ANSI SQL

Sub-queries

LEVEL – LEARNER

Icons Used

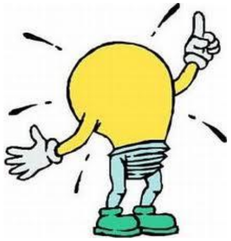
Questions

Hands-on Exercise

Test Your Understanding

Reference

Summary



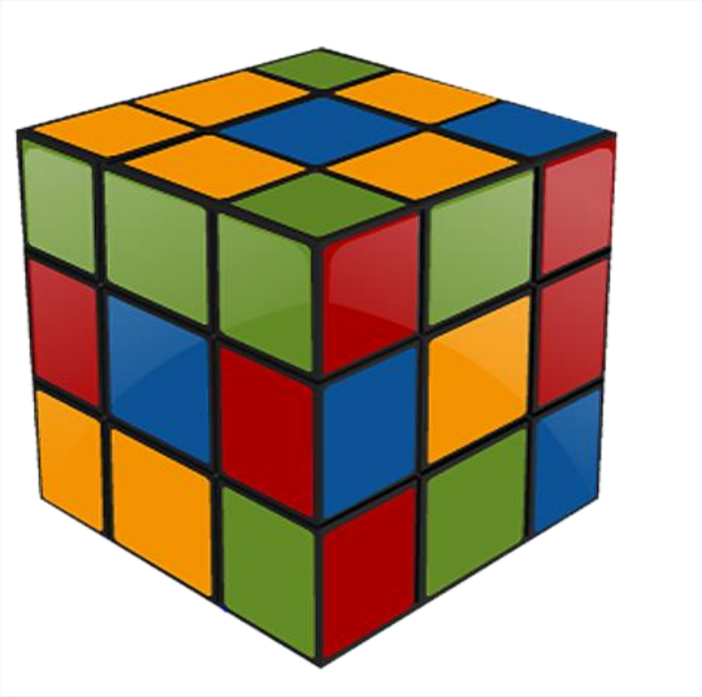
Rule

Overview

This session will give an overview of Sub-queries in SQL.

Objectives

- At the end of this session, you will be able to:
  - Define Sub-queries and its advantages.
  - Identify the rules of Sub-queries.
  - Describe how to use Sub-queries with the SELECT, INSERT, UPDATE, and DELETE statements.
  - Define the types of Sub-queries.
  - Define the use of IN, NOT IN, ALL, ANY, SOME, EXISTS, and NOT EXISTS.
  - Define the use of correlated Sub-queries.
  - Describe the use of difference between correlated and non correlated Sub-query.



Recap Case Study

- For complete understanding of ANSI SQL we are going to make use of **Product Management System** (**PMS**) for ABC Traders.
- ABC Traders is a company which buys collectible model cars, trains, trucks, buses, and trains and ships them directly from manufacturers and sells them to distributors across the globe.
- In order to manage the stocking, supply and payment transactions the above software is developed.
- As per the requirement of the trading company a inventory system is developed to collect the information of products and customers and their payment processing.

Database Tables

- There are many entities involved in **PMS.**
- We will be dealing with the PMS as given below throughout this course.

Offices

To maintain information of offices. For example, office code, address, city, and so on.

Customer

To maintain customer details. For example, customer name and address.

Employees

To maintain employee details. For example, ID, name, and so on.

Products

To maintain information of products. For example, product ID, name, and so on.

Payments

To maintain information of payments done. For example, payment date, amount, and so on.

