

## **DDL commands**

```
-- Create the School table
CREATE TABLE School (
    schoolId INT PRIMARY KEY,
    schoolName VARCHAR(100),
    schoolCity VARCHAR(100)
);

-- Create the Boards table
CREATE TABLE Boards (
    boardId INT PRIMARY KEY,
    boardName VARCHAR(100)
);

-- Create the TwelfthSpecializations table
CREATE TABLE TwelfthSpecializations (
    specId INT PRIMARY KEY,
    specName VARCHAR(100)
);

-- Create the UndergradFields table
CREATE TABLE UndergradFields (
    fieldId INT PRIMARY KEY,
    fieldName VARCHAR(100)
);

-- Create the MBASpecializations table
CREATE TABLE MBASpecializations (
    specId INT PRIMARY KEY,
    specName VARCHAR(100)
);

-- Create the Students table with foreign key references
CREATE TABLE Students (
    studentId INT PRIMARY KEY,
    gender VARCHAR(10),
    schoolId INT,
    tenthBoardId INT,
    twelfthBoardId INT,
    twelfthSpecId INT,
    ungradFieldId INT,
    MBASpecId INT,
    FOREIGN KEY (schoolId) REFERENCES School(schoolId)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    FOREIGN KEY (tenthBoardId) REFERENCES Boards(boardId)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    FOREIGN KEY (twelfthBoardId) REFERENCES Boards(boardId)
```

```

        ON DELETE SET NULL
        ON UPDATE CASCADE,
FOREIGN KEY (twelfthSpecId) REFERENCES TwelfthSpecializations(specId)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
FOREIGN KEY (ungradFieldId) REFERENCES UndergradFields(fieldId)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
FOREIGN KEY (MBASpecId) REFERENCES MBASpecializations(specId)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);

-- Create the EmpStats table
CREATE TABLE EmpStats (
    studentId INT PRIMARY KEY,
    workExperience INT,
    employabilityTest INT,
    FOREIGN KEY (studentId) REFERENCES Students(studentId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

-- Create the Grades table
CREATE TABLE Grades (
    studentId INT PRIMARY KEY,
    gender VARCHAR(10),
    tenthPercent INT,
    twelfthPercent INT,
    undergradPercent INT,
    MBAPercent INT,
    FOREIGN KEY (studentId) REFERENCES Students(studentId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

-- Create the Placements table
CREATE TABLE Placements (
    studentId INT PRIMARY KEY,
    status VARCHAR(100),
    salary INT,
    FOREIGN KEY (studentId) REFERENCES Students(studentId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

```

## Advanced Query 1:

### Function

This query finds the average salary, minimum salary, maximum salary, and most common MBA specialization for a student, given an undergraduate field, specifically Sci&Tech in this example.

### Code

```
SELECT
    uf.fieldName AS SelectedDegree,
    AVG(p.salary) AS AvgSalary,
    MIN(p.salary) AS MinSalary,
    MAX(p.salary) AS MaxSalary,
    (
        SELECT ms.specName
        FROM MBASpecializations AS ms
        WHERE ms.specId IN (
            SELECT s.MBASpecId
            FROM Students s
            WHERE s.undergradFieldId = uf.fieldId
        )
        GROUP BY ms.specName
        ORDER BY COUNT(*) DESC
        LIMIT 1
    ) AS ModeSpecialization

FROM Students AS s LEFT JOIN UndergradFields uf ON s.undergradFieldId =
uf.fieldId
LEFT JOIN Placements AS p ON s.studentId = p.studentId
WHERE uf.fieldName = 'Sci&Tech'
GROUP BY uf.fieldName, ModeSpecialization;
```

### Output

SelectedDegree	AvgSalary	MinSalary	MaxSalary	ModeSpecialization
Sci&Tech	554880.4348	200000	850000	Mkt&Fin

\*only one row, because there is only one set of values to be returned.

