Assignment No: 04

Title: Design Design at least 10 SQL queries for suitable database application using SQL DML statements: all types of Join, Sub-Query and View.

Objectives: To understand and demonstrate DML statements on various SQL objects

Outcomes:

- Students will be able to learn and understand various DML queries like all types of Join, Sub-Query and View
- Students will be able to learn and understand various DML queries with set operations and flexible views of databases .

Hardware requirements: Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements: Ubuntu 14 Operating System, MySQL

Theory:

SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them. Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Try it Yourself »

and it will produce something like this:

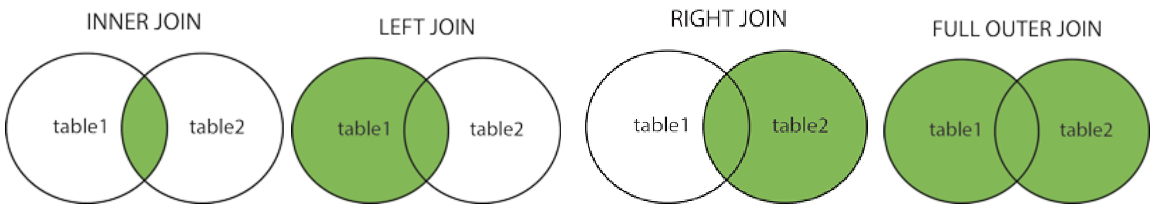
OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996

10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table

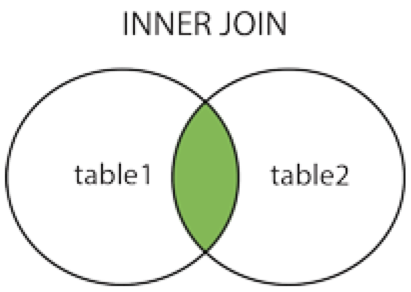


SQL INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;
```



Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

And a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

SQL INNER JOIN Example

The following SQL statement selects all orders with customer information:

Example

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Try it Yourself »

Note: The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not be shown!

JOIN Three Tables

The following SQL statement selects all orders with customer and shipper information:

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

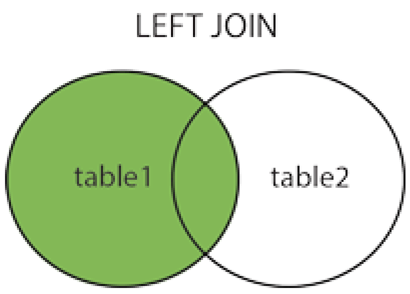
SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.



Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

SQL LEFT JOIN Example

The following SQL statement will select all customers, and any orders they might have:

Example

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

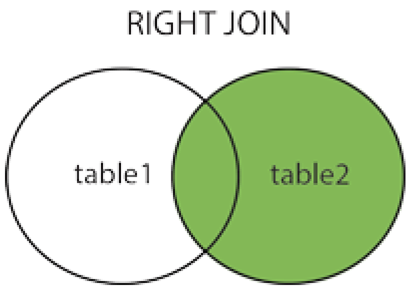
SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```

Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

And a selection from the "Employees" table:

EmployeeID	LastName	FirstName	BirthDate	Photo
1	Davolio	Nancy	12/8/1968	EmpID1.pic
2	Fuller	Andrew	2/19/1952	EmpID2.pic
3	Leverling	Janet	8/30/1963	EmpID3.pic

SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they might have placed:

Example

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

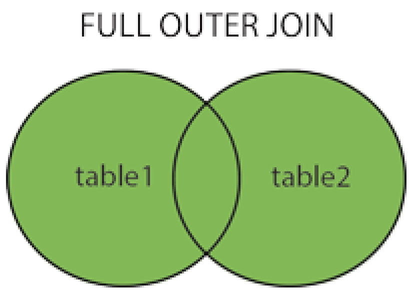
SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword return all records when there is a match in either left (table1) or right (table2) table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;
```



Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3

10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

A selection from the result set may look like this:

CustomerName	**OrderID**
Alfreds Futterkiste	10308
Ana Trujillo Emparedados y helados	10365
Antonio Moreno Taquería	10382
	10351

The FULL OUTER JOIN keyword returns all the rows from the left table (Customers), and all the rows from the right table (Orders). If there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

SQL Self JOIN

A self JOIN is a regular join, but the table is joined with itself.

