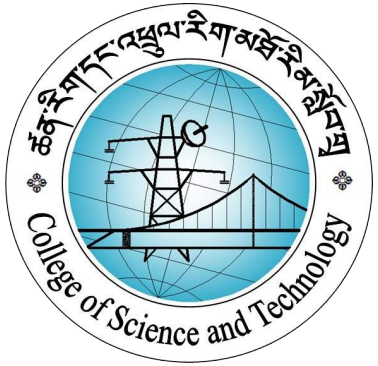


Royal University of Bhutan

DBS101 Database Systems Fundamentals

Lesson 8

Learning Outcomes

1. Understand null values in SQL.
2. Use aggregate functions in SQL.
3. Write nested subqueries.
4. Modify databases using SQL.

Null values in SQL

In SQL, some records in a table may not have values for every field, and such fields are termed as NULL values.

Null values represent missing or unknown information in databases and require special handling.

Null values present special problems in relational operations, including arithmetic operations, comparison operations, and set operations.

Null values in SQL

Arithmetic Operations: Result is null if any input is null. Example: If r.A is null, then r.A + 5 is also null

Comparison Operations: Result is "unknown" (not true/false)

- Example: "1 < null" evaluates to unknown
- SQL uses three-valued logic: true, false, and unknown
- Predicates to test: IS NULL, IS NOT NULL, IS UNKNOWN
- IS NULL checks if a value is null, IS NOT NULL checks if it's not null, and we can even test if a comparison result IS UNKNOWN.

Null values in SQL

AND Operation

- true AND unknown = unknown
- false AND unknown = false
- unknown AND unknown = unknown

OR Operation

- true OR unknown = true
- false OR unknown = unknown
- unknown OR unknown = unknown

NOT Operation

- NOT unknown = unknown

In WHERE clauses: tuples with false OR unknown predicates are filtered out

Null values in SQL

Null Values in Result Processing

Duplicate Elimination (SELECT DISTINCT)

- Two values are identical if both are null or both are non-null and equal
- Example: {('A',null), ('A',null)} are treated as identical tuples

Set Operations (UNION, INTERSECTION, EXCEPT)

- Same approach: nulls in the same positions are treated as identical
- Different from comparison behavior where "null = null" would be unknown

Note: The SELECT DISTINCT statement is used to return only distinct (different) values.

Aggregate Functions

Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value.

- AVG(COL): The average of the values in COL
- MIN(COL): The minimum value in COL
- MAX(COL): The maximum value in COL
- COUNT(COL): The number of tuples in the relation

Lets go to Practical 3

Group By clause

The GROUP BY statement in SQL is used to arrange identical data into groups based on specified columns. If a particular column has the same values in multiple rows, the GROUP BY clause will group these rows together.

Example: here

The Having Clause

States a condition that applies to groups rather than to tuples.

Example:

```
select deptname, avg (salary) as avgSalary
from instructor
group by deptname
having avg (salary) > 42000;
```

Lets go to Practical 3

Aggregation on NULL and Boolean Values

In SQL, aggregate operators like sum() ignore null values in their input collection. Therefore, when calculating sums, null values are disregarded rather than resulting in a null sum.

The SQL:1999 standard introduced a Boolean data type allowing values true, false, and unknown. Aggregate functions like some() and every() can operate on collections of Boolean values, computing disjunction and conjunction, respectively.

Date and Time Manipulation

Operations to manipulate DATE and TIME attributes.

Can be used in either output or predicates.

The specific syntax for date and time operations varies wildly across systems.

Lets go to Practical 3

Output Redirection

Instead of having the result a query returned to the client (e.g., terminal), you can tell the DBMS to **store the results into another table**. You can then access this data in subsequent queries.

Lets go to Practical 3

Output Control

ORDER BY and LIMIT clauses provide output ordering and restriction on result sets.