### Indexing

This is what our initial EXPLAIN ANALYZE looked like.

```

-> Table scan on <temporary> (actual time=93.758..93.758 rows=1 loops=1)
-> Aggregate using temporary table (actual time=93.756..93.756 rows=1 loops=1)
-> Nested loop left join (cost=229.66 rows=508) (actual time=1.749..0.128 rows=224 loops=1)
-> Nested loop inner join (cost=51.69 rows=508) (actual time=0.305..0.454 rows=224 loops=1)
-> Filter: ((uf.fieldName = 'Sci&Tech') and (uf.fieldId is not null)) (cost=0.45 rows=1) (actual time=0.135..0.141 rows=1 loops=1)
-> Table scan on uf (cost=0.45 rows=2) (actual time=0.107..0.113 rows=2 loops=1)
-> Covering index lookup on s using ungradFieldId (undergradFieldId=uf.fieldId) (cost=51.24 rows=508) (actual time=0.169..0.289 rows=224 loops=1)
-> Index lookup on p using pl_student_id_idx (studentId=s.studentId) (cost=0.25 rows=1) (actual time=0.037..0.038 rows=1 loops=224)
-> Select #2 (subquery in projection; dependent)
-> Limit: 1 row(s) (actual time=0.363..0.363 rows=1 loops=225)
-> Sort: count(0) DESC, limit input to 1 row(s) per chunk (actual time=0.362..0.362 rows=1 loops=225)
-> Table scan on <temporary> (actual time=0.359..0.360 rows=2 loops=225)
-> Aggregate using temporary table (actual time=0.359..0.359 rows=2 loops=225)
-> Nested loop inner join (cost=102.35 rows=1017) (actual time=0.352..0.355 rows=2 loops=225)
-> Table scan on ms (cost=0.45 rows=2) (actual time=0.003..0.004 rows=2 loops=225)
-> Single-row index lookup on <subquery3> using <auto_distinct_key> (MBASpecId=ms.specId, undergradFieldId=uf.fieldId) (actual time=0.175..0.175 rows=1 loops=450)
-> Materialize with deduplication (cost=105.45..105.45 rows=508) (actual time=0.347..0.347 rows=2 loops=225)
-> Filter: ((s.MBASpecId is not null) and (s.undergradFieldId is not null)) (cost=54.60 rows=508) (actual time=0.101..0.290 rows=224 loops=225)
-> Index lookup on s using ungradFieldId (undergradFieldId=uf.fieldId) (cost=54.60 rows=508) (actual time=0.101..0.260 rows=224 loops=225)

```

We then created an index for MBASpecId in the Students table because it was being used within a boolean operator and count aggregate function to determine the most popular speciality given an undergraduate field, which could increase overall time.

```
CREATE INDEX mba_id_idx on Students(MBASpecId);
```

This is what the EXPLAIN ANALYZE outputted after this index was created.

```

-> Table scan on <temporary> (actual time=81.928..81.928 rows=1 loops=1)
-> Aggregate using temporary table (actual time=81.927..81.927 rows=1 loops=1)
-> Nested loop left join (cost=229.66 rows=508) (actual time=0.059..1.162 rows=224 loops=1)
-> Nested loop inner join (cost=51.69 rows=508) (actual time=0.049..0.156 rows=224 loops=1)
-> Filter: ((uf.fieldName = 'Sci&Tech') and (uf.fieldId is not null)) (cost=0.45 rows=1) (actual time=0.019..0.023 rows=1 loops=1)
-> Table scan on uf (cost=0.45 rows=2) (actual time=0.013..0.017 rows=2 loops=1)
-> Covering index lookup on s using ungradFieldId (undergradFieldId=uf.fieldId) (cost=51.24 rows=508) (actual time=0.028..0.110 rows=224 loops=1)
-> Index lookup on p using pl_student_id_idx (studentId=s.studentId) (cost=0.25 rows=1) (actual time=0.003..0.004 rows=1 loops=224)
-> Select #2 (subquery in projection; dependent)
-> Limit: 1 row(s) (actual time=0.348..0.349 rows=1 loops=225)
-> Sort: count(0) DESC, limit input to 1 row(s) per chunk (actual time=0.348..0.348 rows=1 loops=225)
-> Table scan on <temporary> (actual time=0.345..0.345 rows=2 loops=225)
-> Aggregate using temporary table (actual time=0.345..0.345 rows=2 loops=225)
-> Nested loop inner join (cost=102.35 rows=1017) (actual time=0.339..0.341 rows=2 loops=225)
-> Table scan on ms (cost=0.45 rows=2) (actual time=0.002..0.003 rows=2 loops=225)
-> Single-row index lookup on <subquery3> using <auto_distinct_key> (MBASpecId=ms.specId, undergradFieldId=uf.fieldId) (actual time=0.168..0.168 rows=1 loops=450)
-> Materialize with deduplication (cost=105.45..105.45 rows=508) (actual time=0.335..0.335 rows=2 loops=225)
-> Filter: ((s.MBASpecId is not null) and (s.undergradFieldId is not null)) (cost=54.60 rows=508) (actual time=0.096..0.277 rows=224 loops=225)
-> Index lookup on s using ungradFieldId (undergradFieldId=uf.fieldId) (cost=54.60 rows=508) (actual time=0.095..0.249 rows=224 loops=225)

