Nearl', 'BBS', 2),
(10, 'Lapipi', 'BBS', 3),
(11, 'Kalstit', 'BBS', 1),
(12, 'Amiya', 'BBS', 2),
(13, 'Hoolheyak', 'CSE', 2),
(14, 'Muelseye', 'CSE', 3),
(15, 'Dorothy', 'CSE', 1),
(16, 'Saria', 'CSE', 4);
-- 16 tset
```

Course

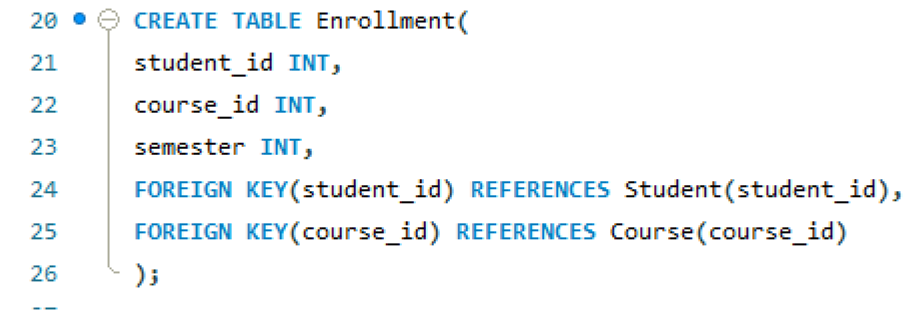


```sql
CREATE TABLE Course(
course_id INT NOT NULL,
course_name VARCHAR(50) NOT NULL,
department VARCHAR(50),
credits INT,
PRIMARY KEY(course_id)
);
```



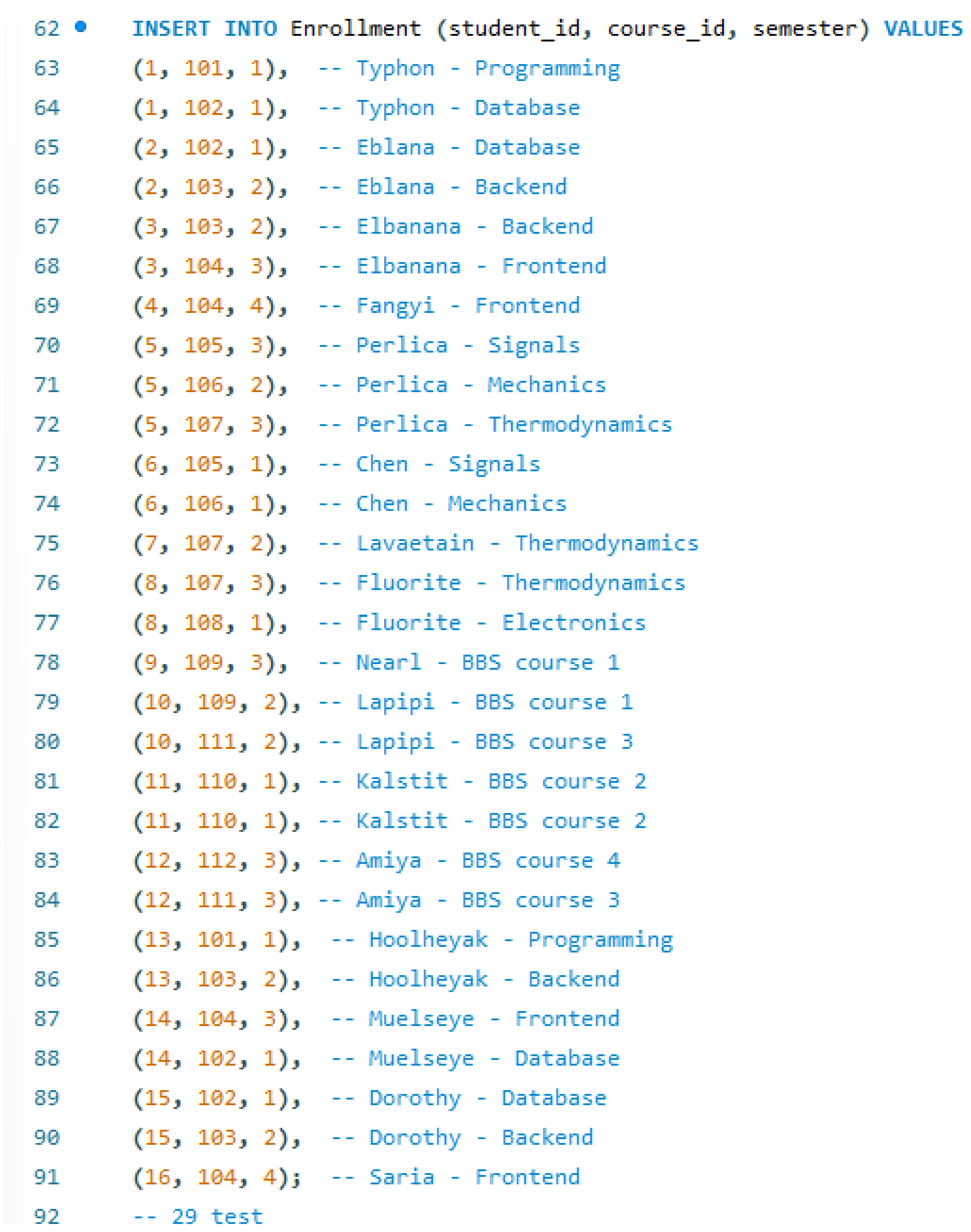
```sql
INSERT INTO Course (course_id, course_name, department, credits) VALUES
(101, 'Programming', 'CSE', 4),
(102, 'Database', 'CSE', 3),
(103, 'Backend', 'CSE', 3),
(104, 'Frontend', 'CSE', 4),
(105, 'Signals', 'ECE', 3),
(106, 'Mechanics', 'ECE', 3),
(107, 'Thermodynamics', 'ECE', 4),
(108, 'Electronics', 'ECE', 3),
(109, 'BBS course 1', 'BBS', 3),
(110, 'BBS course 2', 'BBS', 3),
(111, 'BBS course 3', 'BBS', 3),
(112, 'BBS course 4', 'BBS', 4);
-- 12 test
```

Enrollment



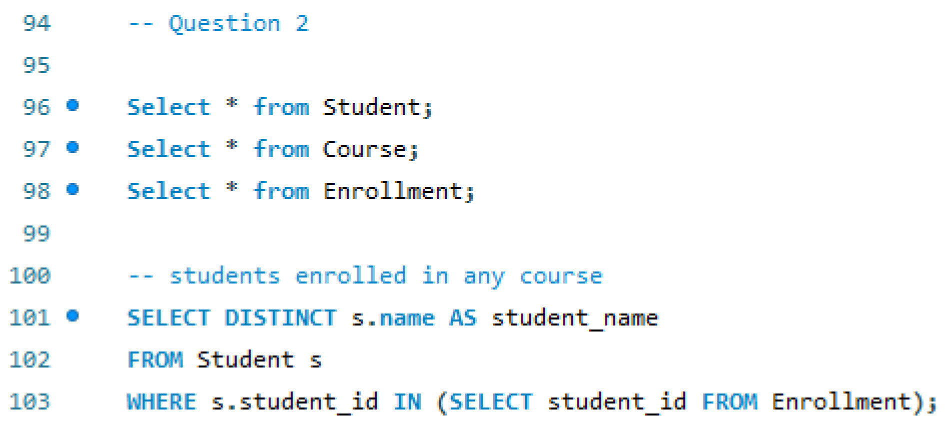
```sql
CREATE TABLE Enrollment(
student_id INT,
course_id INT,
semester INT,
FOREIGN KEY(student_id) REFERENCES Student(student_id),
FOREIGN KEY(course_id) REFERENCES Course(course_id)
);
```

- student_id as a FOREIGN KEY referencing STUDENT(student_id)
- course_id as a FOREIGN KEY referencing COURSE(course_id)

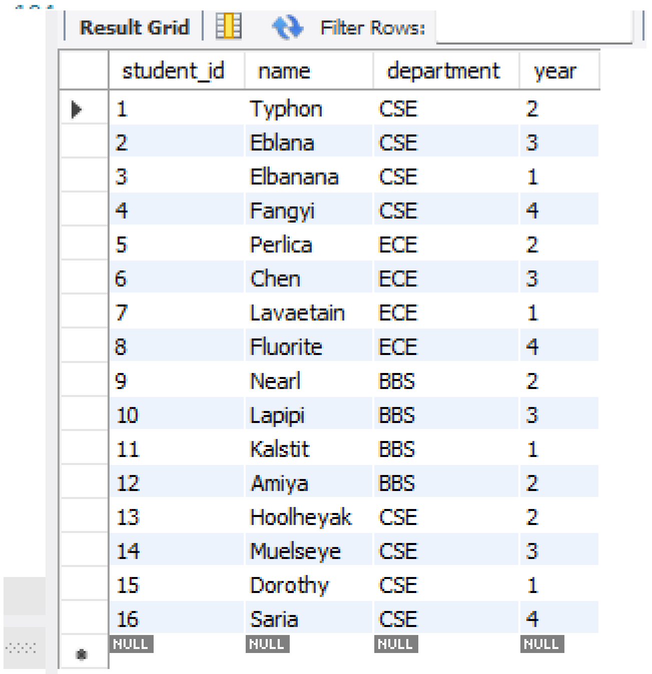