Self JOIN Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

SQL Self JOIN Example

The following SQL statement matches customers that are from the same city:

Example

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

SQL CREATE VIEW Examples

If you have the Northwind database you can see that it has several views installed by default.

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table. The view is created with the following SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued = No;
```

Then, we can query the view as follows:

```
SELECT * FROM [Current Product List];
```

Another view in the Northwind sample database selects every product in the "Products" table with a unit price higher than the average unit price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, UnitPrice
FROM Products
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Products);
```

We can query the view above as follows:

```
SELECT * FROM [Products Above Average Price];
```

Another view in the Northwind database calculates the total sale for each category in 1997. Note that this view selects its data from another view called "Product Sales for 1997":

```
CREATE VIEW [Category Sales For 1997] AS
SELECT DISTINCT CategoryName, Sum(ProductSales) AS CategorySales
FROM [Product Sales for 1997]
GROUP BY CategoryName;
```

We can query the view above as follows:

```
SELECT * FROM [Category Sales For 1997];
```

We can also add a condition to the query. Let's see the total sale only for the category "Beverages":

```
SELECT * FROM [Category Sales For 1997]
WHERE CategoryName = 'Beverages';
```

SQL Updating a View

You can update a view by using the following syntax:

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Now we want to add the "Category" column to the "Current Product List" view. We will update the view with the following SQL:

```
CREATE OR REPLACE VIEW [Current Product List] AS
SELECT ProductID, ProductName, Category
FROM Products
WHERE Discontinued = No;
```

SQL Dropping a View

You can delete a view with the DROP VIEW command.

SQL DROP VIEW Syntax

```
DROP VIEW view_name;
```

subquery in SQL?

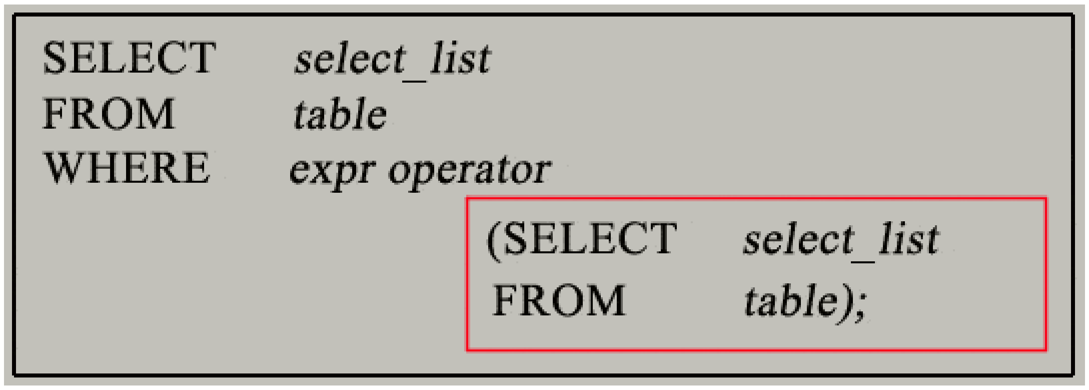
A subquery is a SQL query nested inside a larger query.

- A subquery may occur in :
    - - A SELECT clause
    - - A FROM clause
    - - A WHERE clause
- The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.
- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
- You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.
- A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.
- The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

You can use a subquery in a SELECT, INSERT, DELETE, or UPDATE statement to perform the following tasks:

- Compare an expression to the result of the query.
- Determine if an expression is included in the results of the query.
- Check whether the query selects any rows.

Syntax :

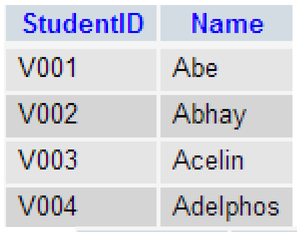


```
SELECT    select_list
FROM      table
WHERE     expr operator
                        (SELECT   select_list
                         FROM     table);
```

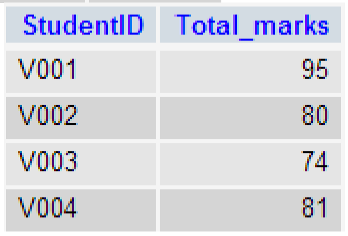
- The subquery (inner query) executes once before the main query (outer query) executes.
- The main query (outer query) use the subquery result.