```

We then created an index for MBASpecId in the Students table because it was being used within the subquery, possibly increasing the overall time of the query.

```
CREATE INDEX undergrad_id_idx on Students(undergradFieldId);
```

This is what the EXPLAIN ANALYZE outputted after this index was created.

```

-> Table scan on <temporary> (actual time=81.498..81.498 rows=1 loops=1)
-> Aggregate using temporary table (actual time=81.497..81.497 rows=1 loops=1)
-> Nested loop left join (cost=229.66 rows=508) (actual time=0.065..1.141 rows=224 loops=1)
-> Nested loop inner join (cost=51.69 rows=508) (actual time=0.053..0.163 rows=224 loops=1)
-> Filter: ((uf.fieldName = 'Sci&Tech') and (uf.fieldId is not null)) (cost=0.45 rows=1) (actual time=0.022..0.026 rows=1 loops=1)
-> Table scan on uf (cost=0.45 rows=2) (actual time=0.015..0.019 rows=2 loops=1)
-> Covering index lookup on s using undergrad_id_idx (undergradFieldId=uf.fieldId) (cost=51.24 rows=508) (actual time=0.030..0.114 rows=224 loops=1)
-> Index lookup on p using pl_student_id_idx (studentId=s.studentId) (cost=0.25 rows=1) (actual time=0.003..0.004 rows=1 loops=224)
-> Select #2 (subquery in projection; dependent)
-> Limit: 1 row(s) (actual time=0.346..0.346 rows=1 loops=225)
-> Sort: count(0) DESC, limit input to 1 row(s) per chunk (actual time=0.346..0.346 rows=1 loops=225)
-> Table scan on <temporary> (actual time=0.344..0.344 rows=2 loops=225)
-> Aggregate using temporary table (actual time=0.343..0.343 rows=2 loops=225)
-> Nested loop inner join (cost=102.35 rows=1017) (actual time=0.337..0.340 rows=2 loops=225)
-> Table scan on ms (cost=0.45 rows=2) (actual time=0.002..0.003 rows=2 loops=225)
-> Single-row index lookup on <subquery3> using <auto_distinct_key> (MBASpecId=ms.specId, undergradFieldId=uf.fieldId) (actual time=0.168..0.168 rows=1 loops=450)
-> Materialize with deduplication (cost=105.45..105.45 rows=508) (actual time=0.333..0.333 rows=2 loops=225)
-> Filter: ((s.MBASpecId is not null) and (s.undergradFieldId is not null)) (cost=54.60 rows=508) (actual time=0.095..0.276 rows=224 loops=225)
-> Index lookup on s using undergrad_id_idx (undergradFieldId=uf.fieldId) (cost=54.60 rows=508) (actual time=0.095..0.246 rows=224 loops=225)

