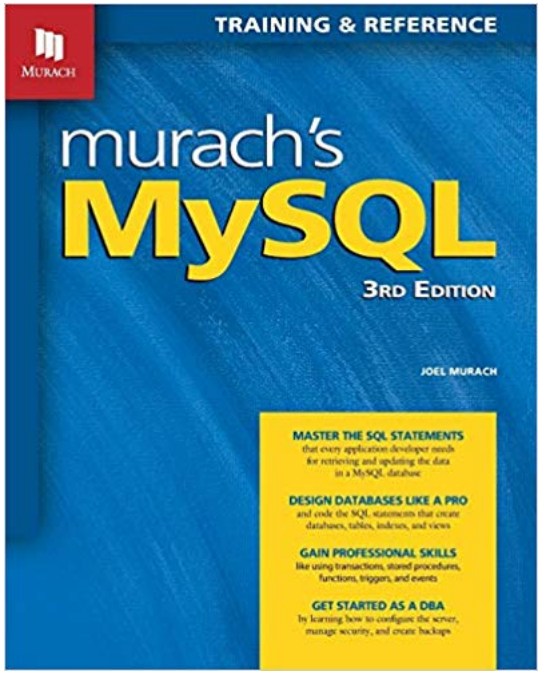
# Subqueries or Nested queries

Topic 3
Lesson 5 – Building up a query

# Chapter 7 Murach's MySQL



# Getting the results, you want

Sometimes, it takes many transformations of the data in a database to get the results you want.

Nesting queries allows you to develop each of the transformations separately.

This allows you to take a divide and conquer approach to creating your queries.

This approach is a more modular approach, leading to a simpler approach for both building and testing your queries.

# Example of a nested query

Suppose you want to find all the students that have earned above the average number of earned credits.

| sid | Name | School | Credits_Earned | Credits_Req | Yr_grad |
|---|---|---|---|---|---|
| 1 | Smith | Khoury | 32 | 120 | 2019 |
| 2 | Shah | D'Amore McKim | 64 | 128 | 2019 |
| 3 | Li | Khoury | 50 | 120 | 2020 |

Step 1: Find the average number of earned credits for students

Step 2: Find all students who have earned more than the average calculated in step 1.

# Solution: nested query

Step 1: Find the average number of credits

```
SELECT AVG(credits_earned) FROM student;
```

Step 2: Find all students who have earned more than the average calculated in step 1.

```
SELECT sid, name, credits_earned FROM student
        WHERE credits_earned >
      (SELECT AVG(credits_earned) FROM student);
```

| sid | Name | School | Credits_Earned | Credits_Req | Yr_grad |
|---|---|---|---|---|---|
| 1 | Smith | Khoury | 32 | 120 | 2019 |
| 2 | Shah | D'Amore McKim | 64 | 128 | 2019 |
| 3 | Li | Khoury | 50 | 120 | 2020 |

# A subquery can appear …

1. In a WHERE clause as a search condition
2. In a HAVING clause as a search condition
3. In the FROM clause as a table specification
4. In the SELECT clause as a column specification

- These are the only clauses where a subquery can appear.
- A subquery can return a single value, a list of values or a table of values.
- Subqueries are surrounded by parentheses.
- A subquery may contain other subqueries.
- Our previous example had a subquery in the WHERE clause.

# Subquery Example in the WHERE clause

A subquery can be used to collect the values that you want to match for a criteria from a table.

Suppose you want to find all the student ids who are majoring in CS or DS.

student

| sid | Name | School | Credits_Earned | Credits_Req | Yr_grad |
|---|---|---|---|---|---|
| 1 | Smith | Khoury | 32 | 120 | 2019 |
| 2 | Shah | D'Amore McKim | 64 | 128 | 2019 |
| 3 | Li | Khoury | 50 | 120 | 2020 |

Available_major

| MID | MAJOR |
|---|---|
| 1 | CS |
| 2 | DS |
| 3 | Accounting |

student_major

| SID | MID |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

# Solution: A nested query with IN

EXAMPLE: SELECT sid FROM student_major
WHERE mid IN
(SELECT mid from available_major WHERE
major = ‘CS’ or major = ‘DS’ )

available_major

| MID | MAJOR |
|---|---|
| 1 | CS |
| 2 | DS |
| 3 | Accounting |

student_major

| SID | MID |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

| SID |
|---|
| 1 |
| 2 |
| 3 |

What about solving this with a JOIN?

