Cairo University
Faculty of Computers and Artificial Intelligence
Information Systems Department

Database I, Year 2022/ 2023
Lab - 4
DML 3 (Aggregate/group By Functions)
DML 4 (Subqueries)

Part (1) (set operations, Aggregations, group by and having)

Set operations: allow the results of multiple queries to be combined into a single result set.[1] Set operators include `UNION`, `INTERSECT`, and `EXCEPT`.

1- **UNION OPERATOR:** combines the results of two SQL queries into a single table of all matching rows. The two queries must result in the same number of columns and compatible data types in order to unite. Any duplicate records are automatically removed unless `UNION ALL` is used.

2- **INTERSECT OPERATOR:** takes the results of two queries and returns only rows that appear in both result sets. The `INTERSECT` operator removes duplicate rows from the final result set. The `INTERSECT ALL` operator does not remove duplicate rows from the final result set.

3- **EXCEPT OPERATOR:** takes the distinct rows of one query and returns the rows that do not appear in a second result set. The `EXCEPT ALL` operator does not remove duplicates.

Aggregate functions are used to summarize information from multiple tuples into a single-tuple summary.
Grouping is used to create subgroups of tuples before summarization

Example1:
 	
Get the customers IDS who didn't make any order.
	`SELECT CustomerID FROM Customers EXCEPT`
	`SELECT CustomerID FROM Orders`

Example 2:
 	
Get all names of suppliers who supply products of category named 'Beverages' and products of category named 'Condiments'
	`SELECT Suppliers.CompanyName`
	
	`FROM Products, Suppliers, Categories`
	
	`WHERE Suppliers.SupplierID = Products.SupplierID`
	
	`AND Products.CategoryID =Categories.CategoryID`

	`AND Categories.CategoryName ='Beverages'`
	
	`INTERSECT`
	
	`SELECT Suppliers.CompanyName`
	
	`FROM Products, Suppliers, Categories`
	
	`WHERE Suppliers.SupplierID =Products.SupplierID`
	
	`AND Products.CategoryID =Categories.CategoryID`
	
	`AND Categories.CategoryName ='Condiments'`

Example 3:
 	
For a mass-mailing purpose, get the list of all names, addresses, cities and postal codes known to the database and remove duplicates from the result if any (these data are stored in the customers and employees tables).
	`SELECT ContactName AS Name,Address, City,PostalCode`
	
	`FROM Customers`
	
	`UNION`
	
	`SELECT FirstName+' '+LastName AS Name, Address, City,PostalCode`
	
	`FROM Employees`

Example 4:
 	
For each product ordered in March 1998, get the product id, number of orders for that product and the total quantity ordered. Give suitable name(s) for any unnamed displayed column(s). Sort by total quantity descendingly
	`SELECT ProductID, count(od.orderid) as numOfOrders, Sum(quantity) as totalQuantityOrdered`
	`FROM [Order Details] od, orders o`
	`where od.OrderID = o.OrderID`
	`and year(OrderDate) ='1998'`
	`and month(OrderDate) = '3'`
	`GROUP BY ProductID`
	`Order by TotalQuantityOrdered desc`
	
	Comments:
	
	1. All the columns in the select with the aggregate function should be included in the group by clause. In the example, the GROUP BY will give you one value (the numer of orders) for each ProductID.
	2. In this query, if we have a column next to the aggregate function in the select caluse, we have to write a GROUP BY this column. Otherwise, an error will be given because we are trying to tell the DBMS to show ProductID "Multipe Values" next to an aggregate function.
	3. If we removed the other column(s) in the select and the

	GROUP BY and keep only the aggregate functions. The SUM/COUNT will get us the sum of quantities and total number of orders for the whole table at once.
Example 5: **Get the supplier names (Contact names) and the total number of supplied products for each supplier, for only those suppliers that supplied more than 3 products. Give suitable name(s) for any unnamed displayed column(s).**	`SELECT ContactName, Count (ProductID) AS NumberOfSuppliedProducts` `FROM Products, Suppliers` `where Products.SupplierID = Suppliers.SupplierID` `GROUP BY ContactName` `HAVING Count (ProductID) > 3` **Comments:** 1. HAVING, this clause could be written only in case you have a GROUP BY clause. It is a filtering condition that will be applied on the resultant groups, to filter the groups which do not satisfy the given condition. 2. Conditions which are applied to the data before grouping (the raw data not the aggregated groups) , should be put in the where clause not the HAVING.
Example 6: **For all customers with at least 15 orders after 1996, get the customer id, total number of orders, the last order date and the average unit price for products in his/her orders . Give suitable name(s) for any unnamed displayed column(s). Display the result starting from customers with the highest number of orders.**	`SELECT CustomerID, COUNT (o.OrderID) AS NumberOfOrders, MAX (OrderDate) AS LastOrderDate, Avg(Unitprice) as AveragePrice` `FROM Orders o, [order details] od` `WHERE o.OrderID = od.orderid` `and YEAR (OrderDate) > 1996` `GROUP BY CustomerID` `HAVING COUNT (o.OrderID) > 15` `and Avg(unitPrice)> 25` `ORDER BY COUNT (o.OrderID) DESC`