```

We then created an index for StudentId in the Students table because it was being used within the main query in the FROM clause to join the Student table with the UndergradFields and Placements table.

```
CREATE INDEX stu_id_idx on Students(studentId);
```

This is what the EXPLAIN\_ANALYZE outputted after this index was created.

```
-> Table scan on <temporary> (actual time=82.286..82.286 rows=1 loops=1)
-> Aggregate using temporary table (actual time=82.285..82.285 rows=1 loops=1)
-> Nested loop left join (cost=229.66 rows=508) (actual time=0.097..1.326 rows=224 loops=1)
-> Nested loop inner join (cost=51.69 rows=508) (actual time=0.080..0.230 rows=224 loops=1)
-> Filter: ((uf.fieldName = 'Sci&Tech') and (uf.fieldId is not null)) (cost=0.45 rows=1) (actual time=0.042..0.047 rows=1 loops=1)
-> Table scan on uf (cost=0.45 rows=2) (actual time=0.033..0.038 rows=2 loops=1)
-> Table scan on s using undergrad_id_idx (undergradFieldId=uf.fieldId) (cost=51.24 rows=508) (actual time=0.036..0.160 rows=224 loops=1)
-> Index lookup on p using pl_student_id_idx (studentId=s.studentId) (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=224)
-> Select #2 (subquery in projection; dependent)
-> Limit: 1 row(s) (actual time=0.348..0.348 rows=1 loops=225)
-> Sort: count(0) DESC, limit input to 1 row(s) per chunk (actual time=0.348..0.348 rows=1 loops=225)
-> Table scan on <temporary> (actual time=0.345..0.345 rows=2 loops=225)
-> Aggregate using temporary table (actual time=0.344..0.344 rows=2 loops=225)
-> Nested loop inner join (cost=102.35 rows=1017) (actual time=0.339..0.341 rows=2 loops=225)
-> Table scan on ms (cost=0.45 rows=2) (actual time=0.002..0.003 rows=2 loops=225)
-> Single-row index lookup on <subquery3> using <auto distinct key> (MBASpecId=ms.specId, undergradFieldId=uf.fieldId) (actual time=0.168..0.169 rows=1 loops=450)
-> Materialize with deduplication (cost=185.45..185.45 rows=508) (actual time=0.335..0.335 rows=2 loops=225)
-> Filter: ((s.MBASpecId is not null) and (s.undergradFieldId is not null)) (cost=54.60 rows=508) (actual time=0.096..0.278 rows=224 loops=225)
-> Index lookup on s using undergrad_id_idx (undergradFieldId=uf.fieldId) (cost=54.60 rows=508) (actual time=0.095..0.240 rows=224 loops=225)
```

We noticed that adding the indexes for these attributes did not improve the cost or time for our query greatly. This is because most of the attributes in the students table are foreign keys that reference other tables and foreign keys are already indexed. Because of this, we decided to not to add any indexes for this query.

## Advanced Query 2:

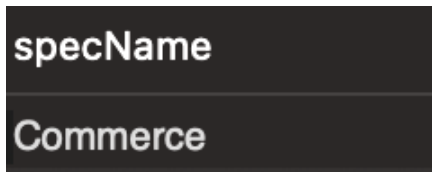
### Function

This query finds the most popular twelfth grade specialization among students with a certain salary, specifically above 500000 in this example.

### Code

```
SELECT specName
FROM TwelfthSpecializations
WHERE specId IN
(
SELECT ROUND(AVG(t.specId), 0)
FROM Students s LEFT JOIN TwelfthSpecializations t ON s.twelfthSpecId =
t.specId
WHERE studentId IN (SELECT p.studentId FROM Placements p WHERE p.salary >=
500000)
);
```