```sql
INSERT INTO Enrollment (student_id, course_id, semester) VALUES
(1, 101, 1),  -- Typhon - Programming
(1, 102, 1),  -- Typhon - Database
(2, 102, 1),  -- Eblana - Database
(2, 103, 2),  -- Eblana - Backend
(3, 103, 2),  -- Elbanana - Backend
(3, 104, 3),  -- Elbanana - Frontend
(4, 104, 4),  -- Fangyi - Frontend
(5, 105, 3),  -- Perlica - Signals
(5, 106, 2),  -- Perlica - Mechanics
(5, 107, 3),  -- Perlica - Thermodynamics
(6, 105, 1),  -- Chen - Signals
(6, 106, 1),  -- Chen - Mechanics
(7, 107, 2),  -- Lavaetain - Thermodynamics
(8, 107, 3),  -- Fluorite - Thermodynamics
(8, 108, 1),  -- Fluorite - Electronics
(9, 109, 3),  -- Nearl - BBS course 1
(10, 109, 2), -- Lapipi - BBS course 1
(10, 111, 2), -- Lapipi - BBS course 3
(11, 110, 1), -- Kalstit - BBS course 2
(11, 110, 1), -- Kalstit - BBS course 2
(12, 112, 3), -- Amiya - BBS course 4
(12, 111, 3), -- Amiya - BBS course 3
(13, 101, 1),  -- Hoolheyak - Programming
(13, 103, 2),  -- Hoolheyak - Backend
(14, 104, 3),  -- Muelseye - Frontend
(14, 102, 1),  -- Muelseye - Database
(15, 102, 1),  -- Dorothy - Database
(15, 103, 2),  -- Dorothy - Backend
(16, 104, 4);  -- Saria - Frontend
-- 29 test
```

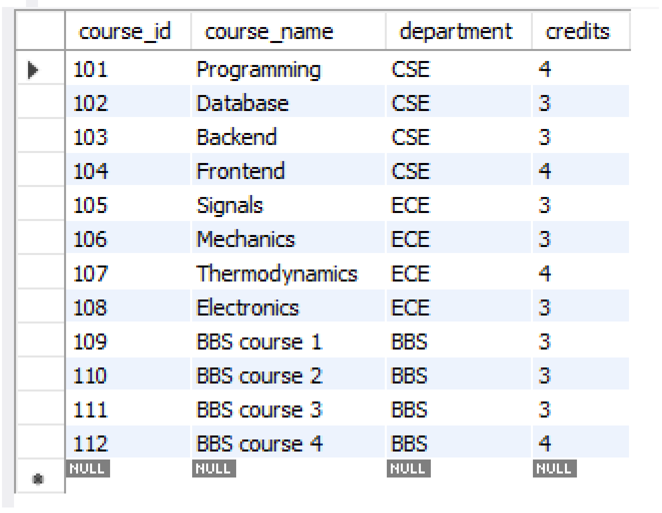
2. Write an SQL query to display the names of all students enrolled in any course or display all student, course and enrolment tables. Include outcome in the assignment file also.



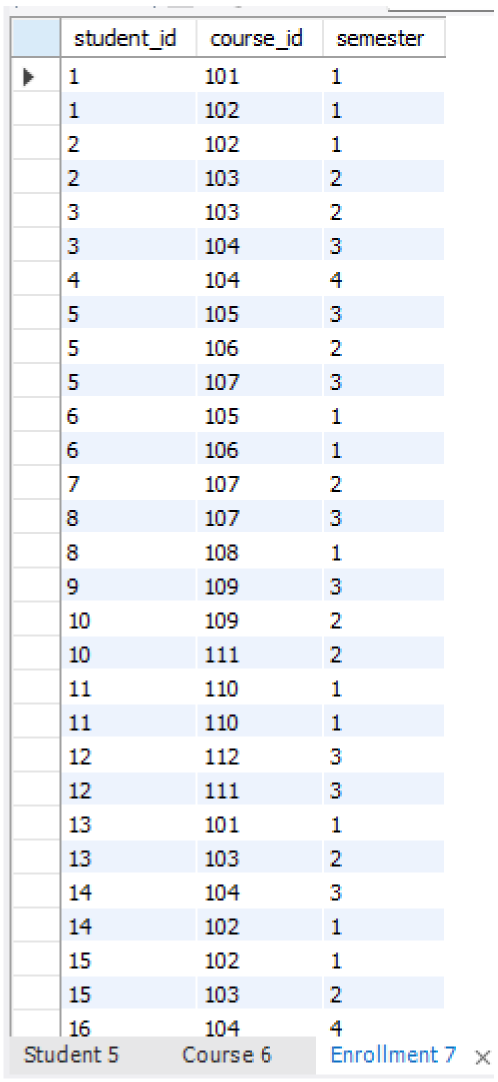
```sql
-- Question 2

Select * from Student;
Select * from Course;
Select * from Enrollment;

-- students enrolled in any course
SELECT DISTINCT s.name AS student_name
FROM Student s
WHERE s.student_id IN (SELECT student_id FROM Enrollment);
```



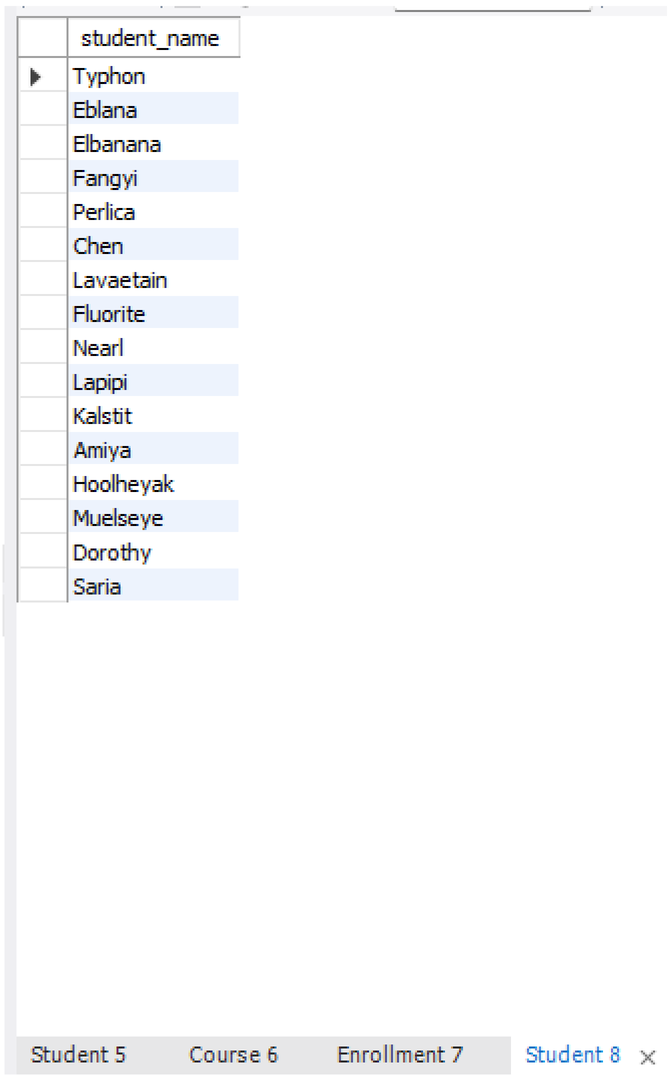
student_id	name	department	year
1	Typhon	CSE	2
2	Eblana	CSE	3
3	Elbanana	CSE	1
4	Fangyi	CSE	4
5	Perlica	ECE	2
6	Chen	ECE	3
7	Lavaetain	ECE	1
8	Fluorite	ECE	4
9	Nearl	BBS	2
10	Lapipi	BBS	3
11	Kalstit	BBS	1
12	Amiya	BBS	2
13	Hoolheyak	CSE	2
14	Muelseye	CSE	3
15	Dorothy	CSE	1
16	Saria	CSE	4
NULL	NULL	NULL	NULL



course_id	course_name	department	credits
101	Programming	CSE	4
102	Database	CSE	3
103	Backend	CSE	3
104	Frontend	CSE	4
105	Signals	ECE	3
106	Mechanics	ECE	3
107	Thermodynamics	ECE	4
108	Electronics	ECE	3
109	BBS course 1	BBS	3
110	BBS course 2	BBS	3
111	BBS course 3	BBS	3
112	BBS course 4	BBS	4
NULL	NULL	NULL	NULL

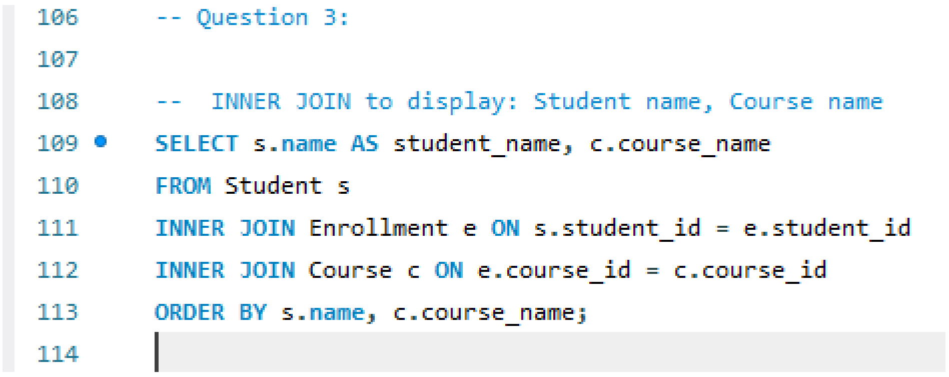


student_id	course_id	semester
1	101	1
1	102	1
2	102	1
2	103	2
3	103	2
3	104	3
4	104	4
5	105	3
5	106	2
5	107	3
6	105	1
6	106	1
7	107	2
8	107	3
8	108	1
9	109	3
10	109	2
10	111	2
11	110	1
11	110	1
12	112	3
12	111	3
13	101	1
13	103	2
14	104	3
14	102	1
15	102	1
15	103	2
16	104	4

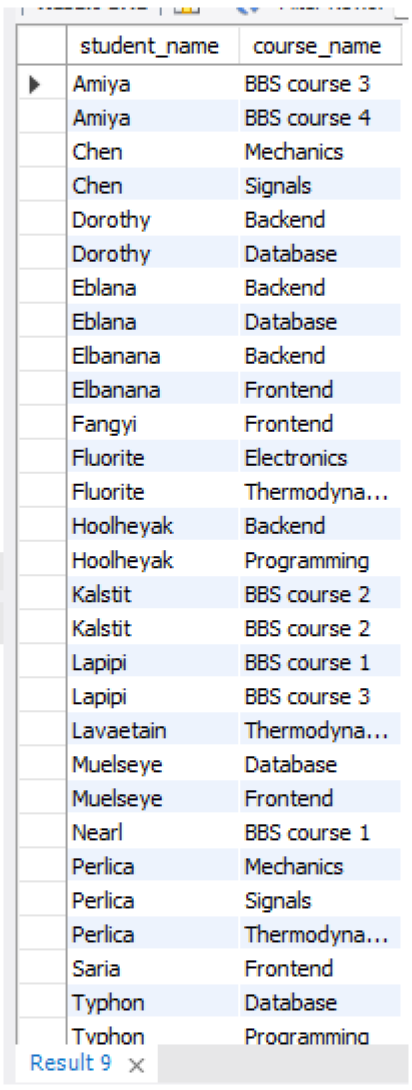


	student_name
▶	Typhon
	Eblana
	Elbanana
	Fangyi
	Perlica
	Chen
	Lavaetain
	Fluorite
	Nearl
	Lapipi
	Kalstit
	Amiya
	Hoolheyak
	Muelseye
	Dorothy
	Saria

3. Write an SQL query using INNER JOIN to display:
- Student name
- Course name



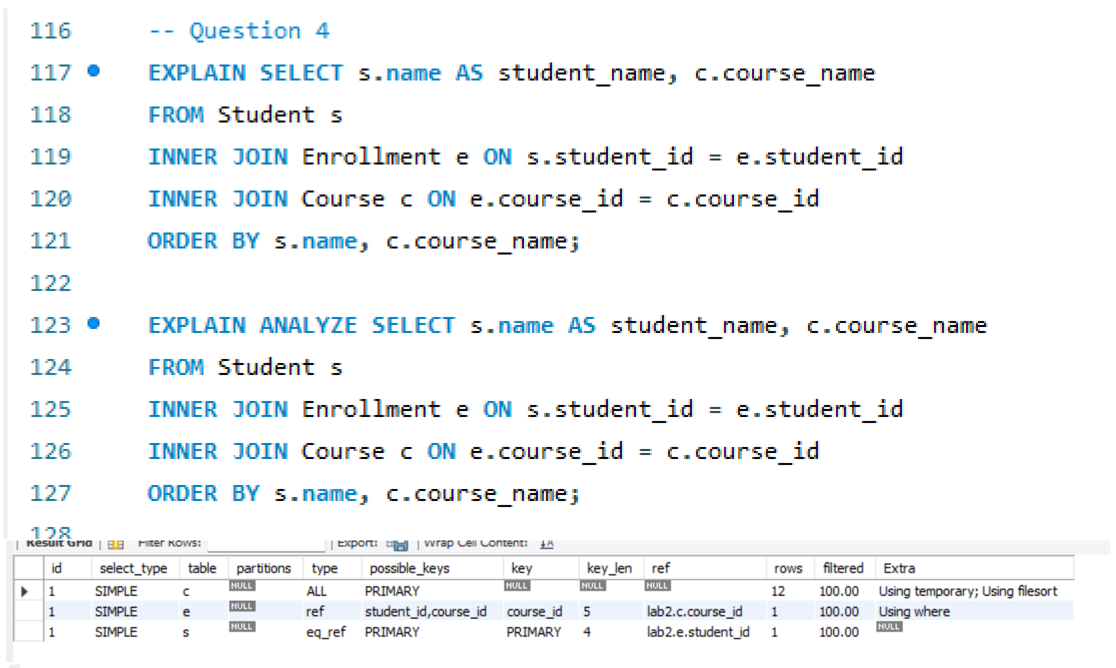
```
-- Question 3:

--  INNER JOIN to display: Student name, Course name
SELECT s.name AS student_name, c.course_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
ORDER BY s.name, c.course_name;
```



student_name	course_name
Amiya	BBS course 3
Amiya	BBS course 4
Chen	Mechanics
Chen	Signals
Dorothy	Backend
Dorothy	Database
Eblana	Backend
Eblana	Database
Elbanana	Backend
Elbanana	Frontend
Fangyi	Frontend
Fluorite	Electronics
Fluorite	Thermodyna...
Hoolheyak	Backend
Hoolheyak	Programming
Kalstit	BBS course 2
Kalstit	BBS course 2
Lapipi	BBS course 1
Lapipi	BBS course 3
Lavaetain	Thermodyna...
Muelseye	Database
Muelseye	Frontend
Nearl	BBS course 1
Perlica	Mechanics
Perlica	Signals
Perlica	Thermodyna...
Saria	Frontend
Typhon	Database
Typhon	Programming

4. Use the EXPLAIN command on the join query discussed in Question 3 to identify:

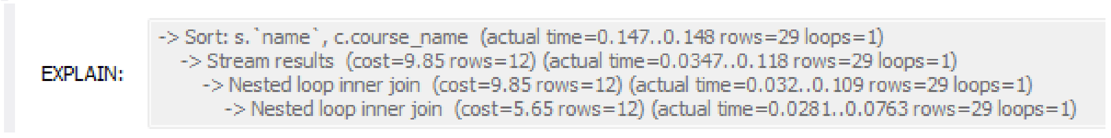
- **The join algorithm used**
- **The order of table access**
- **Note - DBMS uses a Nested Loop / Hash Join based on optimization**



```sql
-- Question 4
EXPLAIN SELECT s.name AS student_name, c.course_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
ORDER BY s.name, c.course_name;

EXPLAIN ANALYZE SELECT s.name AS student_name, c.course_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
ORDER BY s.name, c.course_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	ALL	PRIMARY	NULL	NULL	NULL	12	100.00	Using temporary; Using filesort
1	SIMPLE	e	NULL	ref	student_id,course_id	course_id	5	lab2.c.course_id	1	100.00	Using where
1	SIMPLE	s	NULL	eq_ref	PRIMARY	PRIMARY	4	lab2.e.student_id	1	100.00	NULL

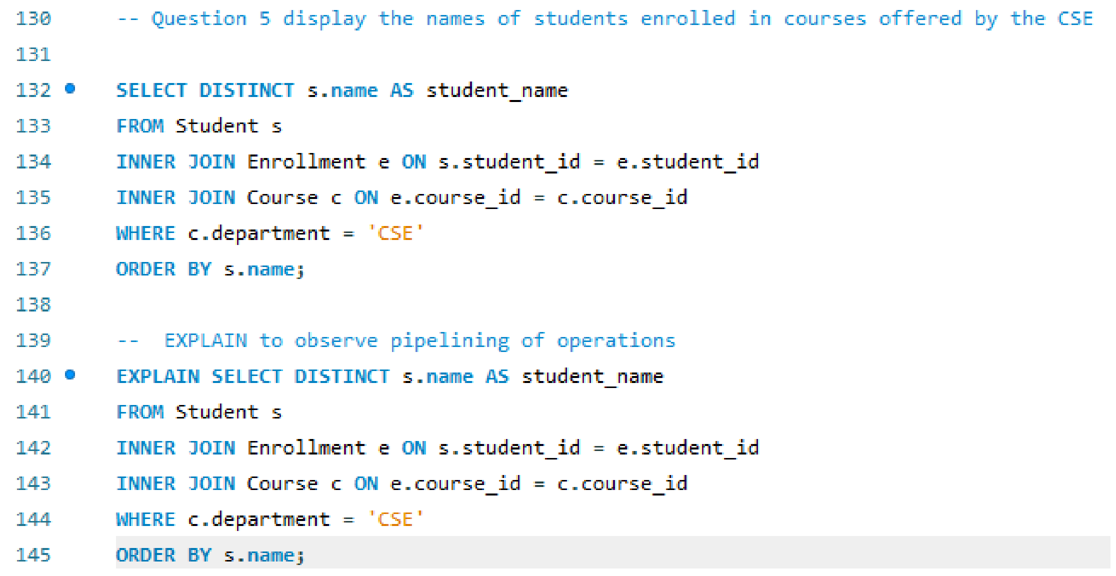


EXPLAIN:

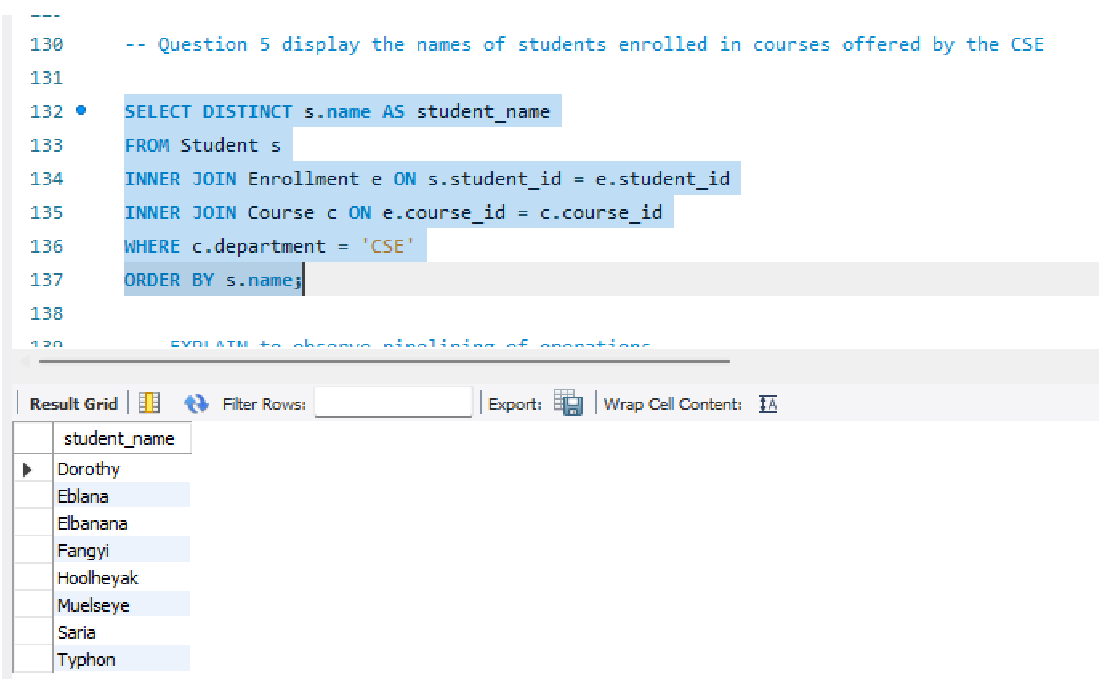
```
-> Sort: s.`name`, c.course_name  (actual time=0.147..0.148 rows=29 loops=1)
    -> Stream results  (cost=9.85 rows=12) (actual time=0.0347..0.118 rows=29 loops=1)
        -> Nested loop inner join  (cost=9.85 rows=12) (actual time=0.032..0.109 rows=29 loops=1)
            -> Nested loop inner join  (cost=5.65 rows=12) (actual time=0.0281..0.0763 rows=29 loops=1)
```

- The join algorithm used: Nested Loop Join
- The order of table access: Course -> Enrollment -> Student

5. Write an SQL query to display the names of students enrolled in courses offered by the CSE department. Use EXPLAIN to observe pipelining of operations

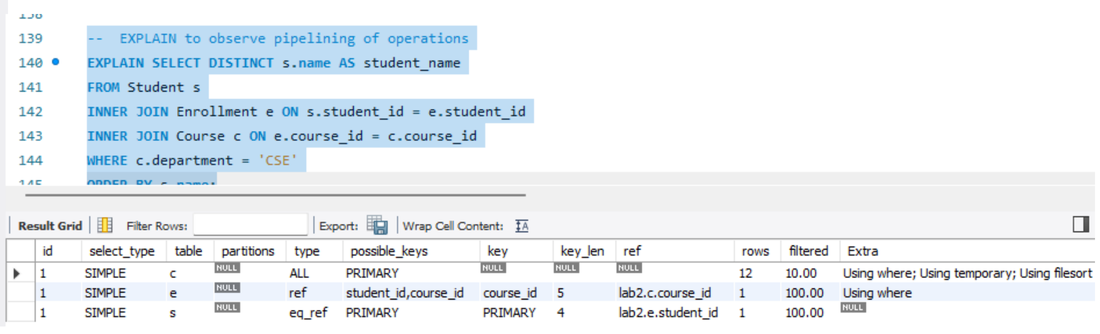


```sql
-- Question 5 display the names of students enrolled in courses offered by the CSE

SELECT DISTINCT s.name AS student_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
WHERE c.department = 'CSE'
ORDER BY s.name;

--  EXPLAIN to observe pipelining of operations
EXPLAIN SELECT DISTINCT s.name AS student_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
WHERE c.department = 'CSE'
ORDER BY s.name;
```



```sql
-- Question 5 display the names of students enrolled in courses offered by the CSE

SELECT DISTINCT s.name AS student_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
WHERE c.department = 'CSE'
ORDER BY s.name;
```

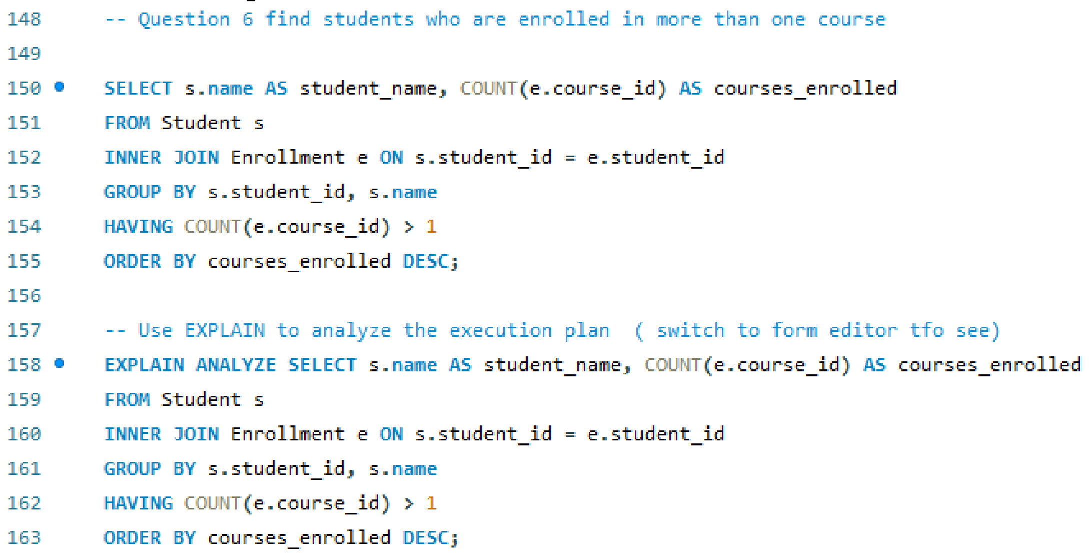
student_name
Dorothy
Eblana
Elbanana
Fangyi
Hoolheyak
Muelseye
Saria
Typhon



