CSIS05I

Database Systems II

Lab (2)

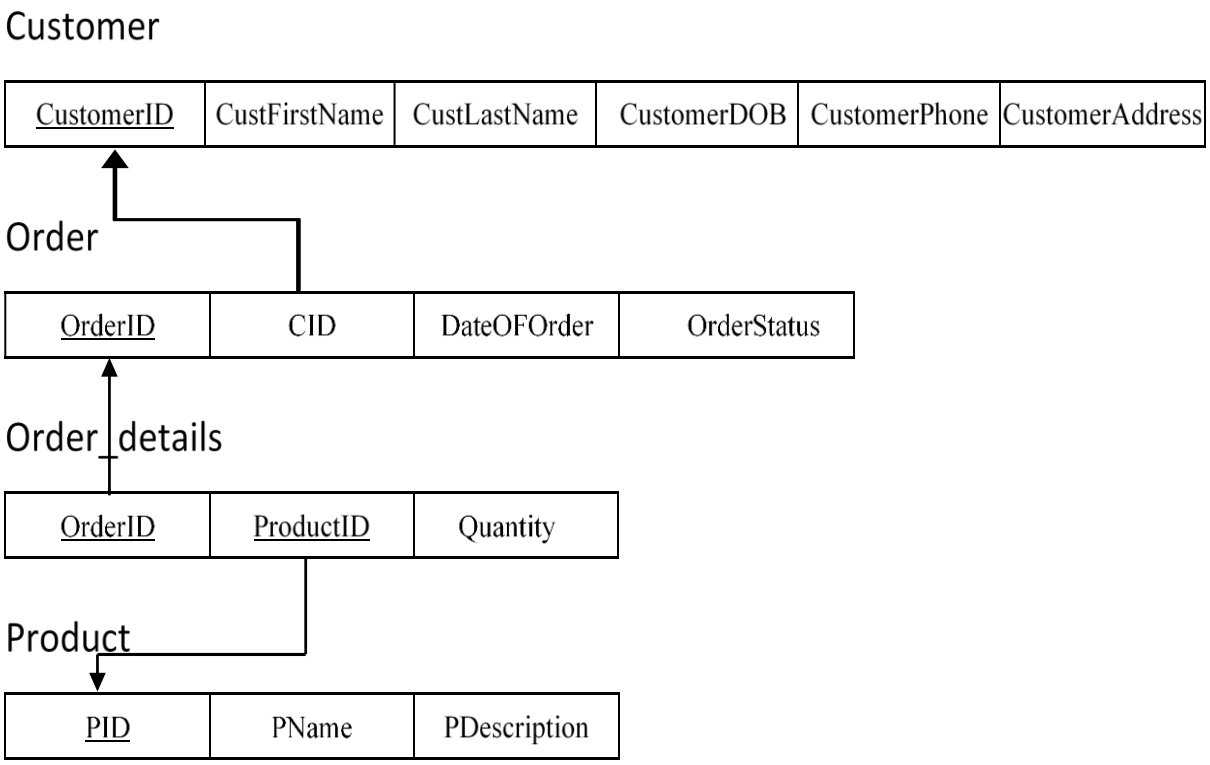
Views
Create, Alter, Drop, Rename, Update Views and Indexed Views

Lab (2)

The Overview

This lab we will discuss a new concept which is view. The view is a virtual table based on the result-set of an SQL statement. But before explaining the views, we will discuss the domain integrity (default and check constraints).

Store Schema:



Last time we created table customer and table order. This time we will create table product and table order_details and add the domain integrity constraints as well:

- Default constraint:

```
CREATE TABLE Product
(
PID             INT                 NOT NULL,
PName           VARCHAR(30),
PDescription    VARCHAR(30)     default   'NA' ,

CONSTRAINT Product_pk PRIMARY KEY (PID)
);

INSERT INTO Product VALUES ('201', 'TV', default);
```

Another way to create the default constraint is to make it using the Alter command:

```
ALTER TABLE Product
ADD CONSTRAINT Pdesc_default
DEFAULT 'nothing' FOR PDescription;
```

Note:

- You have to use only one of the methods above when adding a default constraint to the same attribute.
- You can make more than one default constraint to different attributes in the same table.

To drop the default constraint:

```
ALTER TABLE Product
drop constraint Pdesc_default
```

- Check constraint:

```
CREATE TABLE Order_Details
(
OrderID      INT        NOT NULL,
ProductID    INT        NOT NULL,
Quantity     INT        check (Quantity > 3),

CONSTRAINT Orderdetails_pk PRIMARY KEY (OrderID, ProductID),
CONSTRAINT Order_Product_fk FOREIGN KEY (OrderID) REFERENCES Orderr (OrderID),
CONSTRAINT Product_Order_fk FOREIGN KEY (ProductID) REFERENCES Product (PID)
);
```

Another example for check condition can be:

```
Quantity     INT        check (Quantity between 1 and 3)
```

Another way to create the check constraint is to make it using the Alter command:

```
ALTER TABLE Order_details
ADD CONSTRAINT quantity_check CHECK (Quantity > 3);
```

To drop the check constraint:

```
ALTER TABLE Order_details
DROP CONSTRAINT quantity_check;
```

Creating views

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.

Advantage of views:

- **Views** can join and simplify multiple tables into a single virtual table.
- **Views** can hide the complexity of data.
- **Views** can hide confidential data to preserve a certain security level.
- **Views** can be used as security mechanisms by letting users access data through the view, without granting the users permissions to directly access the underlying base tables of the view.

Creating view syntax

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

OR

```
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.
- ✓ Views do NOT store data like tables.

Example 1:

Creates a view that shows all customers names that live in New Cairo.