# Solution: with a JOIN

EXAMPLE: SELECT sid FROM student_major
JOIN available_major USING (mid)
WHERE major = 'CS' OR major = 'DS' ;

available_major

| MID | MAJOR |
|---|---|
| 1 | CS |
| 2 | DS |
| 3 | Accounting |

student_major

| sid | mid |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

result

| SID |
|---|
| 1 |
| 2 |
| 3 |

IF all result fields for a JOIN are from one table this is known as a **SEMIJOIN.** It will generate a simpler query plan than a typical join.

# ANY and ALL keywords

Many times a subquery will return more than one value, however the tuple only has one value. You can address this discrepancy by using the keywords: SOME, ANY, or ALL.

ANY - compares a value to each value in the result list from a subquery and evaluates to TRUE if the conditional is true for ANY of the values. SOME provides the same functionality.

ALL - compares a value to each value in the result list from a subquery and evaluates to TRUE if the conditional is true for ALL of the values

# Example with ALL keyword in WHERE clause

Write a query to find student ids that have credits_earned greater than all values for credits_earned for all CS majors.

student

| sid | Name | School | Credits_Earned | Credits_Req | Yr_grad |
|---|---|---|---|---|---|
| 1 | Smith | Khoury | 32 | 120 | 2019 |
| 2 | Shah | D'Amore McKim | 64 | 128 | 2019 |
| 3 | Li | Khoury | 50 | 120 | 2020 |

Available_major

| MID | MAJOR |
|---|---|
| 1 | CS |
| 2 | DS |
| 3 | Accounting |

student_major

| sid | mid |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

# Solution: ALL keyword

```
SELECT sid from student WHERE credits_earned > ALL
    (SELECT credits_earned FROM student WHERE sid IN
        (SELECT sm.sid FROM available_major  a JOIN
            student_major sm USING (mid) WHERE major = 'CS' );
```

| sid | Name | School | Credits_Earned | Credits_Req | Yr_grad |
|---|---|---|---|---|---|
| 1 | Smith | Khoury | 32 | 120 | 2019 |
| 2 | Shah | D'Amore McKim | 64 | 128 | 2019 |
| 3 | Li | Khoury | 50 | 120 | 2020 |

student

Available_major

| MID | MAJOR |
|---|---|
| 1 | CS |
| 2 | DS |
| 3 | Accounting |

**solution**

| id |
|---|

student_major

| sid | mid |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

# Example: ANY keyword

If we wanted to find the students that had credits_earned greater than any of the CS majors then use the ANY keyword.

```
SELECT sid from student
        WHERE credits_earned > ANY
                (SELECT credits_earned FROM student
                WHERE sid IN
                 (SELECT sm.sid FROM available_major a
                        JOIN  student_major sm
                    USING (mid) WHERE major = 'CS' );
```

**solution**

| id |
|---|
| 3 |

# Uncorrelated and correlated queries

In all examples, our subquery is computed in one stage and our outer query is computed in a following stage. This is known as an uncorrelated query. The inner query is computed once, and its result values are used by the outer query which is also computed once.

Sometimes you may want the inner and outer queries to be computed in the same phase of the query. This is known as a correlated query.

With a correlated query the inner query has access to the field values in the outer query. For each tuple in the outer query the inner query is computed. A correlated query is comparable to executing a loop in a host language.

# Correlated query example

Find students who have earned credits_earned greater than the average credit_earned for each student's respective major.

| sid | Name | School | Credits_Earned | Credits_Req | Yr_grad |
|---|---|---|---|---|---|
| 1 | Smith | Khoury | 32 | 120 | 2019 |
| 2 | Shah | D'Amore McKim | 64 | 128 | 2019 |
| 3 | Li | Khoury | 50 | 120 | 2020 |

| sid | mid |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

| mid | AVG(credits_earned) |
|---|---|
| 1 | 48 |
| 2 | 50 |
| 3 | 32 |

# Correlated query solution

Find students and their major who have earned credits_earned greater than the average credit_earned for each student's respective major.

```
SELECT s.sid, sm.mid FROM student AS s JOIN
                student_major AS sm  USING (sid)
                 WHERE credits_earned >
                        (SELECT AVG(credits_earned)
                                FROM student s2 JOIN
                                student_major AS sm2
                                ON s2.id  = sm2.id
                                WHERE sm.mid = sm2.mid );
```

# EXISTS keyword

Sometimes you are not interested in the actual values returned from a subquery, you are interested if there are any rows in the table that satisfies the condition of the subquery. If so, use the EXISTS keyword.

