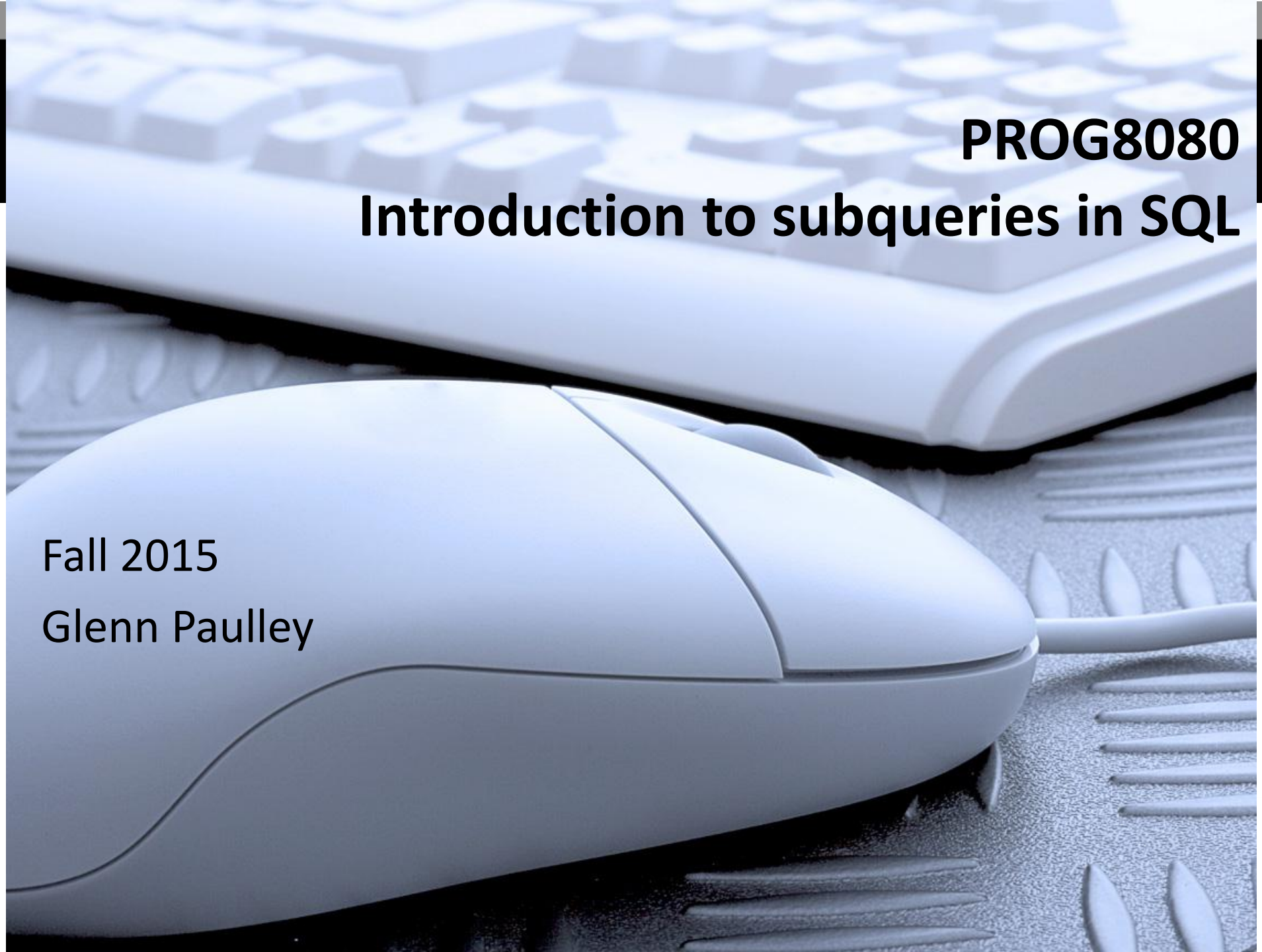

PROG8080
Introduction to subqueries in SQL

Fall 2015

Glenn Paulley

Subqueries - Preliminaries

- A subquery is a SELECT statement within the WHERE (or HAVING) clause of another SELECT statement:

```
SELECT …
        WHERE column <operator> (SELECT …)
SELECT …
        WHERE column IN (SELECT …)
SELECT …
        WHERE column NOT IN (SELECT …)
```

- These are examples of *nested queries*

Subqueries - Preliminaries

- Take a look at the StudentOffence table:

```
SELECT *
FROM StudentOffence
```

- Question: What are the student numbers of those students who have committed Type “A” offences?

```
SELECT studentNumber
FROM StudentOffence
WHERE penaltyCode = ‘A’;
```

Subqueries - Preliminaries

- Now take a look some columns from the CourseStudent table:

```
SELECT  studentNumber, finalMark
FROM CourseStudent
```

- We wish to answer the following question:
  - Question: What are the course grades for students who have committed Type “A” academic offences?
  - One possibility: use a join
  - Another possibility: use a nested query

Subqueries with comparison operators

- If the subquery returns a single value a relational operator (=, !=, <, <=, >, >=) can be used:

  SELECT …

  WHERE *column* = (SELECT …)

  SELECT …

  WHERE *column* != (SELECT …)

  SELECT …

  WHERE *column* > (SELECT …)

  Etc.