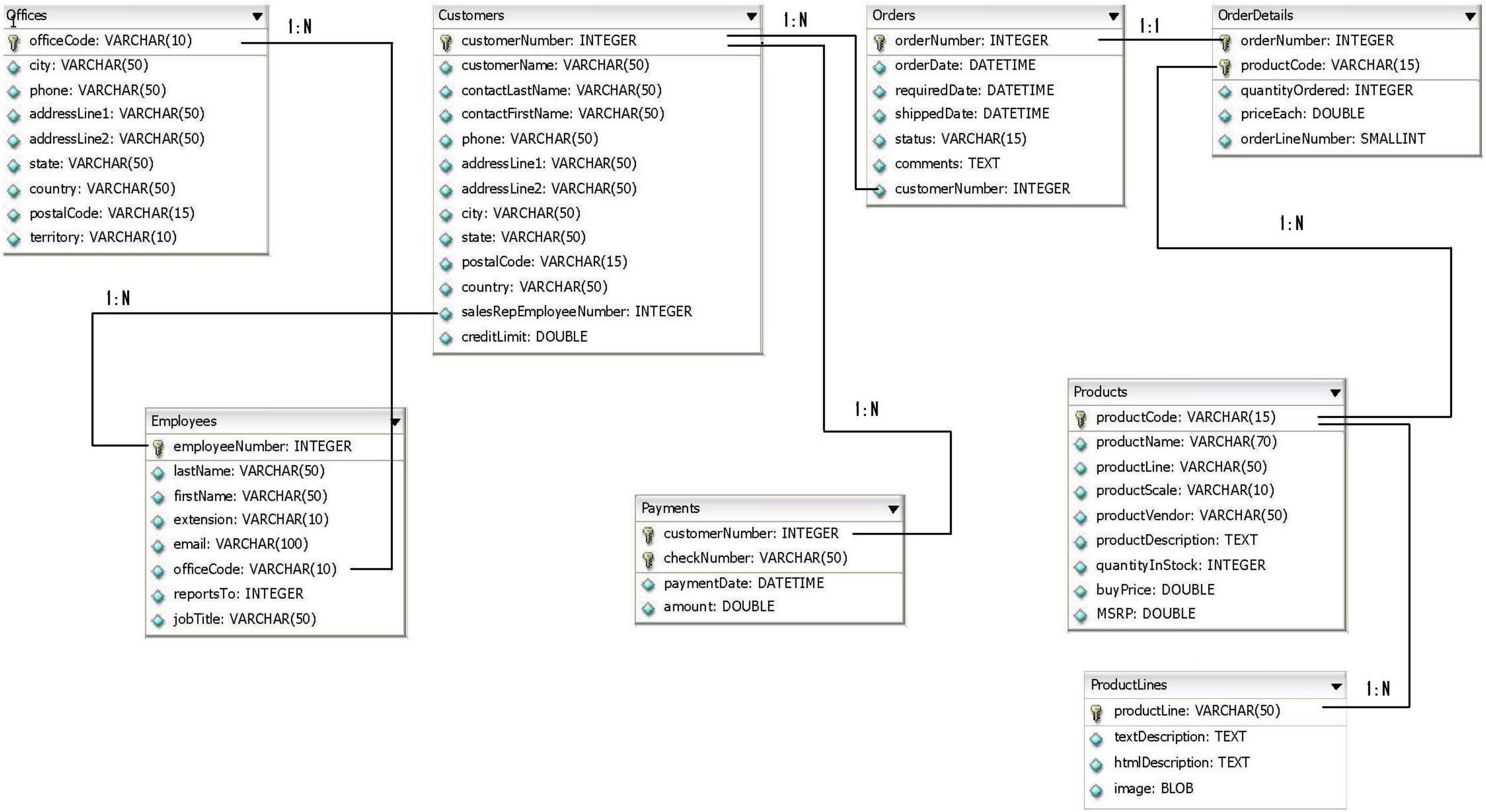
Orders

To maintain Orders done by customers. For example, order no., date, and so on.

Order Details

To maintain Orders done by customers. For example, order no., date, and so on.