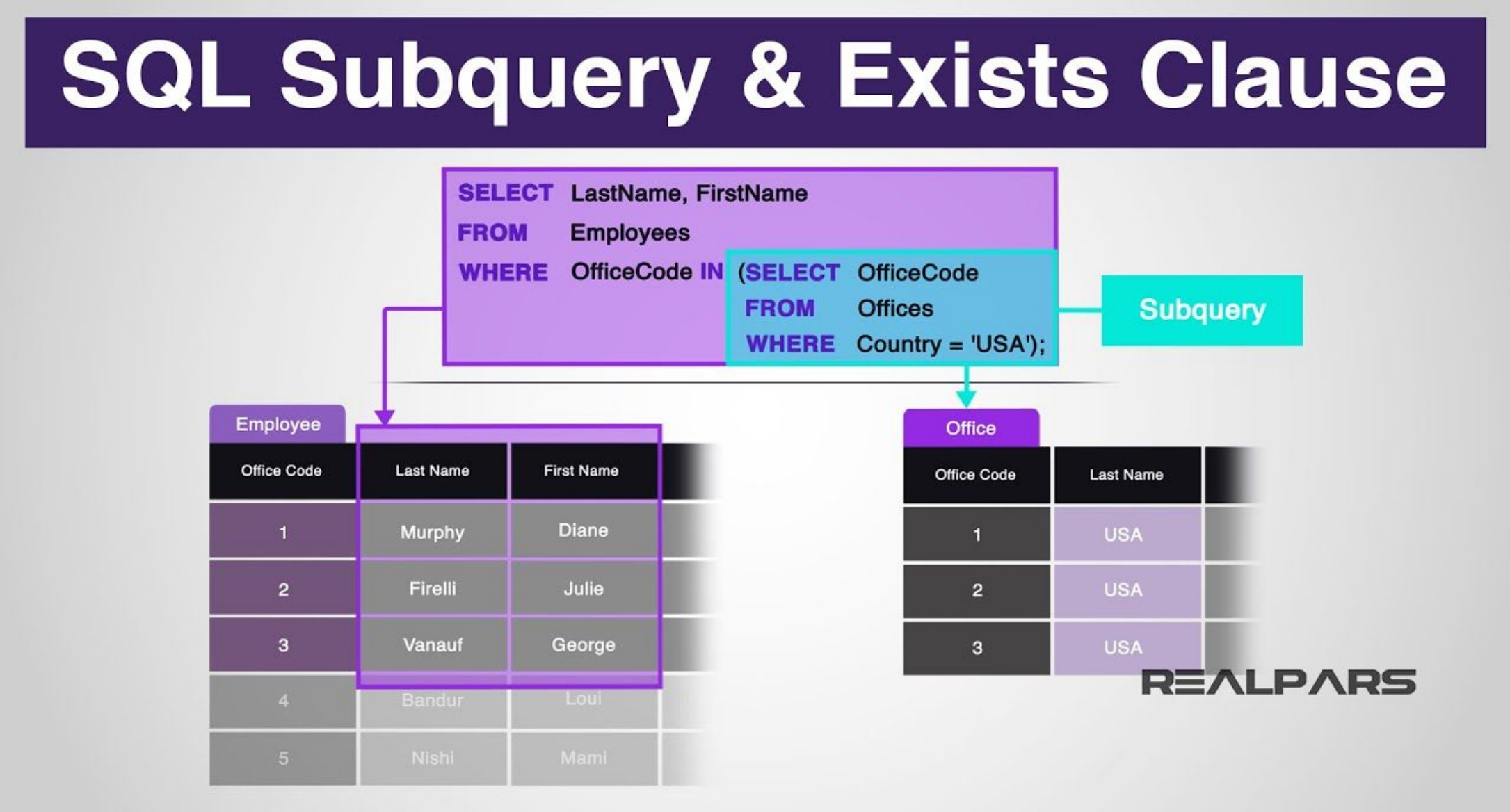
Lets go to Practical 3

Nested Subqueries

A subquery is a select-from-where expression that is nested within another query.

```
select distinct course id
from section
where semester = 'Fall' and year= 2017 and
                 course_id in (select course id
                 from section
                 where semester = 'Spring' and
                 year= 2018);
```

Nested Subqueries



Nested Subqueries

Invoke queries inside of other queries to execute more complex logic within a single query. Nested queries are often difficult to optimize.