### Output



specName
Commerce

\*only one row because only one specialization returned

### Indexing

This is what our initial EXPLAIN ANALYZE looked like.

```
-- Filter: <in_optimizer>(TwelfthSpecializations.specId,TwelfthSpecializations.specId in (select #2)) (cost=0.45 rows=2) (actual time=1.724..1.737 rows=1 loops=1)
--> Table scan on TwelfthSpecializations (cost=0.45 rows=2) (actual time=0.026..0.029 rows=3 loops=1)
--> Select #2 (subquery in condition; run only once)
--> Filter: ((TwelfthSpecializations.specId = '<materialized_subquery>'.ROUND(AVG(t.specId), 0))) (cost=338.78..338.78 rows=1) (actual time=0.424..0.424 rows=0 loops=4)
--> Limit: 1 row(s) (cost=338.68..338.68 rows=1) (actual time=0.423..0.423 rows=0 loops=4)
--> Index lookup on <materialized_subquery> using <auto_distinct_key> (ROUND(AVG(t.specId), 0)=TwelfthSpecializations.specId) (actual time=0.423..0.423 rows=0 loops=4)
--> Materialize with deduplication (cost=338.68..338.68 rows=1) (actual time=1.686..1.686 rows=1 loops=1)
--> Aggregate: avg(t.specId) (cost=338.58 rows=1) (actual time=1.675..1.675 rows=1 loops=1)
--> Nested loop left join (cost=304.78 rows=338) (actual time=0.819..1.635 rows=310 loops=1)
--> Filter: ('<subquery3>'.studentId is not null) (cost=136.64..67.70 rows=338) (actual time=0.801..0.883 rows=310 loops=1)
--> Table scan on <subquery3> (cost=136.87..143.57 rows=339) (actual time=0.801..0.854 rows=310 loops=1)
--> Materialize with deduplication (cost=136.85..136.85 rows=339) (actual time=0.800..0.800 rows=310 loops=1)
--> Filter: (p.studentId is not null) (cost=102.95 rows=339) (actual time=0.060..0.733 rows=310 loops=1)
--> Filter: (p.salary >= 500000) (cost=102.95 rows=339) (actual time=0.059..0.689 rows=310 loops=1)
--> Table scan on p (cost=102.95 rows=1017) (actual time=0.010..0.589 rows=1017 loops=1)
--> Single-row index lookup on s using PRIMARY (studentId='<subquery3>'.studentId) (cost=84.84 rows=1) (actual time=0.001..0.001 rows=1 loops=310)
--> Single-row covering index lookup on t using PRIMARY (specId=s.twelfthSpecId) (cost=84.84 rows=1) (actual time=0.001..0.001 rows=1 loops=310)
```

We then created an index for studentId in the Students table because it was being used to join in the query and could have been increasing the cost.

```
CREATE INDEX student_id_idx ON Students(studentId);
```