Schema Diagram



Offices
- officeCode: VARCHAR(10)
- city: VARCHAR(50)
- phone: VARCHAR(50)
- addressLine1: VARCHAR(50)
- addressLine2: VARCHAR(50)
- state: VARCHAR(50)
- country: VARCHAR(50)
- postalCode: VARCHAR(15)
- territory: VARCHAR(10)

Customers
- customerNumber: INTEGER
- customerName: VARCHAR(50)
- contactLastName: VARCHAR(50)
- contactFirstName: VARCHAR(50)
- phone: VARCHAR(50)
- addressLine1: VARCHAR(50)
- addressLine2: VARCHAR(50)
- city: VARCHAR(50)
- state: VARCHAR(50)
- postalCode: VARCHAR(15)
- country: VARCHAR(50)
- salesRepEmployeeNumber: INTEGER
- creditLimit: DOUBLE

Orders
- orderNumber: INTEGER
- orderDate: DATETIME
- requiredDate: DATETIME
- shippedDate: DATETIME
- status: VARCHAR(15)
- comments: TEXT
- customerNumber: INTEGER

OrderDetails
- orderNumber: INTEGER
- productCode: VARCHAR(15)
- quantityOrdered: INTEGER
- priceEach: DOUBLE
- orderLineNumber: SMALLINT

Employees
- employeeNumber: INTEGER
- lastName: VARCHAR(50)
- firstName: VARCHAR(50)
- extension: VARCHAR(10)
- email: VARCHAR(100)
- officeCode: VARCHAR(10)
- reportsTo: INTEGER
- jobTitle: VARCHAR(50)

Payments
- customerNumber: INTEGER
- checkNumber: VARCHAR(50)
- paymentDate: DATETIME
- amount: DOUBLE

Products
- productCode: VARCHAR(15)
- productName: VARCHAR(70)
- productLine: VARCHAR(50)
- productScale: VARCHAR(10)
- productVendor: VARCHAR(50)
- productDescription: TEXT
- quantityInStock: INTEGER
- buyPrice: DOUBLE
- MSRP: DOUBLE

ProductLines
- productLine: VARCHAR(50)
- textDescription: TEXT
- htmlDescription: TEXT
- image: BLOB

Scenario

Hi! I am Tim and am back again!
Now that you have created tables with constraints and used different operators, functions clauses and met my earlier requirements using Joins, I would want you to take care of some other requirements that I have. This will need you to use a query within a query.

Let us use Sub-queries to meet Tim's requirements..

Do You Know?

- Have you heard about Nesting of Queries in SQL?

Question?

- Nested Queries can be applied only to SELECT clause.

Answer: NO

Advantages of Sub-queries

- What is a Sub-query?
  - A Sub-query is a query within a query. It is also called an inner query or a nested query.

- Advantages of Sub-queries:
  - They allow queries that are structured so that it is possible to isolate each part of a statement.
  - They provide alternative ways to perform operations that would otherwise require complex joins and unions.
  - Many people find Sub-queries are more readable than complex joins or unions. Indeed, it was the innovation of Sub-queries that gave people the original idea of calling the early SQL “Structured Query Language.”

Sub-query Rules

- There are a few rules that Sub-queries must follow:
  - Sub-queries must be enclosed within parentheses.
  - A Sub-query can have only one column in the SELECT clause, unless multiple columns are in the main query for the Sub-query to compare its selected columns.
  - An ORDER BY cannot be used in a Sub-query, although the main query can use an ORDER BY.
  - The GROUP BY can be used to perform the same function as the ORDER BY in a Sub-query.
  - Sub-queries that return more than one row can only be used with multiple value operators, such as the IN operator.
  - A Sub-query cannot be immediately enclosed in a set function.
  - BETWEEN operator cannot be used with a Sub-query; however, BETWEEN can be used within the Sub-query.

Scenario

I want to display name and phone number of the customers who have made the payments.

Let’s use a Sub-query with the SELECT statement to meet Tim’s requirement.

