CSC453/CSC553: Team Project 2

Database Implementation

TEAM 3: JANANI,NOLAN,VAIBHAV,ISHU,JHONG

DATABASE DESCRIPTION

The following is a report on the implementation of a database design for Marcia's Dry Cleaning. The database consists of four tables, which are utilized to track and manage customer accounts, orders, and the various services offered by the business. The database consists of the following four tables:

CUSTOMER (CustomerID, FirstName, LastName,Phone,Email)

INVOICE (InvoiceNumber, CustomerID, DateIn, DateOut, Subtotal, Tax, TotalAmount)

INVOICE_ITEM (InvoiceNumber, ItemNumber, ServiceID, Quantity, UnitPrice, ExtendedPrice)

SERVICE (ServiceID, ServiceDescription, UnitPrice)

**Primary keys for each table are underlined.*

DATABASE TABLE CONSTRAINTS

The following diagrams outline the data constraints for each table fields as well as all primary, foreign, and candidate keys present within each table.

CUSTOMER

PK	**CustomerID: NOT NULL**
	FirstName: NOT NULL(AK1.1)
	LastName: NOT NULL(AK1.2)
	Phone: NULL
	Email: NOT NULL(AK2)

INVOICE

PK	**InvoiceNumber: NOT NULL**
	CustomerID: NOT NULL (FK)
	DateIn: NOT NULL
	DateOut: NULL

	Subtotal: NOT NULL
	Tax: NOT NULL
	TotalAmount: NOT NULL

INVOICE_ITEM

PK	**InvoiceNumber: NOT NULL (AK1.1)**
PK	**ItemNumber: NOT NULL**
	ServiceID: NOT NULL (FK) (AK1.2)
	Quantity: NOT NULL
	UnitPrice: NOT NULL
	ExtendedPrice: NOT NULL

SERVICE

PK	**ServiceID: NOT NULL**
	ServiceDescription: NOT NULL (AK1)
	UnitPrice: NOT NULL

RELATIONSHIP CARDINALITY TABLE

The relationships that exist between the parent-child entities are described below.

RELATIONSHIP			CARDINALITY [Blue = Inferable]	
PARENT	CHILD	TYPE	MAX	MIN
CUSTOMER	INVOICE	Strong	1:N	M-O

INVOICE	INVOICE_ITEM	ID dependent weak entity	1:N	M-M
SERVICE	INVOICE_ITEM	strong	1:1	M-O

RELATIONSHIP CARDINALITY ENFORCEMENT

Because there are a number of relationships with minimum cardinalities of 1 within the database, referential integrity between required parents and children must be enforced. The following tables describe how minimum cardinality relationships will be enforced within the database.

M-O RELATIONSIHP ENFORCEMENT

Parent Required	Action on Parent	Action on Child
Insert	None	Get a parent
Modify Key or Foreign Key	Make corresponding modification on corresponding children(cascade update)	Ok, if new foreign key matches an existing parent
Delete	Delete children related to the parent (cascade delete)	None.

M-M RELATIONSHIP ENFORCEMENT

Both parent and child required	Action on Parent	Action on Child
Insert	Insert trigger to be used. Parents' first record to be inserted only after the insertion of first instance in the child	Insert trigger to be used. The insertion of first instance corresponding to a foreign key happens without an existing parent but following insertions related to that key should happen only after parent is created
Modify Key or Foreign Key	Make corresponding modification on corresponding	Prohibit if changing key for the last child of the parent. Otherwise, OK if new

	children(cascade update)	foreign key matches an existing parent.
Delete	Delete children related to the parent (cascade delete)	Delete trigger to be used. No action until last child.For the last child, either prohibit or delete the parent record corresponding to the key.

DATABASE CREATION WITH SQL

Following establishment of the database table and relationship constraints, the database design can be implemented with SQL. The statements below illustrate the creation of the SQL database tables. The relationships, cardinalities, and enforcement policies described above are implemented. Note that in addition to relationship constraints, the UnitPrice field of the SERVICE table is also limited via a constraint to values in the range 1.50 to 10.00. The following implementation of the Marcia's Dry Cleaning is written for SQL Server 2008:

CUSTOMER TABLE CREATION

```
CREATE TABLE CUSTOMER (

 CustomerID int IDENTITY(100,5) NOT NULL,

 FirstName VARCHAR(50) NOT NULL,

 LastName VARCHAR(50) NOT NULL,

 Phone CHAR(10) NOT NULL,

 Email VARCHAR(100)

CONSTRAINT AK_Alternate Unique (FirstName,LastName,Email)

CONSTRAINT PK_CustomerID PRIMARY KEY (CustomerID)

);
```

INVOICE TABLE CREATION

```
CREATE TABLE INVOICE (

 InvoiceNumber INT PRIMARY KEY,

 CustomerID INT FOREIGN KEY REFERENCES CUSTOMER(CustomerID),

 DateIn DATE NOT NULL,

 DateOut DATE NOT NULL,
 Subtotal MONEY NOT NULL,
 Tax MONEY NOT NULL,
 TotalAmount MONEY NOT NULL
 CONSTRAINT AK_Invoice Unique (InvoiceNumber,ServiceID),
 CONSTRAINT FK_CustomerID FOREIGN KEY (CustomerID)
  REFERENCES CUSTOMER(CustomerID)
   ON DELETE CASCADE
   ON UPDATE CASCADE
);
```