The scope of outer query is included in an inner query (i.e. the inner query can access attributes from outer query), but not the other way around.

Inner queries can appear in almost any part of a query

Set Membership

SQL allows testing tuples for membership in a relation. The **IN** connective tests for set membership, where the set is a collection of values produced by a select clause. The **NOT IN** connective tests for the absence of set membership

Set Membership

Example 1: SELECT * FROM employees WHERE department IN ('HR', 'Finance')

- Retrieves employees who belong to departments HR and Finance.
-

Example 2: SELECT * FROM employees WHERE department NOT IN ('HR', 'Finance')

- Retrieves employees who do not belong to departments HR and Finance.

Set Comparison

SQL allows set comparison as well.

Set comparison involves comparing sets of values using operators like IN, EXISTS,ALL and ANY.

Lets go to Practical 3

Testing Relations

SQL allows for testing of relations:

1. Testing for empty relations
2. Test for the Absence of Duplicate Tuples

Lets go to Practical 3

Subqueries in the FROM clause

SQL allows a subquery expression to be used in the from clause.

The key concept applied here is that any select-from-where expression returns a relation as a result and, therefore, can be inserted into another select-from-where anywhere that a relation can appear.

Lets go to Practical 3

The With Clause

The WITH clause, also known as a Common Table Expression (CTE), allows you to define temporary result sets that can be referenced within the scope of a single SQL statement, including SELECT, INSERT, UPDATE, or DELETE statements.

This can help make complex queries more readable and maintainable by breaking them down into smaller, more manageable parts.

Lets go to Practical 3

Scalar Subqueries

SQL allows subqueries to occur wherever an expression returning a value is permitted, provided the subquery returns only one tuple containing a single attribute; such subqueries are called scalar subqueries.

Scalar subqueries are commonly used within a larger SQL statement where a single value is expected, such as in a SELECT list, WHERE clause, or as part of an expression.

Lets go to Practical 3

Nested Queries Results Expressions

- ALL: Must satisfy expression for all rows in sub-query.
- ANY: Must satisfy expression for at least one row in sub-query.
- IN: Equivalent to =ANY().
- EXISTS: At least one row is returned.

Modification of Database

Deletion: A delete request is expressed in much the same way as a query. We can delete only whole tuples; we cannot delete values on only particular attributes. SQL expresses a deletion by:

```
delete from r
where P;
```

Modification of Database

Insertion: To insert data into a relation, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted.

Example:

```
insert into course
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

Modification of Database

Updates: To update values in a table the update statement can be used.

Example:

```
update instructor
set salary= salary * 1.05;
```

Window Functions

A window function perform “sliding” calculation across a set of tuples that are related. Like an aggregation but tuples are not grouped into a single output tuple.

It can be used to calculate aggregates, rankings, and perform other analytical tasks without the need for subqueries or self-joins.

Window Functions

Functions: The window function can be any of the aggregation functions discussed above.

There are also also special window functions:

1. **ROW NUMBER:** The number of the current row.
2. **RANK:** The order position of the current row.

Grouping: The OVER clause specifies how to group together tuples when computing the window function. Use PARTITION BY to specify group.

Lets go to Practical 3

Next Session

Unit IV : Intermediate and Advanced SQL

4.1 Join Expressions
4.2 Views
4.3 Transactions
4.4 Integrity Constraints
4.5 SQL Data Types and Schemas
4.6 Index Definition in SQL
4.7 Authorization
4.8 Accessing SQL from a Programming Language
4.9 Functions and Procedures
4.10 Triggers
4.11 Recursive Queries
4.12 Advanced Aggregation Features