Sub-query: SELECT Statement

- Sub-queries with the SELECT Statement:
  - Sub-queries are most frequently used with the SELECT statement.
  - The basic syntax is as follows:

```
SELECT column_name [,column_name]
FROM table1 [,table2]
WHERE column_name OPERATOR
       (SELECT column_name
       [,column_name]
       FROM table1 [,table2]
       WHERE row_condition );
```

- Example:

```
SELECT customers.customername, customers.phone
FROM customers
WHERE customernumber IN (SELECT customernumber FROM payments);
```

Scenario

I want to insert records into a table from another table based on a certain condition, using Sub-query.

Let us use a Sub-query with the INSERT statement to meet Tim's requirement.

Sub-query: INSERT Statement

- Sub-queries with the INSERT Statement:
  - Sub-queries can also be used with INSERT statements. The INSERT statement uses the data returned from the Sub-query to insert into another table.
  - The selected data in the Sub-query can be modified with any of the character, date, or number functions.
  - The basic syntax is as follows:

```
INSERT INTO table_name [(column1,
[,column2])]
          SELECT [* | column1
         [,column2]
         FROM table1 [,table2]
         [ WHERE VALUE OPERATOR];
```

Note: Create a new table USA_Offices with similar structure as that of Offices.

Subquery: INSERT Statement (Contd.)

```
CREATE TABLE USA_Offices (
  officeCode VARCHAR(10) NOT NULL,
  city VARCHAR(50) NOT NULL,
  phone VARCHAR(50) NOT NULL,
  addressLine1 VARCHAR(50) NOT NULL,
  addressLine2 VARCHAR(50) NULL,
  state VARCHAR(50) NULL,
  country VARCHAR(50) NOT NULL,
  postalCode VARCHAR(15) NOT NULL,
  territory VARCHAR(10) NOT NULL,
  PRIMARY KEY (officeCode)
);
```

- Let us see how we can copy records having country as USA, from Offices table into USA_Offices table, using Subquery with INSERT statement.

```
INSERT INTO USA_Offices
SELECT * FROM Offices
WHERE country IN (SELECT country FROM offices
                                WHERE country = 'USA');
```

Scenario

I want to update values in ‘addressLine2’ column of USA_Offices to ‘Suite 327’. This can be done only to the records which has the ‘city’ value equals to ‘Boston’ in Office tables.

Let us use a Sub-query with the UPDATE statement to meet Tim’s requirement.

Subquery – UPDATE Statement

- Subqueries with the UPDATE Statement:
  - The Subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a Subquery with the UPDATE statement.

- The basic syntax is as follows:

```
UPDATE table
SET column_name = new_value
[WHERE OPERATOR [VALUE]
     (SELECT column_name FROM
     table_name)
     [WHERE row_condition)];
```

Note: We will use the same new table created earlier, USA_Offices and Offices table.

Sub-query: UPDATE Statement (Contd.)

- Using a Sub-query with the UPDATE statement:
  - Let us see, how we can update values in ‘addressLine2’ column of USA_Offices to ‘Suite 327’ if the ‘city’ value of these records appear in the those records of Office tables where city value is ‘Boston’.

```
UPDATE USA_Offices
     SET addressLine2 = 'Suite 327'
      WHERE city IN (SELECT city FROM Offices
                     WHERE city LIKE '%Boston%');
```

Scenario

I want to delete records from USA_Offices where the values in city column of USA_Offices appear in the values in city column of Offices for ‘NY’ state.

Let us use a Sub-query with the DELETE statement to meet Tim’s requirement.

Sub-query: DELETE Statement

- Sub-queries with the DELETE Statement:
  - The Sub-query can be used in conjunction with the DELETE statement like with any other statements mentioned before.

- The basic syntax is as follows:

```
DELETE FROM table_name
[WHERE OPERATOR [VALUE]
     (SELECT column_name
     FROM table_name)
     [WHERE) ];
```

Note: We will use the same new table created earlier, USA_Offices and Offices table.

Subquery – DELETE Statement (Contd.)

