Department of Information Technology

S.Y. BTech (IT)
SUB: DBMS LAB
Experiment No: 4

To study and implement Joins and Views

JOINS

By using joins, you can retrieve data from two or more tables based on logical relationships between the tables. Joins indicate how to use data from one table to select the rows in another table.

Joins can be specified in either the FROM or WHERE clauses. The join conditions combine with the WHERE and HAVING search conditions to control the rows that are selected from the base tables referenced in the FROM clause. Specifying the join conditions in the FROM clause, however, helps separate them from any other search conditions that might be specified in a WHERE clause and is the recommended method for specifying joins.

When joins are processed, the query engine chooses the most efficient method (from several possibilities) of processing the join. Although the physical execution of various joins uses many different optimizations, the logical sequence is as follows:

- The join conditions in the FROM clause are applied.
- The join conditions and search conditions from the WHERE clause are applied.
- The search conditions from the HAVING clause are applied.

This sequence can sometimes influence the result of the query if conditions are moved between the FROM and WHERE clauses.

Inner Joins

An inner join is a join in which the values in the columns being joined are compared through the use a comparison operator. In the SQL-92 standard, inner joins can be specified in either the FROM or WHERE clause

Syntax:

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

Outer Joins

Three types of outer joins: left, right, and full. All rows retrieved from the left table are referenced with a left outer join, and all rows from the right table are referenced in a right outer join. All rows from both tables are returned in a full outer join.

Using Left Outer Joins

A result set generated by a *SELECT* statement that includes a left outer join includes all rows from the table referenced to the left of LEFT OUTER JOIN. The only rows that are retrieved from the table to the right are those that meet the join condition.

Syntax:

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

Using Right Outer Joins

A result set generated by a *SELECT* statement that includes a right outer join includes all rows from the table referenced to the right of RIGHT OUTER JOIN. The only rows that are retrieved from the table to the left are those that meet the join condition.

Syntax:

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

Using Full Outer Joins

A result set generated by a *SELECT* statement that includes a full outer join includes all rows from both tables, regardless of whether the tables have a matching value (as defined in the join condition).

Syntax:

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

Views

A view can be thought of as either a virtual table or a stored query. The data accessible through a standard view is not stored in the database as a distinct object. What is stored in the database is a *SELECT* statement. The result set of the *SELECT* statement forms the virtual table returned by the view. A user can use this virtual table by referencing the view name in Transact-SQL statements in the same way a table is referenced.

There are no restrictions on querying through views and few restrictions on modifying data through them. In addition, a view can reference another view.
You can use a view to perform any or all of the following functions:

- Restricting a user to specific rows in a table
- Restricting a user to specific columns
- Joining columns from multiple tables so that they look like a single table
- Aggregating information instead of supplying details

Creating Standard Views
You can use the *CREATE VIEW* statement in to create a view.

Example:
Orders Table:
OrderID, CustomerID, EmployeeID, OrderDate, ShippedDate, ShipAddress

Customers Table:
CustomerID, CompanyName, ContactName, ContactTitle, Address, Phone

```
CREATE VIEW CustomerOrders
AS
SELECT o.OrderID, c.CompanyName, c.ContactName
FROM Orders o JOIN Customers c
ON o.CustomerID = c.CustomerID
```

Modifying Views
After a view is defined, you can change its name or modify its definition without dropping and recreating the view, which would cause it to lose its associated permissions.
When you rename a view, adhere to the following guidelines:

- The view to be renamed must be in the current database.
- The new name must follow the rules for identifiers.
- You can rename only views that you own.
- The database owner can change the name of any user’s view.

Altering a view does not affect any dependent objects (such as stored procedures or triggers) unless the definition of the view changes in such a way that the dependent object is no longer valid.
To modify a view, you can use *ALTER VIEW* statement to update the SELECT query, as shown in the following example:

```
ALTER VIEW CustomerOrders
AS
SELECT o.OrderID, o.OrderDate, c.CompanyName, c.ContactName
FROM Orders o JOIN Customers c
ON o.CustomerID = c.CustomerID
```

In this statement, you are modifying the select list so that it includes the order date. The SELECT query in the *ALTER VIEW* statement replaces the SELECT query that was defined in the original *CREATE VIEW* statement.

Deleting Views

After a view has been created, you can delete the view if it is not needed or if you want to clear the view definition and the permissions associated with it. When a view is deleted, the tables and the data upon which it is based are not affected. Any queries that use objects that depend on the deleted view fail when they are next executed (unless a view with the same name is created). If the new view does not reference objects expected by any objects dependent on the new view, however, queries using the dependent objects will fail when executed.

For example, suppose you create a view named MyView that retrieves all columns from the Authors table in the Pubs database, and this view is deleted and replaced by a new view named MyView that retrieves all columns from the Titles table instead. Any stored procedures that reference columns from the underlying Authors table in MyView now fail because those columns are replaced by columns from the Titles table instead.

To delete a view, you can use *DROP VIEW* statement as shown in the following example:

```
DROP VIEW CustomerOrders
```

Department of Information Technology

--EXERCISE--

Implement the SQL statements for the following questions

1. Display movie no, title, type ,cust_id, issue date and return date using INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN and FULL OUTER JOIN in the descending order of movie no.
2. Find the lname,fname who have been issued moves
3. Find out the title and type of movies that have been issued to Tina.
4. Display the first names and last names of the customers who have issued movies after 23rd July 95.
5. Find the customer name and area with invoice number 'I10'.
6. Find the names and movie numbers of all the customers who have been issued a movie.
7. Find out which customers have been issued movie number 9.
8. Find the name of the movie issued to Tina and Allan.
9. For the above query create a view.
10. Modify the above view to add the price of the movie.
11. Drop the view.