Subqueries with comparison operators

- Question: What are the course grades for students who have committed Type “A” academic offences?

```
SELECT *
FROM CourseStudent
WHERE studentNumber = ( SELECT studentNumber
                        FROM StudentOffence
                        WHERE penaltyCode = 'A');
```

- Semantics:
  - The search condition (WHERE clause) is evaluated for each row in the CourseStudent table
  - If the studentNumber column matches any computed by the subquery, then the WHERE condition evaluates to TRUE

Subqueries with comparison operators

- Question: What are the course grades for students who have not committed Type “A” academic offences?

```
SELECT *
FROM CourseStudent
WHERE studentNumber != ( SELECT studentNumber
                         FROM StudentOffence
                         WHERE penaltyCode = 'A');
```

- But this isn’t quite correct; what if there were two or more students who had committed an offence?

Subqueries and comparison operators

- If using a subquery without a quantifying operator (IN, SOME, ANY, ALL) in a comparison predicate:
  - The subquery **must return a single value and a single row**
  - If the subquery attempts to return more than a single value (more than one item in the subquery's SELECT list), you get syntax error
  - If the subquery will return more than one row, the server will return a run-time error

Subqueries with IN and NOT IN

- If the subquery could return multiple rows, a **quantified** subquery predicate *must* be used – for example, IN or NOT IN:

  SELECT …

  WHERE *column* IN (SELECT …)

  SELECT …

  WHERE *column* NOT IN (SELECT …)

- Using 'IN' is equivalent to the syntax '= ANY()' or '= SOME()'
- IN subqueries are very commonly used in practice

Subqueries with IN and NOT IN

- Correction to original nested query (using IN):

```
SELECT *
FROM CourseStudent
WHERE studentNumber IN ( SELECT studentNumber
                         FROM StudentOffence
                         WHERE penaltyCode = 'A');
```

Subqueries with IN and NOT IN

- And the negated version:

```
SELECT *
FROM CourseStudent
WHERE studentNumber NOT IN ( SELECT studentNumber
                             FROM StudentOffence
                             WHERE penaltyCode = 'A');
```

Subqueries with IN and NOT IN

- Question: What are the names of the employees who work in the School of Business?

```
SELECT lastName, firstName
FROM Person
WHERE number IN ( SELECT number
                  FROM Employee
                  WHERE schoolCode = 'BUS');
```

Subqueries with = ANY ()

- Question: What are the names of the employees who work in the School of Business?
  - This variant is equivalent

```
SELECT lastName, firstName
FROM Person
WHERE number = ANY ( SELECT number
                     FROM Employee
                     WHERE schoolCode = 'BUS');
```

Subqueries with EXISTS

- Question: What are the names of the employees who work in the School of Business?
  - This variant is also equivalent

Correlation reference to column in the outer block

```
SELECT lastName, firstName
FROM Person
WHERE EXISTS (SELECT *
                    FROM Employee
                    WHERE Person.number = Employee.number
                          AND schoolCode = 'BUS');
```

CONESTOGA
Connect Life and Learning

Subquery Flexibility

- Note that column names don't have to match, as long as the data type is comparable:

```
SELECT lastname, firstname
        FROM person
        WHERE id = (SELECT customer_id
                    FROM order_header
                    WHERE invoice_number = 2345)
```

EXISTS and NOT EXISTS

- An EXISTS predicate evaluates a subquery for the existence of any rows – SQL Server will halt the execution of the subquery once the first row is found and the predicate evaluates to TRUE
  - The expression in the SELECT list of the subquery doesn't matter, it is typically an asterisk (*) but it could be, for example, a literal constant.

```
SELECT lastname, firstname
FROM Person
WHERE EXISTS  (  SELECT *
                 FROM Employee
                  WHERE location = '4A17'  AND
                       Person.number = Employee.number);
```

EXISTS and NOT EXISTS

- Similarly, a NOT EXISTS predicate is evaluated for the existence of any rows – SQL Server will halt the execution of the subquery once the first row is found
  - If a row is found, the predicate evaluates to FALSE
  - The expression in the SELECT list of the subquery doesn't matter, it is typically an asterisk (*) but it could be, for example, a literal constant.

```
SELECT lastname, firstname
FROM Person
WHERE NOT EXISTS  (  SELECT *
                     FROM Employee
                     WHERE location = '4A17'  AND
                          Person.number = Employee.number);
```

EXISTS and NOT EXISTS

- In tuple relational calculus,
  - EXISTS embodies existential quantification, and
  - NOT EXISTS embodies universal quantification
- EXISTS and NOT EXISTS are the canonical forms for nested queries in SQL
- All other forms of subqueries can be translated to either EXISTS or NOT EXISTS and such transformations are commonly done by DBMS optimizers
  - Precisely which variants are used has no effect on performance in a modern database system

Nested Subqueries

- Question: Which employees work in 2A139 in the School of Business?

Nested Subqueries

```
SELECT lastname, firstname
FROM Person
WHERE EXISTS  ( SELECT *
                    FROM Employee
                    WHERE location = '2A139'
                        AND Person.number = Employee.number
                         AND schoolCode IN
                            (SELECT code
                            FROM School
                            WHERE name LIKE '%Business%' )
                    )
ORDER BY lastName;
```

Scalar subquery in a SELECT List

- Subquery is (logically) recomputed for each row in the output
- As with nested queries, a scalar subquery in a query's SELECT list can reference expressions from its parent block
  - Such outer references are treated as a constant value for that subquery's execution
- Example:

```
SELECT e.number, e.schoolCode,
      ( SELECT (p.firstName + ' ' + p.lastName) AS Empname
        FROM Person p
        WHERE p.number = e.number ) as Empname
FROM Employee e
WHERE e.location = '4A17';
```

Subquery Limitations

- The subquery cannot have an ORDER BY clause unless the additional clauses FOR XML or SELECT TOP are used.
- A subquery can be any query expression including one that uses set operations (UNION, INTERSECT, EXCEPT)