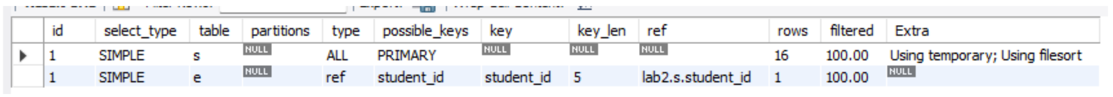
```sql
-- EXPLAIN to observe pipelining of operations
EXPLAIN SELECT DISTINCT s.name AS student_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
WHERE c.department = 'CSE'
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	ALL	PRIMARY	NULL	NULL	NULL	12	10.00	Using where; Using temporary; Using filesort
1	SIMPLE	e	NULL	ref	student_id,course_id	course_id	5	lab2.c.course_id	1	100.00	Using where
1	SIMPLE	s	NULL	eq_ref	PRIMARY	PRIMARY	4	lab2.e.student_id	1	100.00	NULL

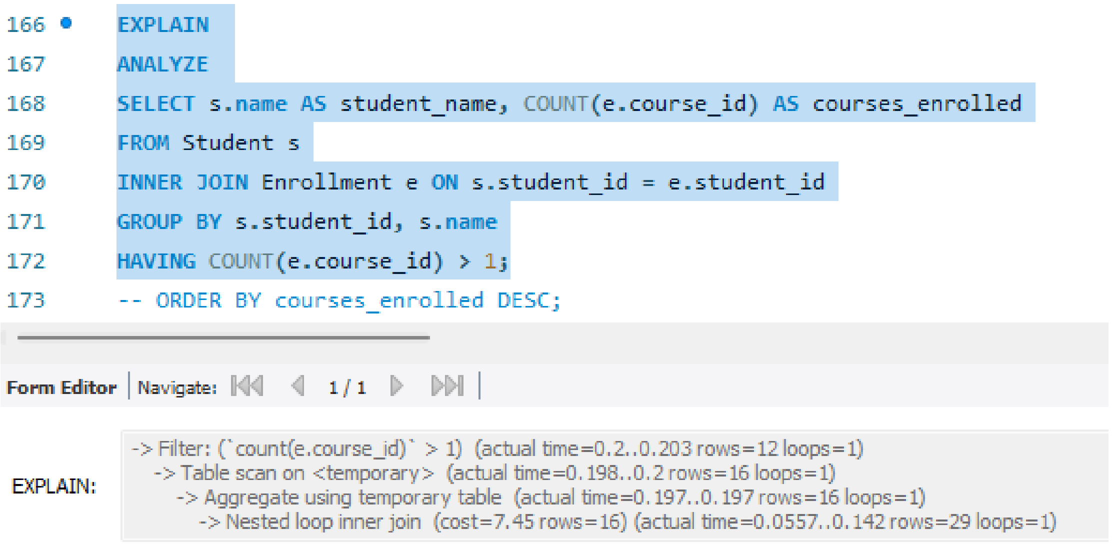
6. Write an SQL query to find students who are enrolled in more than one course. Use
EXPLAIN to analyze the execution plan and note:
- Estimated cost
- Number of rows processed



```sql
-- Question 6 find students who are enrolled in more than one course

SELECT s.name AS student_name, COUNT(e.course_id) AS courses_enrolled
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
GROUP BY s.student_id, s.name
HAVING COUNT(e.course_id) > 1
ORDER BY courses_enrolled DESC;

-- Use EXPLAIN to analyze the execution plan  ( switch to form editor tfo see)
EXPLAIN ANALYZE SELECT s.name AS student_name, COUNT(e.course_id) AS courses_enrolled
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
GROUP BY s.student_id, s.name
HAVING COUNT(e.course_id) > 1
ORDER BY courses_enrolled DESC;
```



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	s	NULL	ALL	PRIMARY	NULL	NULL	NULL	16	100.00	Using temporary; Using filesort
1	SIMPLE	e	NULL	ref	student_id	student_id	5	lab2.s.student_id	1	100.00	NULL



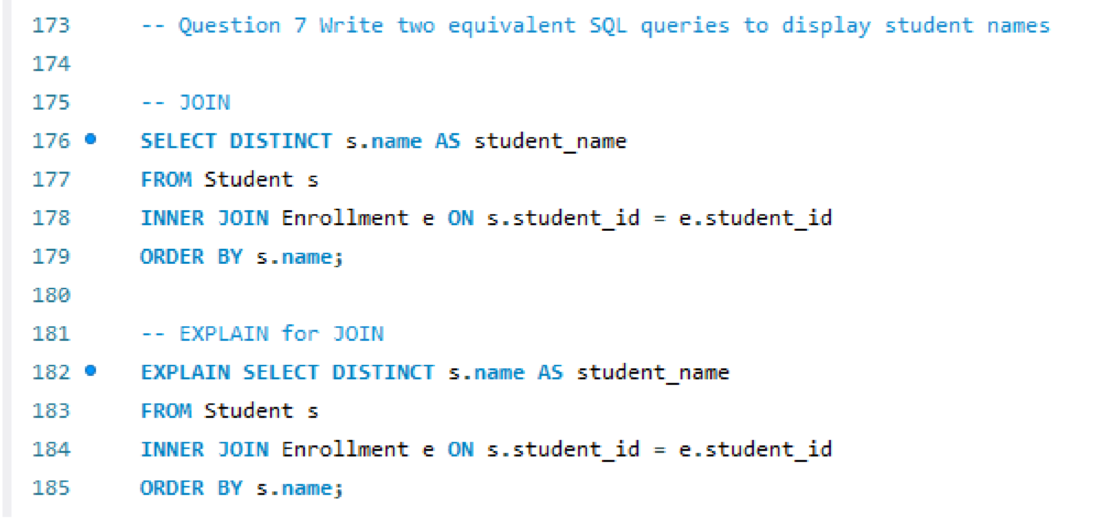
```
EXPLAIN
ANALYZE
SELECT s.name AS student_name, COUNT(e.course_id) AS courses_enrolled
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
GROUP BY s.student_id, s.name
HAVING COUNT(e.course_id) > 1;
-- ORDER BY courses_enrolled DESC;
```

```
-> Filter: (`count(e.course_id)` > 1)  (actual time=0.2..0.203 rows=12 loops=1)
    -> Table scan on <temporary>  (actual time=0.198..0.2 rows=16 loops=1)
        -> Aggregate using temporary table  (actual time=0.197..0.197 rows=16 loops=1)
            -> Nested loop inner join  (cost=7.45 rows=16) (actual time=0.0557..0.142 rows=29 loops=1)
```

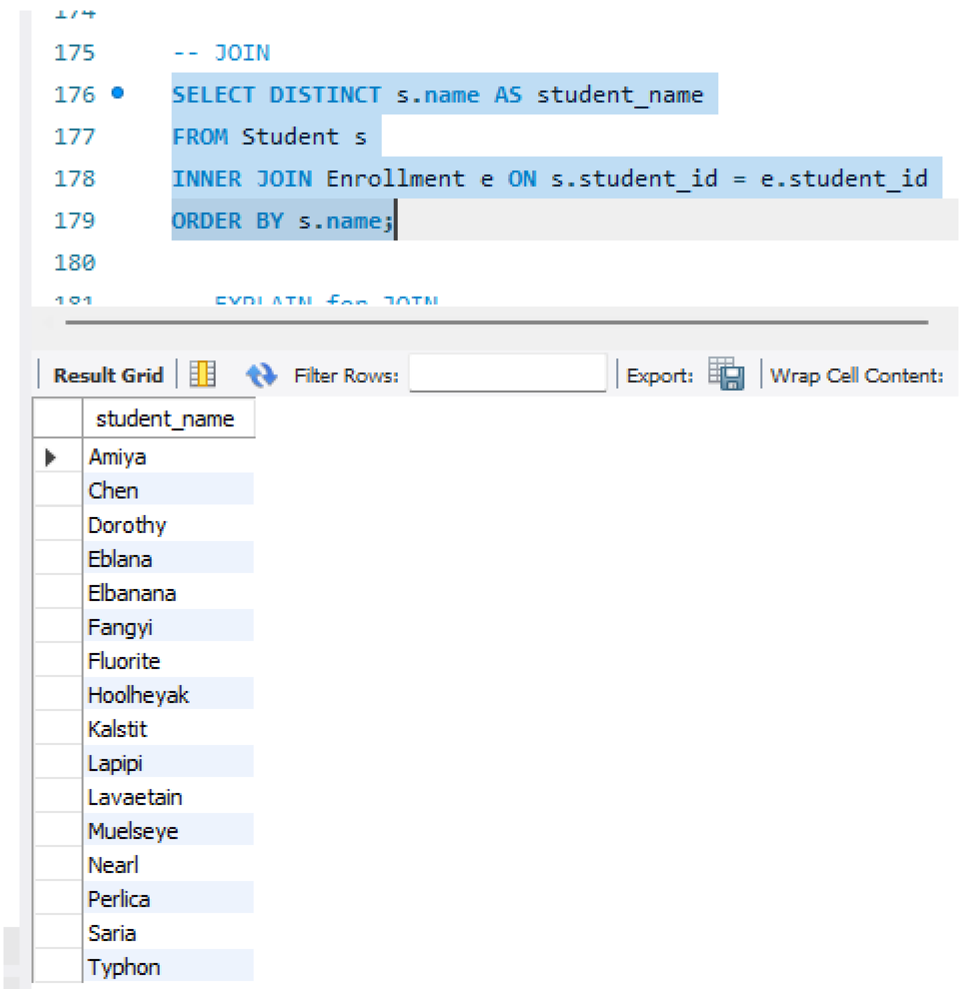
- Estimated cost: 7.45
- Number of rows processed:
+ 16 rows during aggregation
+ 12 rows returned after HAVING and ORDER BY

7. Write two equivalent SQL queries to display student names: Using JOIN and Using a subquery. Use EXPLAIN to compare the execution plans of both queries. Explain outcome in brief of both in the assignment work

JOIN

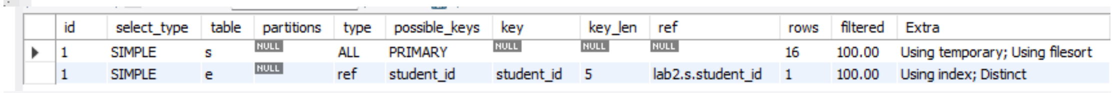


```
-- Question 7 Write two equivalent SQL queries to display student names

-- JOIN
SELECT DISTINCT s.name AS student_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
ORDER BY s.name;