Syntax:

```
WHERE [NOT] EXISTS (subquery)
```

If the subquery returns a result, the conditional result is TRUE, if the subquery does not return a result then the result of the WHERE clause is FALSE.

WHERE EXISTS tests if one or more rows are returned by the subquery, if results, returns TRUE if not FALSE.

WHERE NOT EXISTS tests that no rows are returned by the subquery, if no result then returns TRUE if not FALSE .

# Example EXISTS keyword

Write a query to find student ids that have not declared a major

student

| sid | Name | School | Credits_Earned | Credits_Req | Yr_grad |
|---|---|---|---|---|---|
| 1 | Smith | Khoury | 32 | 120 | 2019 |
| 2 | Shah | D'Amore McKim | 64 | 128 | 2019 |
| 3 | Li | Khoury | 50 | 120 | 2020 |
| 4 | Frosh | COS | 8 | 128 | 2025 |

Available_major

| mid | major |
|---|---|
| 1 | CS |
| 2 | DS |
| 3 | Accounting |

student_major

| sid | mid |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

# Solution: NOT EXISTS

```
SELECT sid FROM student AS s WHERE NOT EXISTS
 (SELECT sid from student_major AS sm
       WHERE sm.sid = s.sid);
```

| sid | Name | School | Credits_Earned | Credits_Req | Yr_grad |
|---|---|---|---|---|---|
| 1 | Smith | Khoury | 32 | 120 | 2019 |
| 2 | Shah | D'Amore McKim | 64 | 128 | 2019 |
| 3 | Li | Khoury | 50 | 120 | 2020 |
| 4 | Frosh | COS | 8 | 128 | 2025 |

| sid | mid |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

| sid |
|---|
| 4 |

# Exercise: Solve this with a JOIN

Write a query to find student ids that have not declared a major

student

| sid | Name | School | Credits_Earned | Credits_Req | Yr_grad |
|---|---|---|---|---|---|
| 1 | Smith | Khoury | 32 | 120 | 2019 |
| 2 | Shah | D'Amore McKim | 64 | 128 | 2019 |
| 3 | Li | Khoury | 50 | 120 | 2020 |
| 4 | Frosh | COS | 8 | 128 | 2025 |

Available_major

| mid | MAJOR |
|---|---|
| 1 | CS |
| 2 | DS |
| 3 | Accounting |

student_major

| sid | mid |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

# Solution: LEFT OUTER JOIN

```
SELECT s.sid, sm.sid FROM student s
        LEFT JOIN student_major sm
                ON s.sid = sm.sid WHERE sm.sid IS NULL;
```

| sid | Name | School | Credits_Earned | Credits_Req | Yr_grad |
|---|---|---|---|---|---|
| 1 | Smith | Khoury | 32 | 120 | 2019 |
| 2 | Shah | D'Amore McKim | 64 | 128 | 2019 |
| 3 | Li | Khoury | 50 | 120 | 2020 |
| 4 | Frosh | COS | 8 | 128 | 2025 |

| sid | mid |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

| s.sid | sm.sid |
|---|---|
| 4 | NULL |

# A Subquery in the SELECT clause

Many times you want to report an aggregated value for each tuple in a table. Use a subquery in the SELECT clause to create this type of result.

If a subquery is in the SELECT clause it must return a single value.

Subqueries in the SELECT clause are typically correlated.

EXAMPLE: Find the number of majors for each student, report the student id, student name and number of majors for each student.

# EXAMPLE: Subquery in the SELECT clause

EXAMPLE: Find the number of majors for each student, report the student id, student name and number of majors for each student.