- Using a Subquery with the DELETE statement:
  - Let us see, how we can delete records from USA_Offices where the values in ‘city’ column of USA_Offices appear in the values in city column of Offices for ‘NY’ state.

```
DELETE FROM USA_Offices
    WHERE city IN (SELECT city FROM Offices
                         WHERE state LIKE '%NY%');
```

Scenario

I want to display customer Number, check Number, amount for those customers whose have paid amount more than the average amount paid by the customers.

Let us understand the different types of Sub-queries – Scalar, Single Row, and Multiple Row. Let's use a scalar Sub-query to meet the above requirement given by Tim.

Scalar Sub-query

- Scalar Sub-query:
  - A scalar Sub-query returns a variable like a number, date, or string.
  - A scalar Sub-query returns only one column for a single row and is also known as an SQL expression. You can use a scalar Sub-query in:
    - the WHERE clause of a SELECT
    - the VALUES clause of an INSERT statement
    - the SET or WHERE clauses of an UPDATE
    - the WHERE clause of a DELETE statement

- Example:

```
SELECT customerNumber, checkNumber, amount
FROM payments
WHERE amount > (SELECT AVG(amount) FROM payments);
```

Scenario

I want to display the Order Number and required date for all orders whose requirement date is less than the payment date for check number 'HQ336336’.

Let’s use a single row Sub-query to meet Tim’s requirement.

Single Row Sub-query

- Single Row Sub-query:
  - A single row Sub-query returns all columns for a single row.
  - You can use a single row Sub-query in:
    - the INSERT statement when it provides all required values for a single row insertion
    - the SET or WHERE clauses of an UPDATE
    - the WHERE clause of a DELETE statement
- Legal operators for row Sub-query comparisons are: **> < >= <> != <=>**
- Example:

```
SELECT o.OrderNumber, o.requireddate
FROM orders o
WHERE o.requireddate < (SELECT p.paymentdate
                        FROM payments p
                             WHERE p.checknumber = 'HQ336336');
```

Multiple Row Sub-query

- Multiple Row Sub-query:
  - A multiple row Sub-query returns an aggregate table, which is a filtered result set of one or more columns and one or more rows.
  - You can use a multiple row Sub-query in SELECT, INSERT, UPDATE, or DELETE statements.
  - A multiple row Sub-query can replace the VALUES clause in an INSERT statement, like the single-row Sub-query discussed earlier.
  - You can put a multiple row Sub-query in the WHERE clause of a SELECT, UPDATE, and DELETE statement. However, unlike scalar and single-row Sub-queries, multiple row Sub-queries cannot work with the equality, =, operator.
  - Multiple row Sub-queries require either the IN operator, or and equality/inequality operator, combined with an ALL, ANY, or SOME, IN, NOT IN operator.
  - You typically use a multiple row Sub-query when you lookup a related set of information that has more than one row.

Note: All columns must return the same number of rows of data.

Scenario

I want to display the order number and customer number from orders for all customers who have made a payment of more than 15000.

Let us use a multiple row Sub-query to meet Tim's requirement.

Multiple Row Sub-query (Contd.)

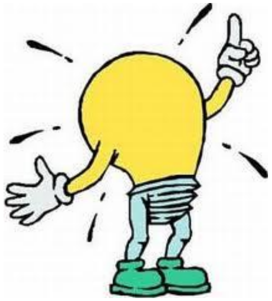
- Let us see an example of multiple row Sub-query which use IN.

```
SELECT o.OrderNumber, o.customerNumber
FROM orders o
WHERE o.customerNumber IN (SELECT p.customerNumber
                           FROM payments p
                                 WHERE p.amount>15000);
```

IN, NOT IN, ALL, ANY, and SOME

- IN:
- ANSI Syntax: `expression IN (Sub-query)`
  - An IN Sub-query condition is TRUE if the value of the expression matches one or more of the values from the Sub-query.

- NOT IN:
- ANSI: `expression NOT IN (Sub-query)`
  - An NOT IN Sub-query condition is TRUE if the value of the expression matches none of the values from the Sub-query.