-- EXPLAIN for JOIN
EXPLAIN SELECT DISTINCT s.name AS student_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
ORDER BY s.name;
```

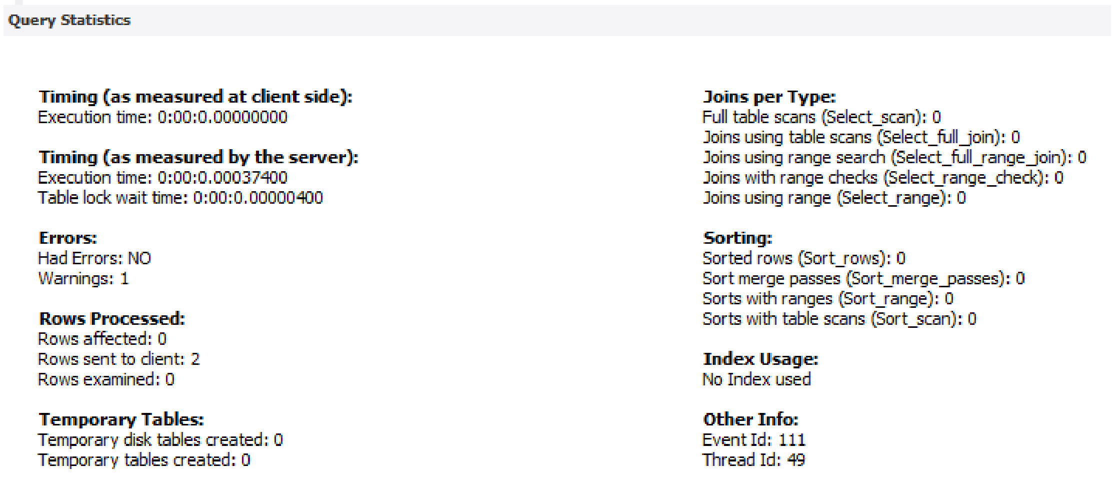


```sql
-- JOIN
SELECT DISTINCT s.name AS student_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
ORDER BY s.name;
```

	student_name
▶	Amiya
	Chen
	Dorothy
	Eblana
	Elbanana
	Fangyi
	Fluorite
	Hoolheyak
	Kalstit
	Lapipi
	Lavaetain
	Muelseye
	Nearl
	Perlica
	Saria
	Typhon



	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	s	NULL	ALL	PRIMARY	NULL	NULL	NULL	16	100.00	Using temporary; Using filesort
	1	SIMPLE	e	NULL	ref	student_id	student_id	5	lab2.s.student_id	1	100.00	Using index; Distinct



Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00037400
Table lock wait time: 0:00:0.00000400

Errors:
Had Errors: NO
Warnings: 1

Rows Processed:
Rows affected: 0
Rows sent to client: 2
Rows examined: 0

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

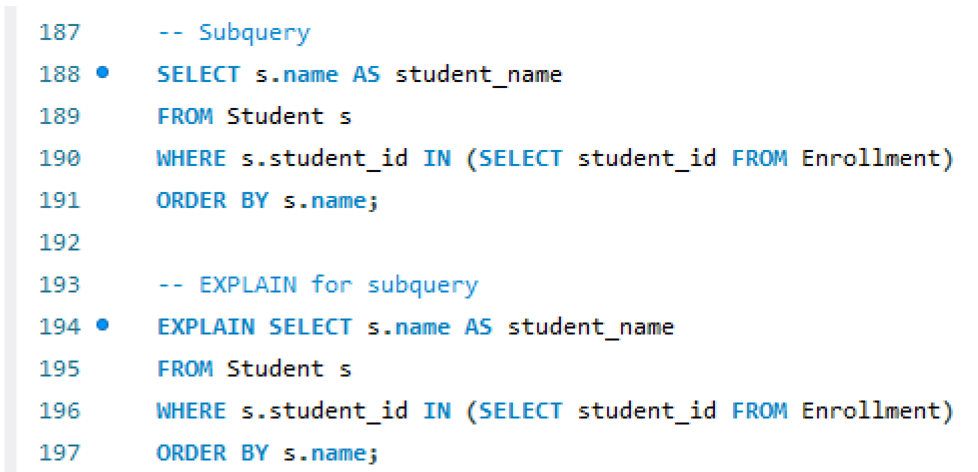
Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

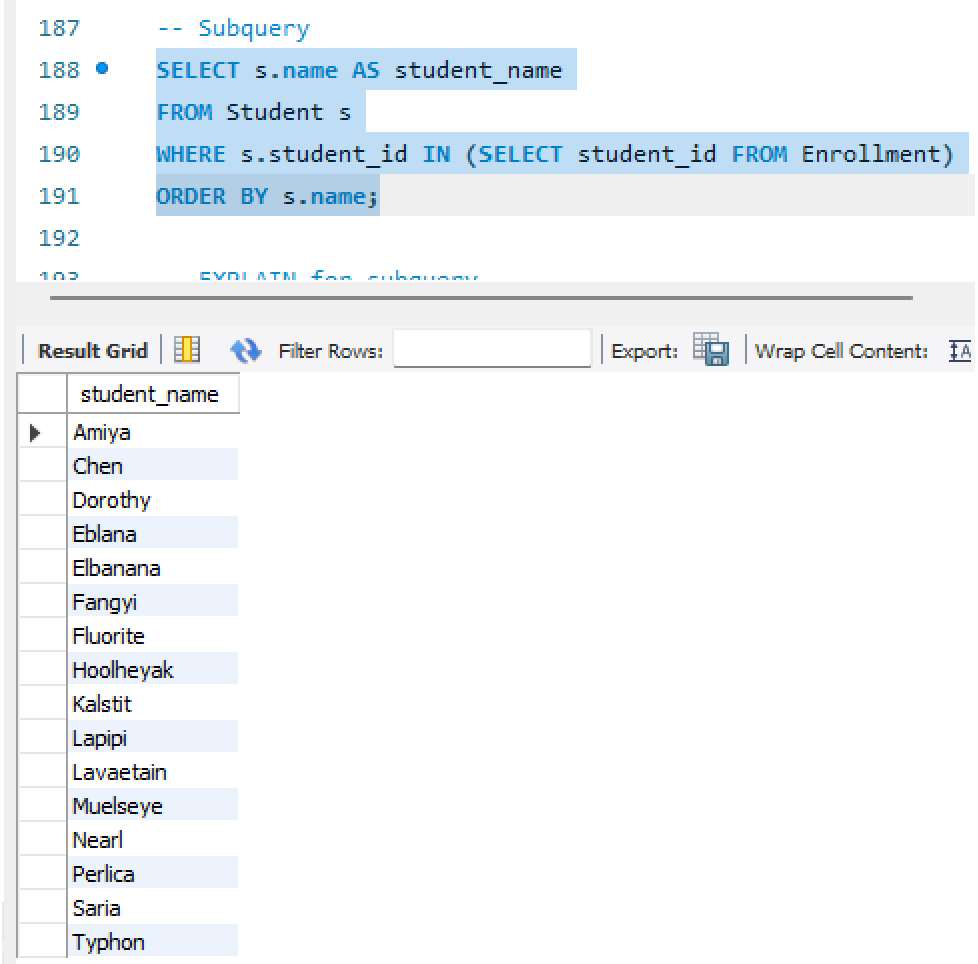
Index Usage:
No Index used

Other Info:
Event Id: 111
Thread Id: 49

Subquery

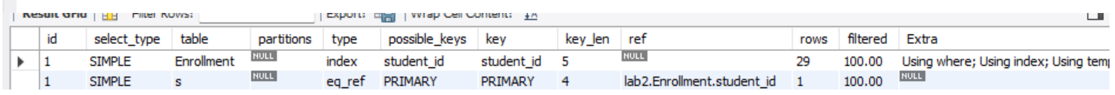


```sql
-- Subquery
SELECT s.name AS student_name
FROM Student s
WHERE s.student_id IN (SELECT student_id FROM Enrollment)
ORDER BY s.name;

-- EXPLAIN for subquery
EXPLAIN SELECT s.name AS student_name
FROM Student s
WHERE s.student_id IN (SELECT student_id FROM Enrollment)
ORDER BY s.name;
```

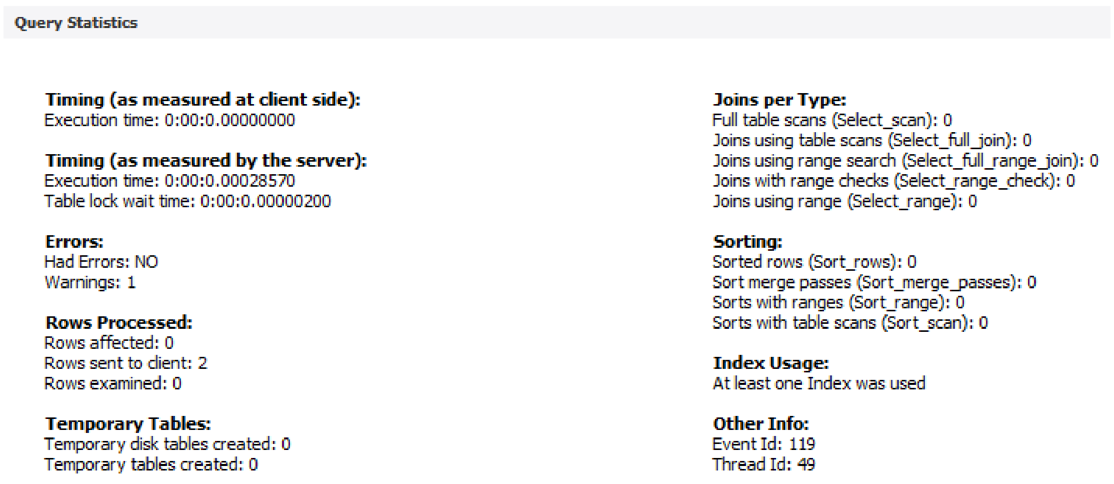


```sql
-- Subquery
SELECT s.name AS student_name
FROM Student s
WHERE s.student_id IN (SELECT student_id FROM Enrollment)
ORDER BY s.name;