Use the student_major table to find the COUNT of majors for each student

SOLUTION:

```
SELECT sid, name, ( SELECT COUNT(*) FROM
  student_major AS sm WHERE sid = s.sid )
      AS num_majors  FROM student AS s;
```

# A Subquery in the FROM clause

An inline view is a SELECT statement in the FROM clause. An in-line view is commonly used to simplify complex queries by removing join operations and condensing several separate queries into a single query.

All subqueries in the FROM clause must be named or given an alias. You should also alias (provide a name) for any field that is a computed

This approach allows queries to evolve into a solution, making it easier to test and validate the subqueries.

# EXAMPLE: A Subquery in the FROM clause

Find the majors for each student, report the student id, student name and major names for each student.

```
SELECT s.sid, name,  major FROM student AS s
      JOIN
      ( SELECT sid, am.mid, major
            FROM student_major AS sm
           JOIN available_majors AS am
               ON sm.mid = am.mid ) AS mm
      ON s.sid = mm.sid;
```

# Result: A Subquery in the FROM clause

```
SELECT s.sid, name,  major FROM student AS s
      JOIN
      ( SELECT sid, am,mid, major
            FROM student_major AS sm
           JOIN available_majors AS am
               ON sm.mid = am.mid ) AS mm ON s.sid = mm.sid;
```

| sid | Name | major |
|---|---|---|
| 1 | Smith | CS |
| 1 | Smith | ACCOUNTING |
| 2 | Shah | CS |
| 3 | Li | DS |

This provides one row per student major. What if you wanted one row per student and the Major values combined ?

# Result: A tuple per student

```
SELECT s.sid, name,  GROUP_CONCAT(major)
      FROM student AS s
     JOIN
     ( SELECT sid, am, mid, major
           FROM student_major AS sm
          JOIN available_majors AS am
              ON sm.mid = am.mid ) AS mm
     ON s.sid = mm.sid GROUP BY id, name;
```

| sid | Name | major |
|---|---|---|
| 1 | Smith | CS,Accounting |
| 2 | Shah | CS |
| 3 | Li | DS |

What if we wanted to report on all students?

# Result: A tuple for every student

```
SELECT s.sid, name,  GROUP_CONCAT(major)
      FROM student AS s
     LEFT JOIN
     ( SELECT sid, am.mid, major
           FROM student_major AS sm
          JOIN available_majors AS am
              ON sm.mid = am.mid ) AS mm
     ON s.sid = mm.sid GROUP BY id, name;
```

| sid | Name | major |
|---|---|---|
| 1 | Smith | CS,Accounting |
| 2 | Shah | CS |
| 3 | Li | DS |
| 4 | Frosh | NULL |

# Advantages of different approaches

SUBQUERY

- Allows you to pass a value to the main query
- Queries involving many tables may be easier to comprehend
- Easier to comprehend when the query is not focused on the foreign key/primary key relationship
- May lead to faster processing time

JOIN

- Access to all fields in all tables at the same level
- Easier to comprehend when the query is focused on the foreign key/primary key relationship

# Practice work:

Let's download the subqueries exercises from canvas and work in groups to complete them.

# Advanced topic

Common Table Expressions

# Common Table Expression (CTE)

Nesting queries within the FROM clauses can lead to an SQL query that is difficult to read.

The WITH construct was introduced in ANSI SQL Standard 1999 and is supported in MySQL 8.x. It allows you to define and name a temporary result that can be used within the following SQL statement.

# Common Table Expression: Syntax

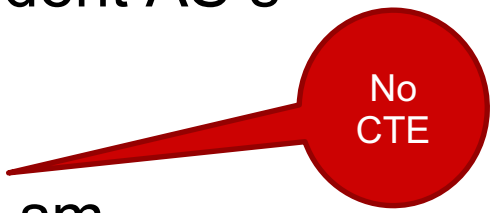
With a CTE, all your IN-line tables are defined at the beginning of your query. Your **sql_statement** can treat your CTE tables as base tables.