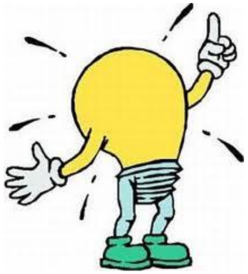


Rules:

- The keyword IN is equivalent to the =ANY specification.
- The keywords NOT IN are equivalent to the !=ALL specification.
- Because the IN Sub-query tests for the presence of rows, duplicate rows in the Sub-query results do not affect the results of the main query.
- Therefore, the UNIQUE or DISTINCT keyword in the Sub-query has no effect on the query results, although not testing duplicates can improve query performance.

IN, NOT IN, ALL, ANY, and SOME (Contd.)

- ALL:
- ANSI Syntax: `expression/operand  comparison_operator ALL (Sub-query)`
  - The ALL keyword specifies that the search condition is TRUE if the comparison is TRUE for every value that the Sub-query returns.
  - If the Sub-query returns no value, the condition is TRUE.
- ANY/SOME:
- ANSI Syntax: `expression/operand comparison_operator ANY (Sub-query)`
  `expression/operand comparison_operator SOME (Sub-query)`
  - The SOME keyword is a synonym for ANY.
  - The ANY keyword denotes that the search condition is TRUE if the comparison is TRUE for at least one of the values that is returned. If the Sub-query returns no value, the search condition is FALSE.



Rule:

- NOT IN is an alias for <> ALL. Thus, these two statements are the same

Correlated Sub-query

- ANSI Syntax:

```
SELECT <columnName> [..,<columnName>]
FROM <table1>
WHERE <columnName> {=|<|>|<=|>=|<>}
 (SELECT {MIN|MAX|AVG} { <columnName>}
 FROM <table>
 WHERE<table1>.<columnName>=<table>.<columnName>
 […AND <table1>.<columnName>=<table>.<columnName>]
);
```

- A correlated Sub-query is a Sub-query that refers to a column of a table that is not in its FROM clause. The column can be in the Projection clause or in the WHERE clause. To find the table to which the queries refer, search the uncorrelated column until a correlation can be found.
- In general, correlated Sub-queries will diminish performance. Use the table name or alias in the Sub-query so that there is no doubt as to which table the column is in.

Correlated Subquery (Contd.)

- The important feature of a correlated Subquery is that, because it depends on a value from the outer SELECT, it must be executed repeatedly, once for every value that the outer SELECT produces.

- Example:

```
SELECT c1.customernumber, c1.customername
 FROM customers AS c1
 WHERE creditLimit > (SELECT AVG(creditLimit)
                      FROM customers AS c2
                             WHERE c2.city=c1.city);
```

EXISTS and NOT EXISTS

- ANSI Syntax:
  - `{where | having} [NOT] EXISTS (Sub-query)`
- EXISTS and NOT EXISTS are SQL conditions/functions that execute a Sub-query and return either TRUE or FALSE depending on whether if rows were found or not.
- EXISTS can only be used in the WHERE clause of a query. The Sub-query can refer to columns in the parent query (called a correlated Sub-query).
- Note that EXISTS will not scan all rows in the Sub-query, only one row is required to determine whether the outcome is TRUE or FALSE. However, NOT EXISTS must scan all rows, which may cause performance problems.
- The Sub-query does not actually produce any data, but returns a value of TRUE or FALSE.
- The Sub-query always contains a reference to a column of the table in the main query. If you use an aggregate function in an EXISTS Sub-query that includes no HAVING clause, at least one row is always returned.

EXISTS and NOT EXISTS (Contd.)