-- EXPLAIN for subquery
```

student_name
Amiya
Chen
Dorothy
Eblana
Elbanana
Fangyi
Fluorite
Hoolheyak
Kalstit
Lapipi
Lavaetain
Muelseye
Nearl
Perlica
Saria
Typhon



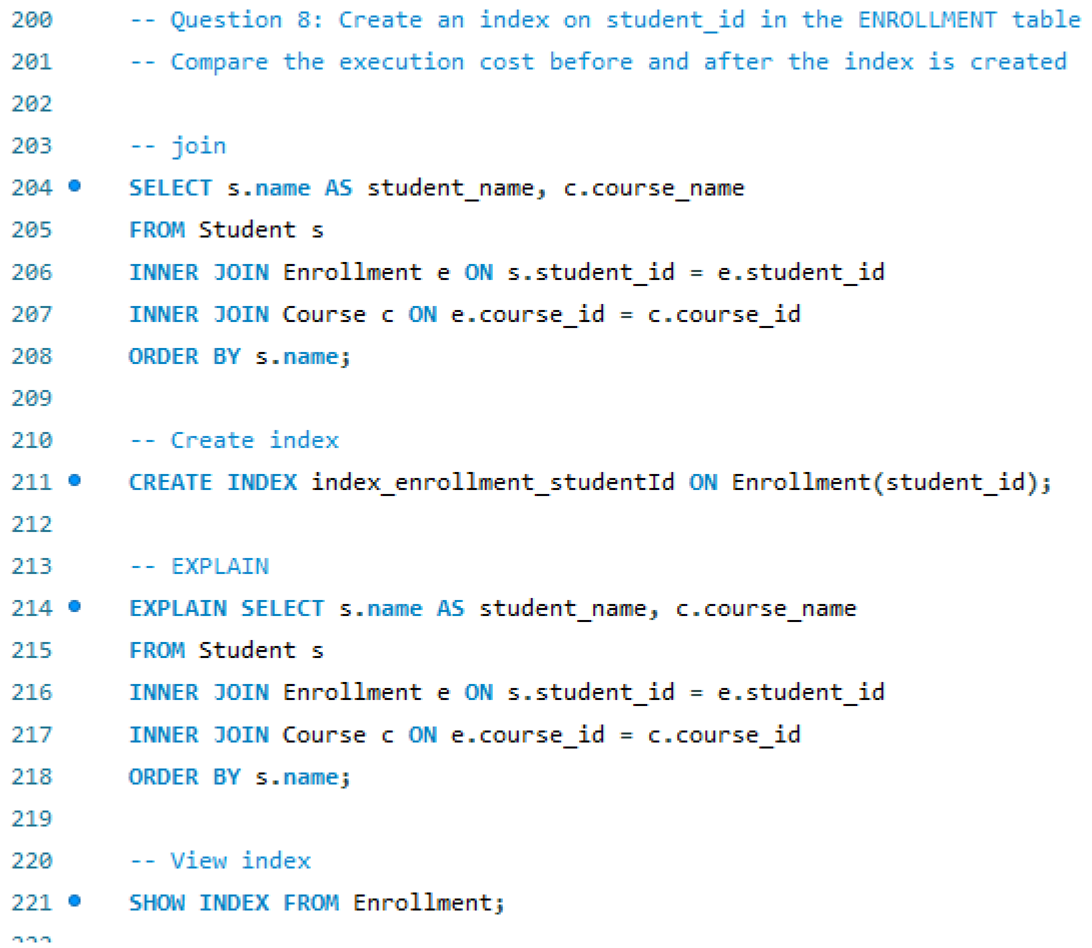
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Enrollment	NULL	index	student_id	student_id	5	NULL	29	100.00	Using where; Using index; Using tem
1	SIMPLE	s	NULL	eq_ref	PRIMARY	PRIMARY	4	lab2.Enrollment.student_id	1	100.00	NULL



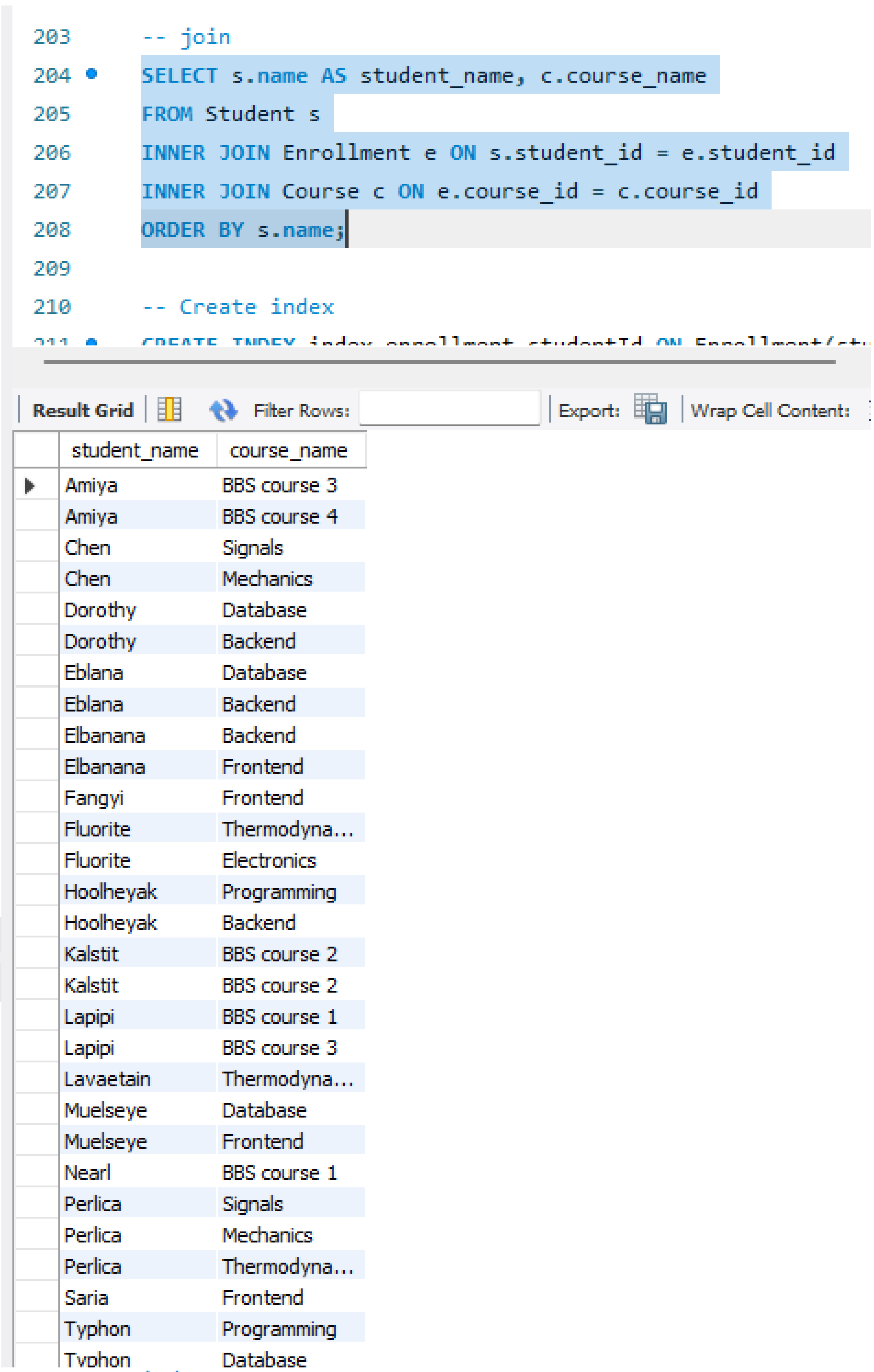
```
Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00028570
Table lock wait time: 0:00:0.00000200

Errors:
Had Errors: NO
Warnings: 1

Rows Processed:
Rows affected: 0
Rows sent to client: 2
Rows examined: 0

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

Other Info:
Event Id: 119
Thread Id: 49
```

Explain:
Inner Join: Join to connect 3 table then it scan the table and filter
Subquery: It only join connect 2 table Student with Enrollment then scan and lookup
Because lookup is faster than join so subquery is faster than innerjoin

8. Create an index on student_id in the ENROLLMENT table. Execute a join query again and use EXPLAIN. Compare the execution cost before and after the index is created

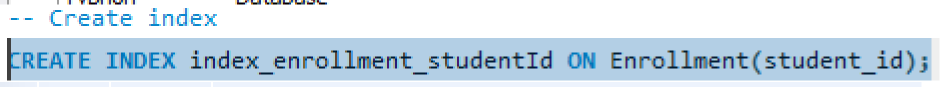


```sql
-- Question 8: Create an index on student_id in the ENROLLMENT table
-- Compare the execution cost before and after the index is created

-- join
SELECT s.name AS student_name, c.course_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
ORDER BY s.name;

-- Create index
CREATE INDEX index_enrollment_studentId ON Enrollment(student_id);

-- EXPLAIN
EXPLAIN SELECT s.name AS student_name, c.course_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
ORDER BY s.name;

-- View index
SHOW INDEX FROM Enrollment;
```

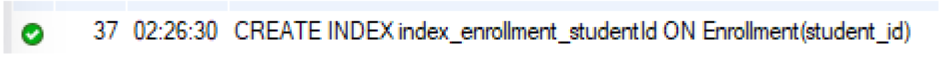


```
-- join
SELECT s.name AS student_name, c.course_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
ORDER BY s.name;

-- Create index
CREATE INDEX index_enrollment_studentId ON Enrollment(stu
```

student_name	course_name
Amiya	BBS course 3
Amiya	BBS course 4
Chen	Signals
Chen	Mechanics
Dorothy	Database
Dorothy	Backend
Eblana	Database
Eblana	Backend
Elbanana	Backend
Elbanana	Frontend
Fangyi	Frontend
Fluorite	Thermodyna...
Fluorite	Electronics
Hoolheyak	Programming
Hoolheyak	Backend
Kalstit	BBS course 2
Kalstit	BBS course 2
Lapipi	BBS course 1
Lapipi	BBS course 3
Lavaetain	Thermodyna...
Muelseye	Database
Muelseye	Frontend
Nearl	BBS course 1
Perlica	Signals
Perlica	Mechanics
Perlica	Thermodyna...
Saria	Frontend
Typhon	Programming
Typhon	Database

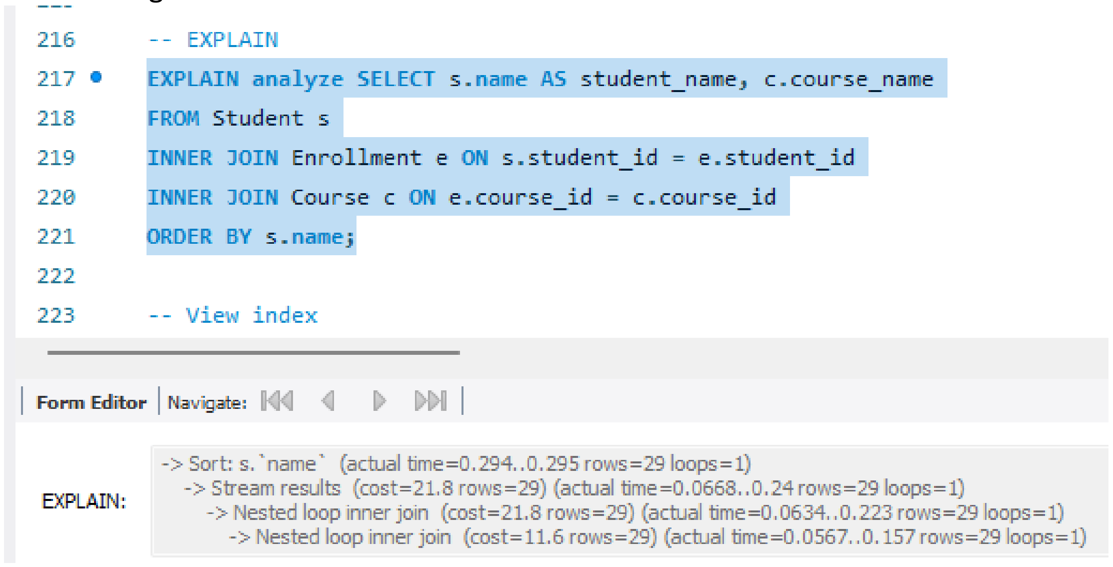


```
-- Create index
CREATE INDEX index_enrollment_studentId ON Enrollment(student_id);
```