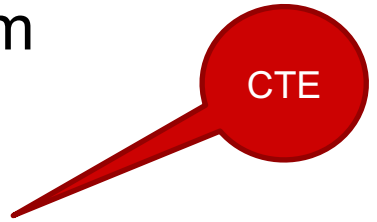
**SYNTAX:**

```
WITH [RECURSIVE] cte_name1
      AS (subquery1)
       [, cte_name2 AS (subquery2)]
        [...]
       sql_statement
```

# EXAMPLE: No CTE vs. CTE

```
SELECT id, name,  major FROM student AS s
      JOIN
      ( SELECT sid, mid, major
            FROM    student_major AS sm
          JOIN available_majors AS am
              ON sm.mid = am.mid ) AS mm
        ON s.sid = mm.sid;
```




```
WITH mm AS
      (SELECT sid, mid, major FROM student_major AS sm
          JOIN available_majors AS am
              ON sm.mid = am.mid )
SELECT id, name,  major FROM student AS s
      JOIN mm ON s.sid = mm.sid;
```

# Recursive CTEs

Recursive CTEs allows a query to process a hierarchical relationship represented in the data

A recursive CTE has a base case and a recursive case. The result of the recursive CTE is the UNION of the base case and the recursive case.

The query will continue to run until there are no tuples left to process.

# Example: Recursive CTE

Let's say you want to trace through the manager – subordinate relationship in the employee table. It is a hierarchal relationship, where the number of levels in the organization is not limited. In order to do this we need some mechanism that will continue to read the tuples until we have reached the employees who have no one reporting to them.

| employee_id | last_name | first_name | department_number | manager_id |
|---|---|---|---|---|
| 1 | Smith | Cindy | 2 | NULL |
| 2 | Jones | Elmer | 4 | 1 |
| 3 | Simonian | Ralph | 2 | 2 |
| 4 | Hernandez | Olivia | 1 | 9 |
| 5 | Aaronsen | Robert | 2 | 4 |
| 6 | Watson | Denise | 6 | 8 |
| 7 | Hardy | Thomas | 5 | 2 |
| 8 | O'Leary | Rhea | 4 | 9 |
| 9 | Locario | Paulo | 6 | 1 |

# Walk the hierarchy

| | employee_id | last_name | first_name | manager_id |
|---|---|---|---|---|
| ▶ | 1 | Smith | Cindy | NULL |
| | 2 | Jones | Elmer | 1 |
| | 3 | Simonian | Ralph | 2 |
| | 4 | Hernandez | Olivia | 9 |
| | 5 | Aaronsen | Robert | 4 |
| | 6 | Watson | Denise | 8 |
| | 7 | Hardy | Thomas | 2 |
| | 8 | O'Leary | Rhea | 9 |
| | 9 | Locario | Paulo | 1 |

No boss = Cindy Smith (1)

Base case (1)

Reports to Cindy (1):

Elmer (2) Paolo (9)

First recursive case (2)

Reports to :
Elmer (2) Paolo (9)

Ralph (3) Rhea (8)
Thomas (7) Olivia (4)

Second recursive case (3)

Reports to :
Rhea (8) Olivia (4)

Denise (6) Robert (5)

Third recursive case (4)

# Recursive RTE to walk the hierarchy

```
WITH RECURSIVE employees_cte AS
(
        -- Nonrecursive query
        SELECT employee_id,
            CONCAT(first_name, ' ', last_name) AS employee_name,
            1 AS ranking
        FROM employees
        WHERE manager_id IS NULL
    UNION ALL
        -- Recursive query
        SELECT employees.employee_id,
            CONCAT(first_name, ' ', last_name),
            ranking + 1
        FROM employees
            JOIN employees_cte
            ON employees.manager_id = employees_cte.employee_id
)
SELECT *
FROM employees_cte
ORDER BY ranking, employee_id
```

# Recursive CTE result

| | employee_id | employee_name | ranking |
|---|---|---|---|
| ▶ | 1 | Cindy Smith | 1 |
| | 2 | Elmer Jones | 2 |
| | 9 | Paulo Locario | 2 |
| | 3 | Ralph Simonian | 3 |
| | 4 | Olivia Hernandez | 3 |
| | 7 | Thomas Hardy | 3 |
| | 8 | Rhea O'Leary | 3 |
| | 5 | Robert Aaronsen | 4 |
| | 6 | Denise Watson | 4 |

# Summary

In this module you learned:

- Subqueries can be found in the SELECT, FROM, WHERE and HAVING clauses
- Correlated queries
- EXISTS
- NOT EXISTS
- ANY and ALL keywords