Part (2) (Nested Queries)

A Subquery is a query within another query. Subqueries can reside in WHERE Clause, FROM Clause or SELECT Clause.

1. Using Sub-queries in the where clause – IN

1. **Display the first name, title of all employees who live in UK and have the same job (title) as Janet or Steven. Sort the result by the first name of the employees in a descending order.**

```
SELECT FirstName, Title
FROM Employees
WHERE Country = 'UK'
AND Title IN (SELECT Title
              FROM Employees
              WHERE FirstName IN ('Janet', 'Steven'))
ORDER BY FirstName DESC
```

2. **Get a list of orders (with order dates) that were processed by employees who are not sales representatives:**

```
SELECT [OrderID], [OrderDate]
FROM [Orders]
WHERE [EmployeeID] IN
(SELECT [EmployeeID] FROM [Employees]
 WHERE [Title]<>'Sales Representative')
```

2. Using Sub-queries in the where clause –Exists

1. **Return a list of customers who has orders shipped to UK**

```
select CustomerID, CompanyName
from customers as a
where exists
(
  select * from orders as b
  where a.CustomerID = b.CustomerID
  and ShipCountry = 'UK'
)
```

OR using join

```
select CustomerID, CompanyName
from customers, orders
where CustomerID = CustomerID
and ShipCountry = 'UK'
```

3. Using Sub-queries in the where clause –Not Exists

1. **Get the distinct names for products that are always ordered with quantity greater than 10.**

```
SELECT DISTINCT ProductName
FROM [Products]
WHERE NOT EXISTS
(SELECT * FROM [Order Details]
 WHERE [Order Details].[ProductID]=[Products].[ProductID]
   and QUANTITY<10);
```

2. **Get the customer name and title for customers that have not placed any orders.**

```
SELECT ContactName,ContactTitle
FROM Customers AS c
WHERE Not EXISTS(SELECT *
      FROM [Orders] AS o
      WHERE o.CustomerID = c.CustomerID)
```

Note:

- When using subquery in the where clause , take care when using operators; If the subquery returns more than one value you can't use operator such as = , but you may use the IN operator.
- (IN) predicate can either return TRUE, FALSE or NULL :
  - TRUE is returned when the required value is NOT NULL and could be found.
  - FALSE is returned when the required value is NOT NULL and is NOT found and the list DOES NOT contain NULL values
  - NULL is returned when the required value is NULL, or when the required values is NOT NULL and was not found in the list and the list contains at least one NULL value
- EXISTS condition is considered "to be met" if the sub query returns at least one row, it always returns TRUE or FALS
  - TRUE as soon as it finds only a single matching row in the sub query
  - FALSE, if it find none.
- NOT EXISTS will return TRUE only if no row is returned from the sub query.

4. **Using Sub-queries in the where clause – “All”**

   1. **Get the last name, all emplyees who were hired before all emplyees working as ‘Sales Managers’ and ‘Inside Sales Coordinators’.**

```
SELECT LastName
FROM Employees
where HireDate < ALL (SELECT HireDate
            FROM Employees
                    WHERE Title IN ('Sales Manager', 'Inside Sales Coordinator'))
```

5. **Using Subquery in FROM Clause “Represents a table”**

   1. **Get the supplier name (contact name) and the supplied product name for products having unit price less than 7. Fully optimize your query to minimize the cost of accessing any unneeded columns and/or rows from the relevant tables.**

```
SELECT S.ContactName, P.ProductName
FROM (SELECT ContactName,SupplierID FROM Suppliers) AS S ,
     (SELECT ProductName, SupplierID, UnitPrice FROM Products WHERE
     UnitPrice < 7) AS P
Where S.SupplierID = P.SupplierID
```

Note:

- Mainly we use this way to shrink the size of the table and decrease the cost of working with the whole table (considering all columns in it). So, we choose only the set of columns we need from the table(s) related to our query.
- As shown in the above example, we can give an alias to the subquery and use it in the main query.