```
37  02:26:30  CREATE INDEX index_enrollment_studentId ON Enrollment(student_id)
```

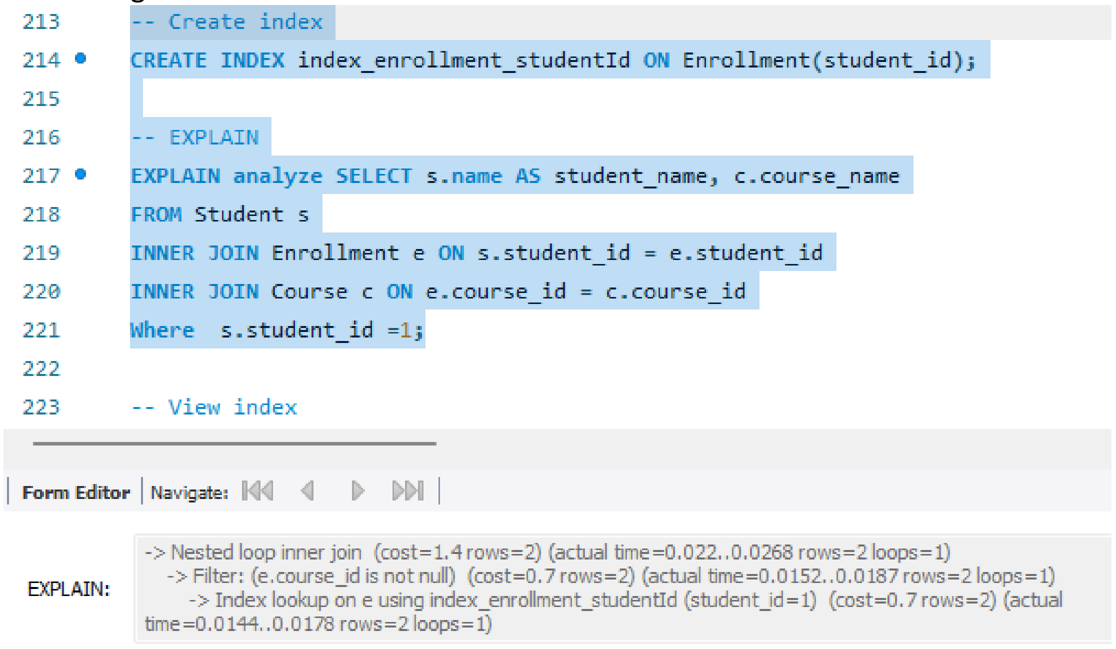
Before using index:



```sql
-- EXPLAIN
EXPLAIN analyze SELECT s.name AS student_name, c.course_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
ORDER BY s.name;

-- View index
```

```
EXPLAIN: -> Sort: s.`name`  (actual time=0.294..0.295 rows=29 loops=1)
            -> Stream results  (cost=21.8 rows=29) (actual time=0.0668..0.24 rows=29 loops=1)
              -> Nested loop inner join  (cost=21.8 rows=29) (actual time=0.0634..0.223 rows=29 loops=1)
                -> Nested loop inner join  (cost=11.6 rows=29) (actual time=0.0567..0.157 rows=29 loops=1)
```

After using index:

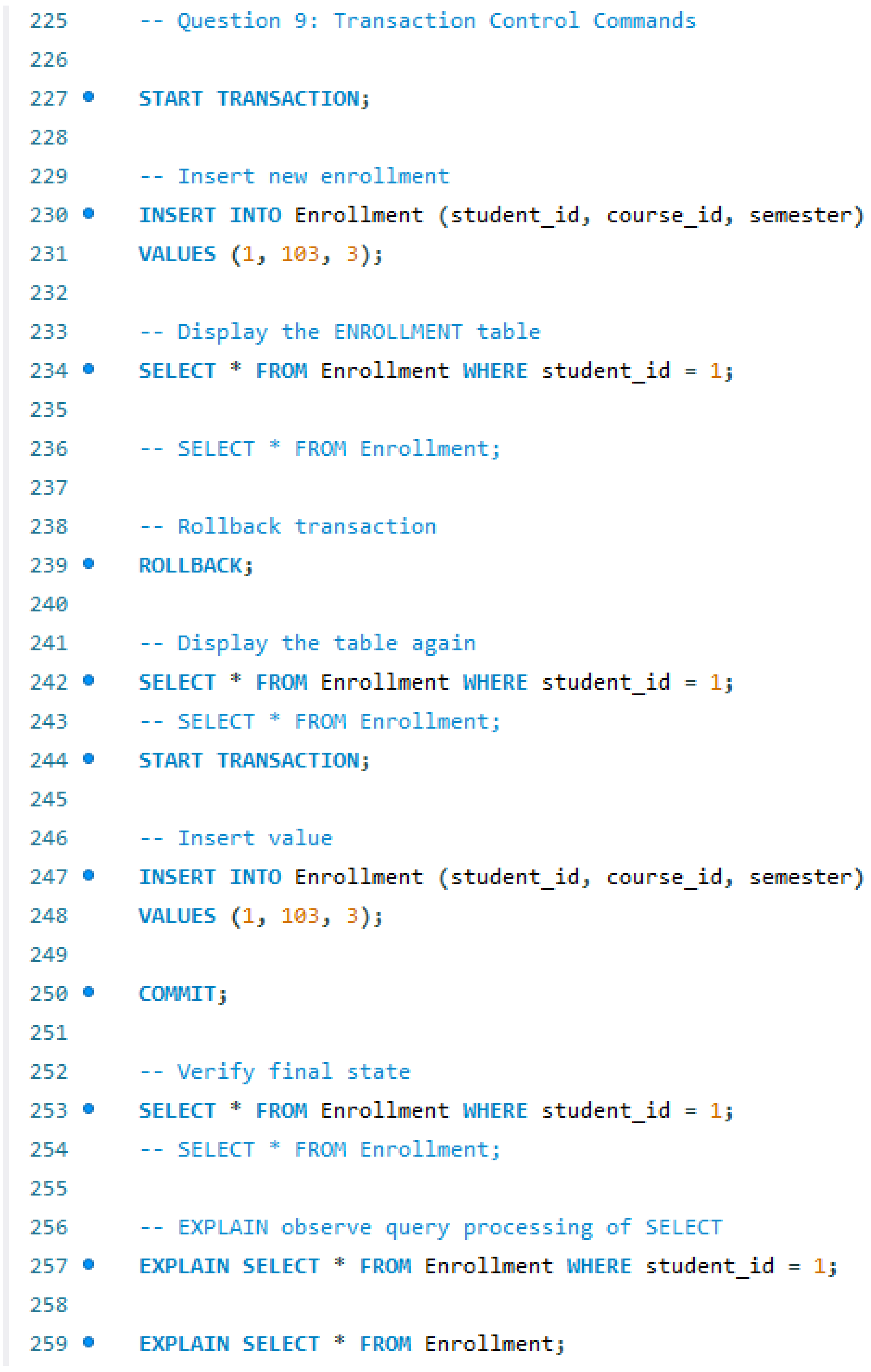


```sql
-- Create index
CREATE INDEX index_enrollment_studentId ON Enrollment(student_id);

-- EXPLAIN
EXPLAIN analyze SELECT s.name AS student_name, c.course_name
FROM Student s
INNER JOIN Enrollment e ON s.student_id = e.student_id
INNER JOIN Course c ON e.course_id = c.course_id
Where  s.student_id =1;

-- View index
```

```
EXPLAIN: -> Nested loop inner join  (cost=1.4 rows=2) (actual time=0.022..0.0268 rows=2 loops=1)
            -> Filter: (e.course_id is not null)  (cost=0.7 rows=2) (actual time=0.0152..0.0187 rows=2 loops=1)
              -> Index lookup on e using index_enrollment_studentId (student_id=1)  (cost=0.7 rows=2) (actual
         time=0.0144..0.0178 rows=2 loops=1)
```

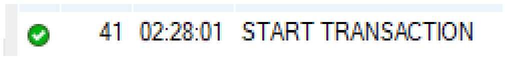
Not using index cost 11.6.
While using index cost 1.7

9. Consider the ENROLLMENT table. Perform the following operations using transaction control commands:



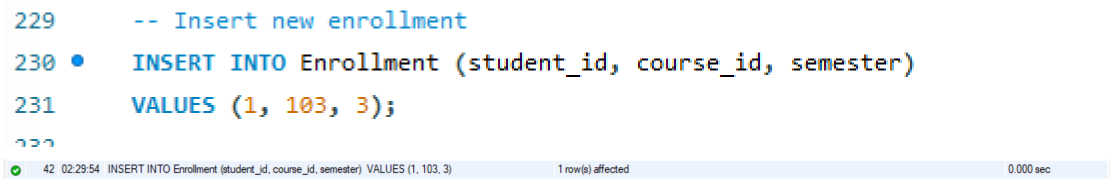
```sql
-- Question 9: Transaction Control Commands

START TRANSACTION;

-- Insert new enrollment
INSERT INTO Enrollment (student_id, course_id, semester)
VALUES (1, 103, 3);

-- Display the ENROLLMENT table
SELECT * FROM Enrollment WHERE student_id = 1;

-- SELECT * FROM Enrollment;

-- Rollback transaction
ROLLBACK;

-- Display the table again
SELECT * FROM Enrollment WHERE student_id = 1;
-- SELECT * FROM Enrollment;
START TRANSACTION;

-- Insert value
INSERT INTO Enrollment (student_id, course_id, semester)
VALUES (1, 103, 3);

COMMIT;

-- Verify final state
SELECT * FROM Enrollment WHERE student_id = 1;
-- SELECT * FROM Enrollment;

-- EXPLAIN observe query processing of SELECT
EXPLAIN SELECT * FROM Enrollment WHERE student_id = 1;

EXPLAIN SELECT * FROM Enrollment;
```

- Start a transaction.



```
41  02:28:01  START TRANSACTION
```

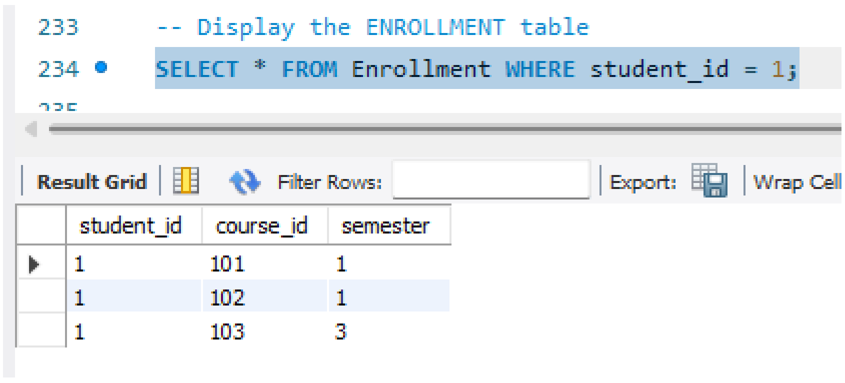
- Insert a new enrollment record for a student.