This is what the EXPLAIN ANALYZE outputted after this index was created.

```
=> Filter: (<in_optimizer>(TwelfthSpecializations.specId,TwelfthSpecializations.specId in (select #2)) (cost=0.45 rows=2) (actual time=2.114..2.137 rows=1 loops=1)
-> Table scan on TwelfthSpecializations (cost=0.45 rows=2) (actual time=0.015..0.021 rows=3 loops=1)
-> Select #2 (subquery in condition; run only once)
-> Filter: ((TwelfthSpecializations.specId = '<materialized_subquery>'.ROUND(AVG(t.specId), 0))) (cost=338.78..338.78 rows=1) (actual time=0.524..0.524 rows=0 loops=4)
-> Limit: 1 row(s) (cost=338.68..338.68 rows=1) (actual time=0.523..0.523 rows=0 loops=4)
-> Index lookup on <materialized_subquery> using <auto_distinct_key> (ROUND(AVG(t.specId), 0)=TwelfthSpecializations.specId) (actual time=0.523..0.523 rows=0 loops=4)
-> Materialize with deduplication (cost=338.68..338.68 rows=1) (actual time=2.083..2.083 rows=1 loops=1)
-> Aggregate: avg(t.specId) (cost=338.58 rows=1) (actual time=2.068..2.069 rows=1 loops=1)
-> Nested loop left join (cost=304.78 rows=338) (actual time=0.879..1.994 rows=310 loops=1)
-> Nested loop inner join (cost=186.24 rows=338) (actual time=0.875..1.563 rows=310 loops=1)
-> Filter: ('<subquery3>'.studentId is not null) (cost=136.64..67.70 rows=338) (actual time=0.863..0.987 rows=310 loops=1)
-> Table scan on <subquery3> (cost=136.87..143.57 rows=339) (actual time=0.863..0.946 rows=310 loops=1)
-> Materialize with deduplication (cost=136.85..136.85 rows=339) (actual time=0.862..0.862 rows=310 loops=1)
-> Filter: (p.studentId is not null) (cost=102.95 rows=339) (actual time=0.068..0.799 rows=310 loops=1)
-> Filter: (p.salary >= 500000) (cost=102.95 rows=339) (actual time=0.068..0.759 rows=310 loops=1)
-> Table scan on p (cost=102.95 rows=1017) (actual time=0.021..0.661 rows=1017 loops=1)
-> Single-row index lookup on s using PRIMARY (studentId='<subquery3>'.studentId) (cost=84.84 rows=1) (actual time=0.002..0.002 rows=1 loops=310)
-> Single-row covering index lookup on t using PRIMARY (specId=s.twelfthSpecId) (cost=84.84 rows=1) (actual time=0.001..0.001 rows=1 loops=310)
```

We then created an index for salary in the Placements table because it was being filtered on in the query and could have been increasing the cost.

```
CREATE INDEX salary_idx ON Placements(salary);
```

This is what the EXPLAIN ANALYZE outputted after this index was created.

```
=> Filter: (<in_optimizer>(TwelfthSpecializations.specId,TwelfthSpecializations.specId in (select #2)) (cost=0.45 rows=2) (actual time=1.806..1.821 rows=1 loops=1)
-> Table scan on TwelfthSpecializations (cost=0.45 rows=2) (actual time=0.025..0.028 rows=3 loops=1)
-> Select #2 (subquery in condition; run only once)
-> Filter: ((TwelfthSpecializations.specId = '<materialized_subquery>'.ROUND(AVG(t.specId), 0))) (cost=310.20..310.20 rows=1) (actual time=0.444..0.444 rows=0 loops=4)
-> Limit: 1 row(s) (cost=310.10..310.10 rows=1) (actual time=0.443..0.443 rows=0 loops=4)
-> Index lookup on <materialized_subquery> using <auto_distinct_key> (ROUND(AVG(t.specId), 0)=TwelfthSpecializations.specId) (actual time=0.443..0.443 rows=0 loops=4)
-> Materialize with deduplication (cost=310.10..310.10 rows=1) (actual time=1.765..1.765 rows=1 loops=1)
-> Aggregate: avg(t.specId) (cost=310.00 rows=1) (actual time=1.752..1.752 rows=1 loops=1)
-> Nested loop left join (cost=279.00 rows=310) (actual time=0.848..1.707 rows=310 loops=1)
-> Nested loop inner join (cost=170.50 rows=310) (actual time=0.841..1.381 rows=310 loops=1)
-> Filter: ('<subquery3>'.studentId is not null) (cost=133.72..62.00 rows=310) (actual time=0.824..0.906 rows=310 loops=1)
-> Table scan on <subquery3> (cost=133.97..140.33 rows=310) (actual time=0.824..0.877 rows=310 loops=1)
-> Materialize with deduplication (cost=133.95..133.95 rows=310) (actual time=0.823..0.823 rows=310 loops=1)
-> Filter: (p.studentId is not null) (cost=102.95 rows=310) (actual time=0.068..0.757 rows=310 loops=1)
-> Filter: (p.salary >= 500000) (cost=102.95 rows=310) (actual time=0.068..0.715 rows=310 loops=1)
-> Table scan on p (cost=102.95 rows=1017) (actual time=0.019..0.575 rows=1017 loops=1)
-> Single-row index lookup on s using PRIMARY (studentId='<subquery3>'.studentId) (cost=77.60 rows=1) (actual time=0.001..0.001 rows=1 loops=310)
-> Single-row covering index lookup on t using PRIMARY (specId=s.twelfthSpecId) (cost=77.60 rows=1) (actual time=0.001..0.001 rows=1 loops=310)
```