- Below are examples which use EXISTS and NOT EXISTS with Sub-query.

```
SELECT customernumber
 FROM customers AS a
 WHERE EXISTS
  (
   SELECT * FROM orders AS b
   WHERE a.customernumber =
               b.customernumber
   AND status ='Shipped'
  );
```

```
SELECT customernumber
FROM customers AS a
WHERE NOT EXISTS
  (
    SELECT * FROM orders AS b
    WHERE a.customernumber =
               b.customernumber
    AND status <> 'Shipped'
  );
```

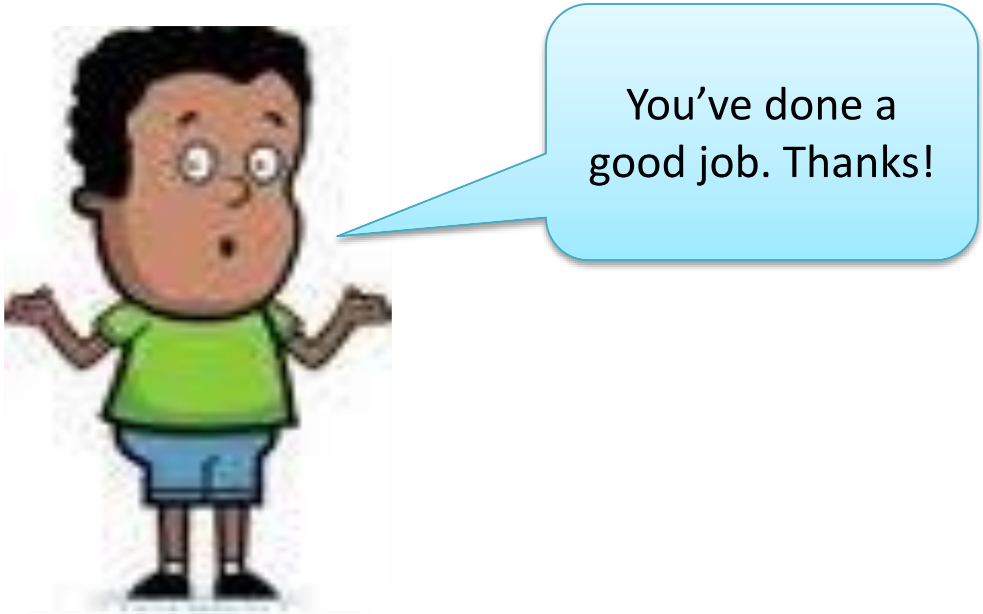
- The first query uses EXISTS keyword in WHERE clause to return a list of customer numbers whose order status is ‘Shipped’.
- Note that the outer query only returns a row where the Sub-query returns TRUE.
- The second query uses NOT EXISTS keyword in WHERE clause to return a list of customer numbers whose order status is a value other than ‘Shipped’.

Correlated vs. Non-correlated Sub-query

- Difference between Correlated and Non-correlated Sub-query:
  - In case of correlated Sub-query inner query depends on outer query while in case of non correlated query inner query or Sub-query doesn't depend on outer query and run by its own.
  - In case of correlated Sub-query, outer query is executed before inner query or Sub-query while in case of non correlated Sub-query inner query executes before outer query.
  - Common example of correlated Sub-query is using EXISTS and NOT EXISTS keywords while non correlated query mostly use IN or NOT IN keywords.

Any Questions?

Hands-On



Now that we are well versed with Sub-queries, let us help Tim meet his requirements for Alliance Online Feedback System. Please check Hands-On document for more details.

Activity

- Now that we are well versed with commands let’s test our understanding using a short case study.

Course **M**anagement **S**ystem (**CMS**) **Cognizant Academy**

Outcome Assured…

- **Case Study Scenario:**
  - — This case study is to develop a Course Management System (CMS) for Cognizant Academy. The following are the two use cases for which the database needs to be designed.
- **Add Course**
  - — To add the course details into the Course Management System.
- **Retrieve Course**
  - — Retrieve the courses stored in the system and display it.
- The courses to be added will have the following attributes:
  - — Course Code, Course Name, Number of participants, Course Description, Course Duration, Course Start Date, and Course Type.

myAcademy

What have you learnt today?

Lend a Hand

- **Pre-Requisite:**
  - Insert the following records
  - Add two new courses in course_info table
  - Add the course fees for the two courses in course_Fees with fees amount < 1500
  - Enroll two students to the newly added courses

- **Problem 1:**
  - Write a query which fetches the student id for students who have enrolled for at least one course whose fees is less than 1500.