```sql
-- Insert new enrollment
INSERT INTO Enrollment (student_id, course_id, semester)
VALUES (1, 103, 3);
```

```
42  02:29:54  INSERT INTO Enrollment (student_id, course_id, semester)  VALUES (1, 103, 3)    1 row(s) affected    0.000 sec
```

- Display the ENROLLMENT table.

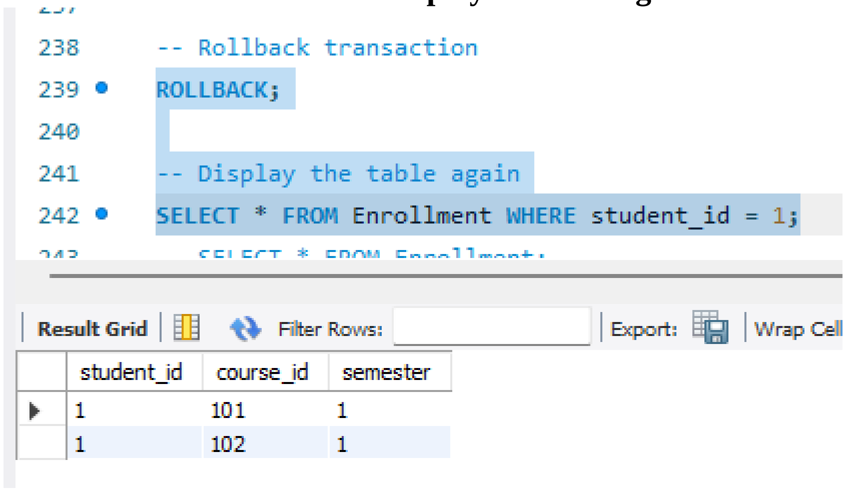


```sql
-- Display the ENROLLMENT table
SELECT * FROM Enrollment WHERE student_id = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell

student_id	course_id	semester
1	101	1
1	102	1
1	103	3

- **Rollback the transaction and display the table again.**

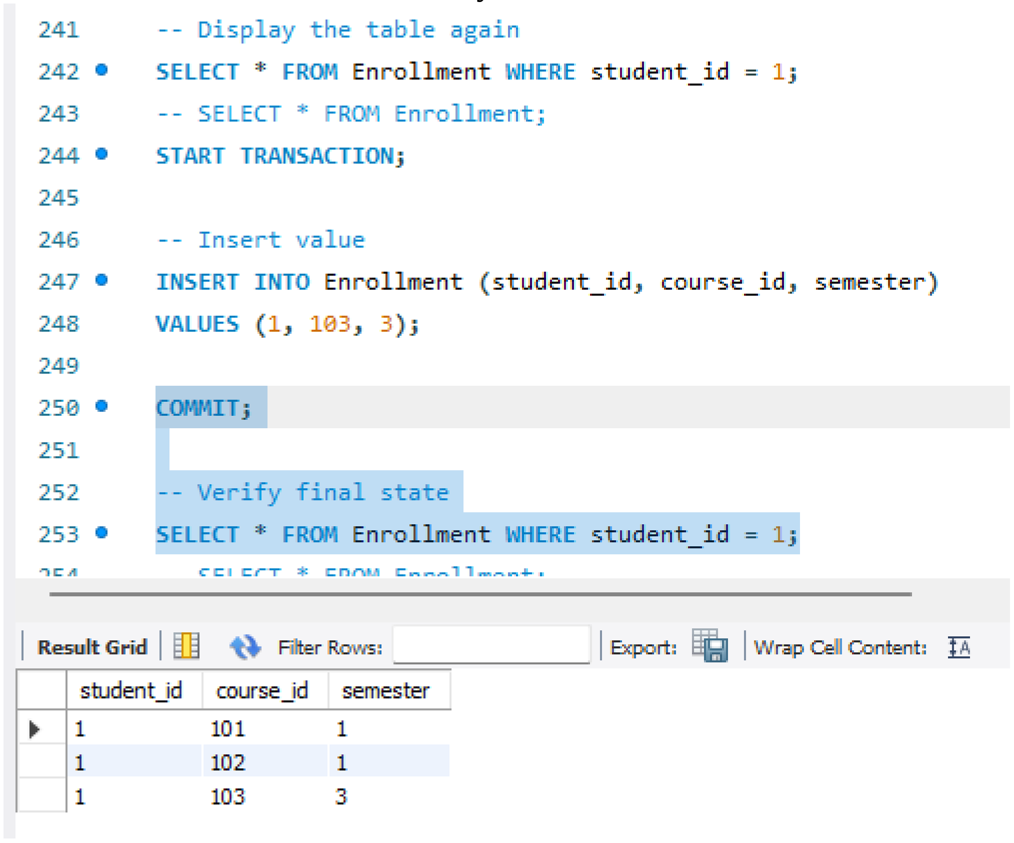


```sql
-- Rollback transaction
ROLLBACK;

-- Display the table again
SELECT * FROM Enrollment WHERE student_id = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell

student_id	course_id	semester
1	101	1
1	102	1

- **Commit the transaction and verify the final state.**

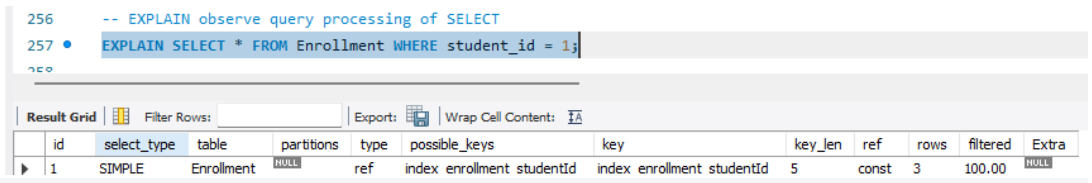


```sql
-- Display the table again
SELECT * FROM Enrollment WHERE student_id = 1;
-- SELECT * FROM Enrollment;
START TRANSACTION;

-- Insert value
INSERT INTO Enrollment (student_id, course_id, semester)
VALUES (1, 103, 3);

COMMIT;

-- Verify final state
SELECT * FROM Enrollment WHERE student_id = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

student_id	course_id	semester
1	101	1
1	102	1
1	103	3

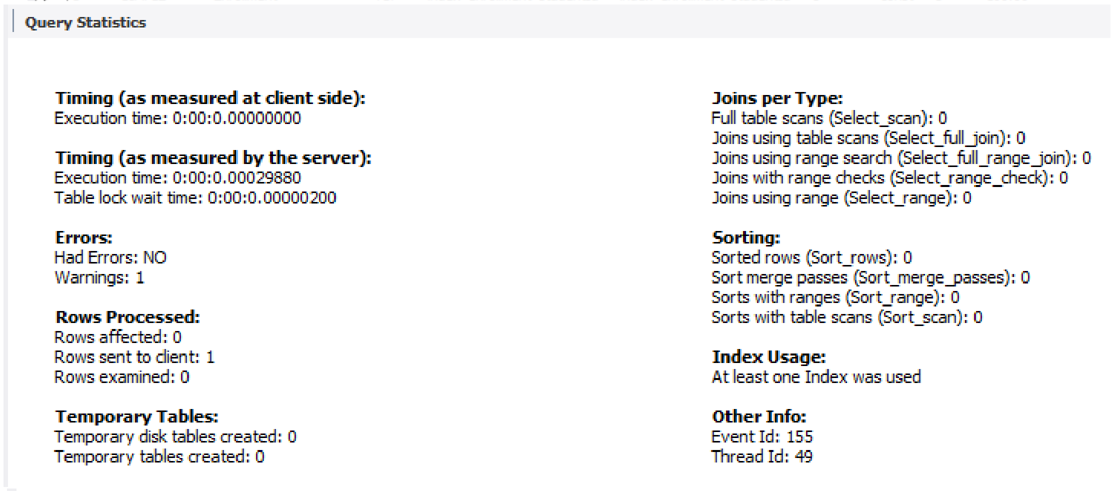
- **Use EXPLAIN to observe the query processing of the SELECT statement.**



```sql
-- EXPLAIN observe query processing of SELECT
EXPLAIN SELECT * FROM Enrollment WHERE student_id = 1;
```

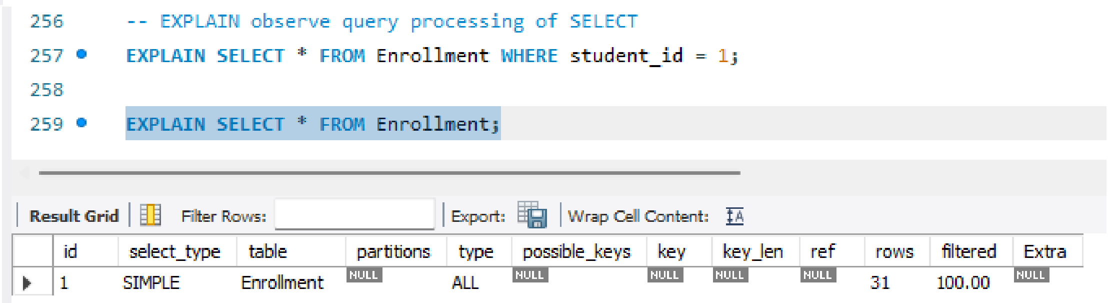
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Enrollment	NULL	ref	index enrollment studentId	index enrollment studentId	5	const	3	100.00	NULL



Query Statistics

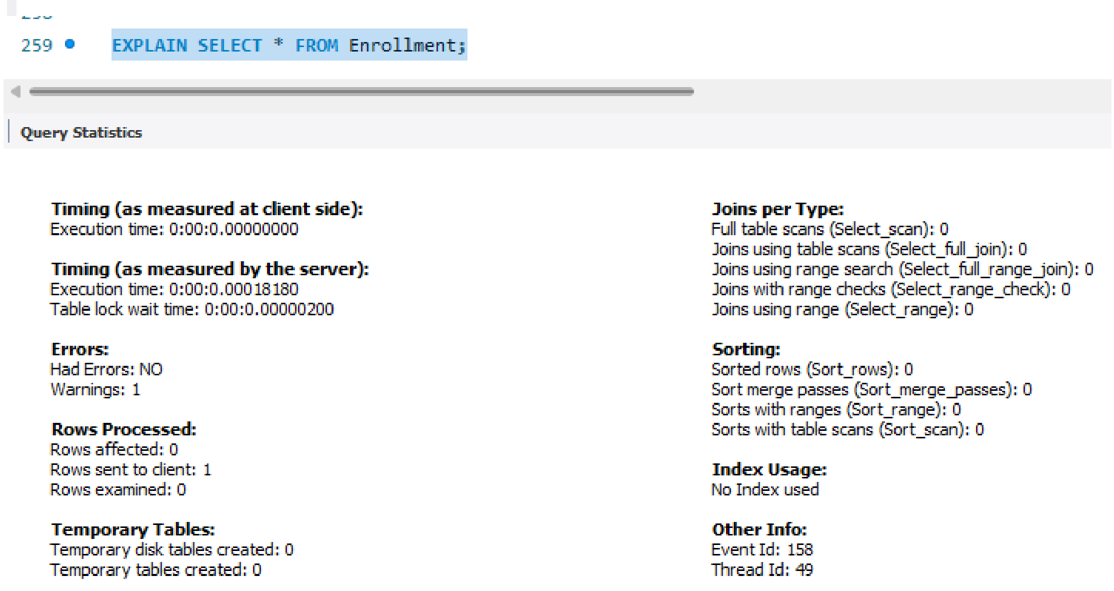
```
Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00029880
Table lock wait time: 0:00:0.00000200

Errors:
Had Errors: NO
Warnings: 1

Rows Processed:
Rows affected: 0
Rows sent to client: 1
Rows examined: 0

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

Other Info:
Event Id: 155
Thread Id: 49
```



```sql
-- EXPLAIN observe query processing of SELECT
EXPLAIN SELECT * FROM Enrollment WHERE student_id = 1;

EXPLAIN SELECT * FROM Enrollment;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Enrollment	NULL	ALL	NULL	NULL	NULL	NULL	31	100.00	NULL



```sql
EXPLAIN SELECT * FROM Enrollment;
```

Query Statistics

```
Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00018180
Table lock wait time: 0:00:0.00000200

Errors:
Had Errors: NO
Warnings: 1

Rows Processed:
Rows affected: 0
Rows sent to client: 1
Rows examined: 0

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
No Index used

Other Info:
Event Id: 158
Thread Id: 49
```