SQL Subqueries Example :

In this section, you will learn the requirements of using subqueries. We have the following two tables 'student' and 'marks' with common field 'StudentID'.



StudentID	Name
V001	Abe
V002	Abhay
V003	Acelin
V004	Adelphos



StudentID	Total_marks
V001	95
V002	80
V003	74
V004	81

student marks

Now we want to write a query to identify all students who get better marks than that of the student who's StudentID is 'V002', but we do not know the marks of 'V002'.
- To solve the problem, we require two queries. One query returns the marks (stored in Total_marks field) of 'V002' and a second query identifies the students who get better marks than the result of the first query.

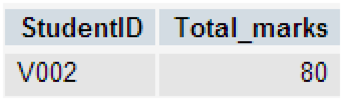
First query:

```
SELECT *
FROM `marks`
```

```
WHERE studentid = 'V002';
```

Copy

Query result:



StudentID	Total_marks
V002	80

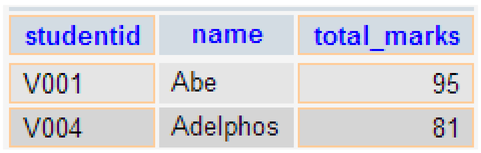
The result of the query is 80.
- Using the result of this query, here we have written another query to identify the students who get better marks than 80. Here is the query :

Second query:

```
SELECT a.studentid, a.name, b.total_marks

FROM student a, marks b

WHERE a.studentid = b.studentid

AND b.total_marks >80;
```

Copy

Query result:



studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

Above two queries identified students who get the better number than the student who's StudentID is 'V002' (Abhay).

You can combine the above two queries by placing one query inside the other. The subquery (also called the 'inner query') is the query inside the parentheses. See the following code and query result :

SQL Code:

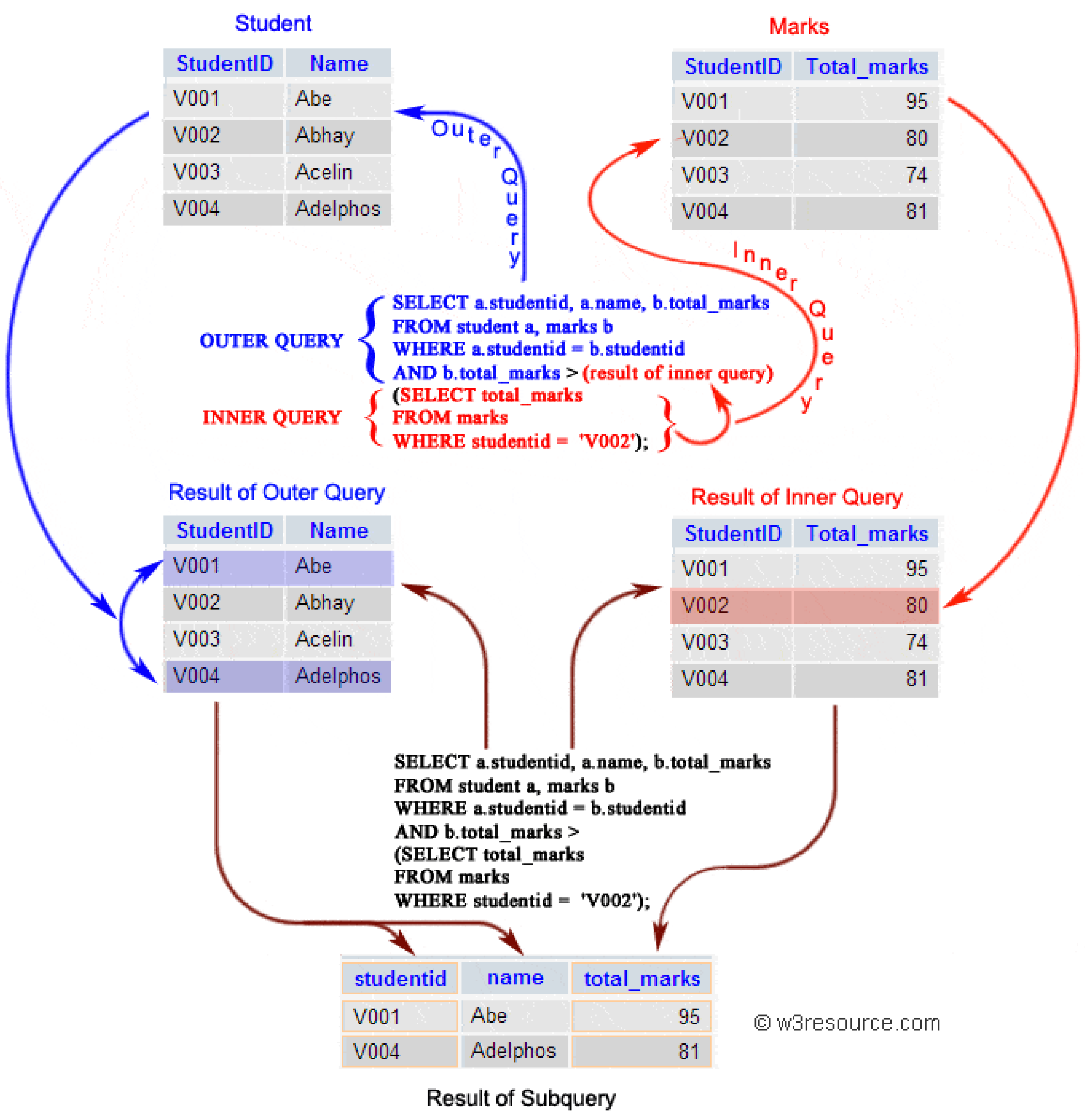
```
SELECT a.studentid, a.name, b.total_marks

FROM student a, marks b

WHERE a.studentid = b.studentid AND b.total_marks >

(SELECT total_marks

FROM marks
WHERE studentid =  'V002');
```