- **Problem 2:**
  - Write a query which fetches the student id and student name for students who have enrolled for at least one course whose fees is less than 1500.

Lend a Hand - Solution

- **Solution 1:**

```
SELECT student_id
FROM student_courses
WHERE course_code IN (SELECT course_code
                      FROM course_fees
                      WHERE special_fees <1500);
```

- **Solution 2:**

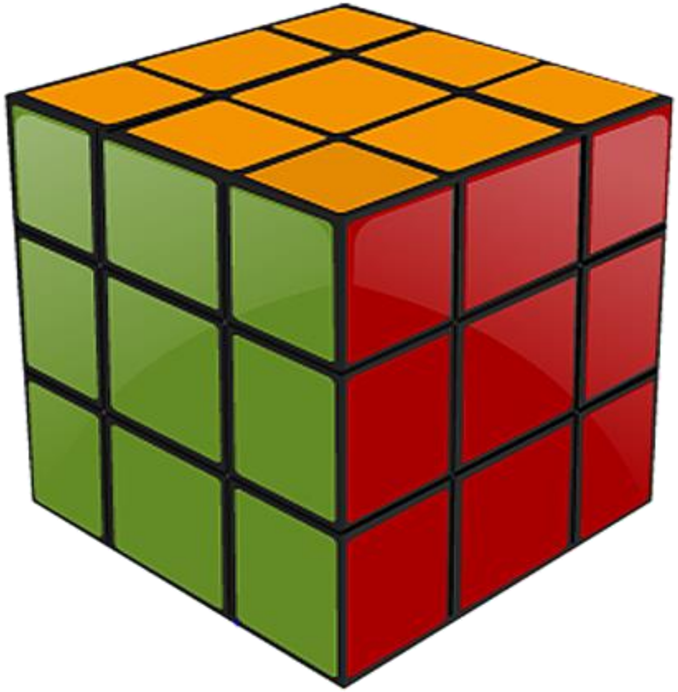
```
SELECT c.student_id, s.first_name
FROM student_courses c, student_info s
WHERE s.student_id = c.student_id
       AND course_code IN
                     (SELECT course_code
                      FROM course_fees
                      WHERE special_fees <1500);
```

Check Your Understanding

- What are Sub-queries?
- What are the advantages of Sub-queries?
- Which are the rules that Sub-queries follow?
- How to use Sub-queries with the SELECT, INSERT, UPDATE, and DELETE Statement?
- What are the types of Sub-queries?
- What is the use of IN, NOT IN, ALL, ANY, and SOME?
- What are Correlated Sub-queries?
- What is the use of EXISTS and NOT EXISTS?
- What is the difference between Correlated and Non Correlated Sub-query?

Summary

- The key points covered in this chapter are:
  - The advantage of a Sub-query is that it allows queries that are structured in order to isolate each part of a statement.
  - Sub-queries must be enclosed within parentheses.
  - The different types of Sub-queries are scalar Sub-query, single row Sub-query, and multiple row Sub-query.
  - EXISTS and NOT EXISTS are SQL conditions that executes a Sub-query and return either TRUE or FALSE depending on whether if rows were found or not.



Source

- **Weblinks:**
  - http://en.wikipedia.org/wiki/SQL#Subqueries

Disclaimer: Parts of the content of this course are based on the materials available from the websites and books listed above. The materials that can be accessed from the linked sites are not maintained by Cognizant Academy and we are not responsible for the contents thereof. All trademarks, service marks, and trade names in this course are the marks of the respective owner(s).

Change Log

Version Number	Changes made			
V1.0	Initial Version			
V1.1	Slide No.	Changed By	Effective Date	Changes Effected
	1-47	Learning Content Team CI Team CATP Technical Team	17-05-2013	Base-lining content

ANSI SQL

You have successfully completed the session on Subqueries