We then created an index for studentId in the Placements table because it was also being used in the join in the query and could have been increasing the cost.

```
CREATE INDEX pl_student_id_idx ON Placements(studentId);
```

This is what the EXPLAIN ANALYZE outputted after this index was created.

```
=> Filter: (<in_optimizer>(TwelfthSpecializations.specId,TwelfthSpecializations.specId in (select #2)) (cost=0.45 rows=2) (actual time=1.812..1.834 rows=1 loops=1)
-> Table scan on TwelfthSpecializations (cost=0.45 rows=2) (actual time=0.019..0.028 rows=3 loops=1)
-> Select #2 (subquery in condition; run only once)
-> Filter: ((TwelfthSpecializations.specId = '<materialized_subquery>'.ROUND(AVG(t.specId), 0))) (cost=310.20..310.20 rows=1) (actual time=0.448..0.448 rows=0 loops=4)
-> Limit: 1 row(s) (cost=310.10..310.10 rows=1) (actual time=0.447..0.447 rows=0 loops=4)
-> Index lookup on <materialized_subquery> using <auto_distinct_key> (ROUND(AVG(t.specId), 0)=TwelfthSpecializations.specId) (actual time=0.447..0.447 rows=0 loops=4)
-> Materialize with deduplication (cost=310.10..310.10 rows=1) (actual time=1.781..1.781 rows=1 loops=1)
-> Aggregate: avg(t.specId) (cost=310.00 rows=1) (actual time=1.768..1.768 rows=1 loops=1)
-> Nested loop left join (cost=279.00 rows=310) (actual time=0.804..1.722 rows=310 loops=1)
-> Nested loop inner join (cost=170.50 rows=310) (actual time=0.784..1.363 rows=310 loops=1)
-> Filter: ('<subquery3>'.studentId is not null) (cost=133.72..62.00 rows=310) (actual time=0.771..0.857 rows=310 loops=1)
-> Table scan on <subquery3> (cost=133.97..140.33 rows=310) (actual time=0.771..0.825 rows=310 loops=1)
-> Materialize with deduplication (cost=133.95..133.95 rows=310) (actual time=0.770..0.770 rows=310 loops=1)
-> Filter: (p.studentId is not null) (cost=102.95 rows=310) (actual time=0.057..0.711 rows=310 loops=1)
-> Filter: (p.salary >= 500000) (cost=102.95 rows=310) (actual time=0.056..0.670 rows=310 loops=1)
-> Table scan on p (cost=102.95 rows=1017) (actual time=0.008..0.572 rows=1017 loops=1)
-> Single-row index lookup on s using PRIMARY (studentId='<subquery3>'.studentId) (cost=77.60 rows=1) (actual time=0.001..0.001 rows=1 loops=310)
-> Single-row covering index lookup on t using PRIMARY (specId=s.twelfthSpecId) (cost=77.60 rows=1) (actual time=0.001..0.001 rows=1 loops=310)
```

We decided to not add any indexes, i.e., go forward with the default index. We chose this plan because as can be seen in the EXPLAIN ANALYZE outputs above, adding indexes did not significantly change the cost of our query. This happened for two main reasons. One, the query is complex. In queries with complex conditions or joins, the database optimizer may decide that a full table scan is more efficient than using an index because it avoids the overhead of index lookups. Second, some of the attributes we indexed are already primary or foreign keys in the schema and in MySQL this means they are already keys. So, adding them as keys did not change anything in the performance. We still chose to try indexing these attributes because they were the only viable options. In brief, we are choosing to go forward with the default indexing because indexing will not optimize our cost due to the complexity of the query and the nature of our database's relational schema (i.e., the predefined primary and foreign keys).