Copy

Query result:

studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

Pictorial Presentation of SQL Subquery:



Student

StudentID	Name
V001	Abe
V002	Abhay
V003	Acelin
V004	Adelphos

Marks

StudentID	Total_marks
V001	95
V002	80
V003	74
V004	81

```
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid
AND b.total_marks > (result of inner query)
(SELECT total_marks
FROM marks
WHERE studentid =  'V002');
```

Result of Outer Query

StudentID	Name
V001	Abe
V002	Abhay
V003	Acelin
V004	Adelphos

Result of Inner Query

StudentID	Total_marks
V001	95
V002	80
V003	74
V004	81

```
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid
AND b.total_marks >
(SELECT total_marks
FROM marks
WHERE studentid =  'V002');
```

studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

Subqueries: General Rules

A subquery SELECT statement is almost similar to the SELECT statement and it is used to begin a regular or outer query. Here is the syntax of a subquery:

Syntax:

```
(SELECT [DISTINCT] subquery_select_argument
FROM {table_name | view_name}
{table_name | view_name} ...
[WHERE search_conditions]
[GROUP BY aggregate_expression [, aggregate_expression] ...]
[HAVING search_conditions])
```

Subqueries: Guidelines

There are some guidelines to consider when using subqueries :

- A subquery must be enclosed in parentheses.
- A subquery must be placed on the right side of the comparison operator.
- Subqueries cannot manipulate their results internally, therefore ORDER BY clause cannot be added into a subquery. You can use an ORDER BY clause in the main SELECT statement (outer query) which will be the last clause.
- Use single-row operators with single-row subqueries.
- If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

Type of Subqueries

- Single row subquery : Returns zero or one row.
- Multiple row subquery : Returns one or more rows.
- Multiple column subqueries : Returns one or more columns.
- Correlated subqueries : Reference one or more columns in the outer SQL statement. The subquery is known as a correlated subquery because the subquery is related to the outer SQL statement.

- Nested subqueries : Subqueries are placed within another subquery.

In the next session, we have thoroughly discussed the above topics. Apart from the above type of subqueries, you can use a subquery inside INSERT, UPDATE and DELETE statement. Here is a brief discussion :

Roll No.	**Name of Student**	**Date of Performance**	**Date of Assessment**	**Grade**	**Sign of Student**	**Sign of Faculty**