```
CREATE VIEW [Cairo Customers] AS
SELECT CustFirstName, CustLastName
FROM Customer
WHERE  CustomerAddress = 'New Cairo';
```

We can query the view above as follows (call it)

```
SELECT * FROM [Cairo Customers];
```

Example 2:

Creates a view that selects every product ID in the "Order_Details" table with quantity higher than 5.

```
CREATE VIEW [Products with high
quantity] AS
  SELECT ProductID, Quantity
  FROM Order_Details
  WHERE Quantity > 5;
```

We can query the view above as follows

```
SELECT * FROM [Products with high quantity];
```

Example 3:

Assume that we want to view all the information about the Customer including all his information that is stored in the database.

```
CREATE VIEW CustomerInfo AS
SELECT *
FROM Customer;
```

We can query the view above as follows

```
SELECT * FROM CustomerInfo;
```

Updating a view

A view can be updated with the ALTER VIEW command or CREATE OR ALTER VIEW command.

- ✓ Note: A view can't be modified and re-executed after creation unlike queries, so you have to alter them.

Alter View syntax

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

We can query the view above as follows

```
SELECT column1, column2, ...
FROM Viewname;
```

Example 4:

Update the view by changing in the result query as the view will search for New Jersy instead of New Cairo.

```
ALTER VIEW [Cairo Customers] AS
SELECT CustFirstName, CustLastName,
CustomerAddress
FROM Customer
WHERE CustomerAddress = 'New Jersy';
```

Create OR Alter View syntax

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

Example 5:

Alter a view in which the First name of the customer has ‘ar’ in it.

```
CREATE OR ALTER VIEW CustomerName AS
SELECT CustomerID, CustFirstName, CustLastName
FROM Customer
WHERE CustFirstName like '%ar%';
```

We can query the view above as follows

```
SELECT * FROM CustomerName;
```

In order to dispose of a view, we use the same command as that of disposing of a relation:

```
DROP VIEW view_name;
```

Or

```
DROP VIEW [view name];
```

Example 6:

Drop from the database view "CustomerName".

```
DROP VIEW CustomerName;
```

Renaming views

In SQL, Views are renamed using the sp_rename system stored procedure. By definition, this SP is used for changing the name of a user-created object in the current database.

Sp_rename:

- System Stored Procedure.
- Changes the name of an object.
- Can rename COLUMN, TABLE, DATABASE, INDEX, CONSTRAINT.
- **Note**: Warning will be displayed.

  Note: if the rename is not updated you can go to Edit then select IntelliSense then RefreshLocal Cache.

Renaming view syntax

```
sp_rename
'Old_View',
'New_view';
```

Example 7:

Rename the view from Cairo Customers to Jersy Customers.

```
sp_rename
'[Cairo Customers]',
'[Jersy Customers]';
```

Data modifications through views:

SQL enables records insertion, update and deletion into the database tables using views.

Insert view syntax

```
INSERT INTO VIEW_NAME
SELECT 'COLUMN1',
'COLUMN2' ..... ;
```

Or

```
INSERT INTO VIEW_NAME
VALUES ('COLUMN1',
'COLUMN2' ..... );
```

Example 8:

Create a product View to display all the products in the store, and then Modify through the product view to insert a new record in the Product which has a Product ID equals 204 and a Product Name called ‘Flash Memory’.

```
Create View product_view AS
Select PID, PName
From Product;

Insert Into product_view Select  204, 'Flash Memory';

Select * From product_view;
```

Using with check option in views

- The purpose of with check option is to ensure that all update and insert satisfy the condition in the view definition otherwise it will return an error.

Example 9:

```
ALTER VIEW product_view AS
SELECT PID, PName
From Product
WHERE PName LIKE '%me%' WITH CHECK OPTION;

INSERT INTO product_view
Values(205, 'Remote Control');
```

Error Message: *The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint. The statement has been terminated.*

```
INSERT INTO product_view
Values(205, 'Digital Camera');
```

Result: *1 rows(s) affected.*

Indexed Views

Non indexed view is just viewing a result set of SQL query when we try to retrieve data from the view the data is actually retrieved from the underlying base tables as views are virtual tables that do not store any data. When an index iscreated on a view it gets materialized which means that the view is now capable of storing data.

Note:

- Only unique clustered index can be created on views.

Guidelines for creating indexed views:

- Views should be created with schema binding option.
- The base tables in the view should be referenced with two-part name.
- If GROUP BY is specified in a view then the view select list must contain COUNT_BIG(*) expressionas COUNT function will not work.

Example 10:

Create a view named OrderStatusView where the status is delivered and then create an index on this view.

```
Create View OrderStatusView

With Schemabinding As

Select OrderID, OrderStatus

From dbo.Orderr

Where OrderStatus ='delivered';


CREATE UNIQUE CLUSTERED INDEX Idx_Status
ON OrderStatusView (OrderID, OrderStatus);
```

Exercises

1. Create the following views in SQL on the Store database schema:
   a) A view that has the customer name and Customer Address, and its DOB is equal to '14/05/1990'.
   b) A view that has the Customer name and order ID in which the order must be delivered.
   c) A view that has the customer ID, and order date and its status with the product name and its description.
2. Drop view that contains order ID and product name (view that you created in question 1(C)).
3. Create a view that has all the customer data, and name it VCustomerAllData, where the customer's first name is 'Mark'.
4. Rename the VCustomerAllData to CustomerView.
5. Create a view that has the product name and product description then name itVProductDetails, where the product description starts with 'S'.
6. Alter the product's view to have with check options.
7. Insert data into the product table using the view.
8. Retrieve the data from the product's view.
9. Create a view that contains the CustomerID, customerFirstName and DateOfOrder then name it VCustomerOrderDate.
10. Create an index on the VCustomerOrderDate view.
11. An extra question for you will be found in the solution pdf.