SERVICE TABLE CREATION

```
   CREATE TABLE SERVICE(
    ServiceID INT NOT NULL PRIMARY KEY,
    ServiceDescription TEXT NOT NULL,
    UnitPrice MONEY NOT NULL
    CONSTRAINT AK_Invoice Unique (ServiceDescription),
    CONSTRAINT Check_UnitPrice
      CHECK (UnitPrice> 1.50 AND UnitPrice<10.00)
   );
```

INVOICE_ITEM TABLE CREATION

```
  CREATE TABLE INVOICE_ITEM (
   InvoiceNumber INT NOT NULL,
   ItemNumber INT NOT NULL,
   ServiceID INT NOT NULL,
   Quantity INT DEFAULT '1',
   UnitPrice MONEY NOT NULL,
   ExtendedPrice MONEY NOT NULL,
   CONSTRAINT pk_id PRIMARY KEY(InvoiceNumber,ItemNumber),
   CONSTRAINT fk_id FOREIGN KEY(InvoiceNumber)
     REFERENCES  INVOICE(InvoiceNumber)
       ON DELETE CASCADE
```

```
        ON UPDATE CASCADE,
    CONSTRAINT fk_id1 FOREIGN KEY(ServiceID)
      REFERENCES SERVICE(ServiceID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
    );
```

```
INVOICE_ITEM :
    InvoiceNumber must belong to INVOICE upon insertion.Prohibit update of
         InvoiceNumber . Deletion is allowed until last item, if last item
         delete the record corresponding to that InvoiceNumber on INVOICE.
```

DATA CONSTRAINT ENFORCEMENT

The table SERVICE is a parent of INVOICE_ITEM where every INVOICE_ITEM must contain a ServiceID but it is not necessary that every ServiceID must belong to an INVOICE_ITEM instance. Thus, whenever there is a deletion or update on a ServiceID, the INVOICE_ITEM corresponding to that ServiceID has to be updated or deleted.

Eforcement of relationships requiring children is performed using triggers. Triggers will check if entries for INVOICE_ITEM.ServiceID and SERVICE.ServiceID match. If they match, then the UnitPrice from the table SERVICE is extracted and saved in INVOICE_ITEM table. The implementation of this trigger is illustrated below under the section header ENTRY OF DATA INTO DATABASE TABLES.

ENTRY OF DATA INTO DATABASE TABLES

POPULATING CUSTOMER TABLE:

```
INSERT INTO CUSTOMER values('Tom','Nelson','4522214321','tom@abc.com');

INSERT INTO CUSTOMER values('Venica','Aldrin','3102426732','valdrin@abc.com');

INSERT INTO CUSTOMER values('Ranjeet','Singh','5620916543','ransingh@efg.com');

select * from CUSTOMER
```

```
CustomerID      FirstName    LastName     Phone        Email

110             Tom          Nelson       4522214321   tom@abc.com

115             Venica       Aldrin       3102426732   valdrin@abc.com

120             Ranjeet      Singh        5620916543   ransingh@efg.com
```

POPULATING SERVICE TABLE :

```
INSERT INTO SERVICE VALUES ('1100','MENS SHIRT','9.99');
INSERT INTO SERVICE VALUES ('1101','WOMENS SHIRT','9.99');
INSERT INTO SERVICE VALUES ('1102','MENS PANTS','7.99');
INSERT INTO SERVICE VALUES ('1103','WOMENS PANTS','7.99');
INSERT INTO SERVICE VALUES ('1104','WOMENS DRESS','5.99');
```

**Verification of successful value insertion is illustrated with the query below*

VERIFICATION OF SUCCESSFUL ENTRY INSERTION VIA A QUERY

```
SELECT * FROM SERVICE;
ServiceID  ServiceDescription   UnitPrice
1100       MENS SHIRT           9.99
1101       WOMENS SHIRT         9.99
1102       MENS PANTS           7.99
1103       WOMENS PANTS         7.99
1104       WOMENS DRESS         5.99
```

POPULATING INVOICE_ITEM TABLE

An example insert statement for INVOICE_ITEM table is provided. Similar statements were used to populate the additional table contents shown below

EXAMPLE TABLE ENTRY STATEMENT

```
INSERT INTO INVOICE_ITEM(InvoiceNumber,ItemNumber,ServiceID,Quantity)
      values('1013','3','1104','2')
```

INVOICE_ITEM TABLE ILLUSTRATION

InvoiceNumber	ItemNumber	ServiceID	Quantity	UnitPrice	ExtendedPrice
1011	1	1101	3	9.99	29.97
1011	2	1103	2	7.99	15.98
1012	2	1103	4	7.99	31.96
1013	1	1100	2	9.99	19.98
1013	2	1101	1	9.99	9.99
1013	3	1104	2	5.99	11.98

POPULATING INVOICE VIA INVOICE_ITEM INSERT TRIGGER

In this project, since there is no front end application which supplies the value for customerid , datein and dateout fields, CustomerID is generated using RAND() , DateIn is set as the current day's date using GetDate() and DateOut is computed by adding 5 business days to DateIn. Other values like InvoiceNumber is set by the insert statement insde INSERT TRIGGER of INVOICE_ITEM and Subtotal,Tax and TotalAmount are calculated accordingly.

InvoiceNumber	CustomerID	DateIn	DateOut	Subtotal	Tax	TotalAmount
1011	120	2016-11-11	2016-11-16	45.95	4.595	50.545
1012	120	2016-11-11	2016-11-16	31.96	3.196	35.156
1013	110	2016-11-11	2016-11-16	41.95	4.195	46.145

UPDATING TABLE ENTRIES

The following sections illustrate changes made to the values stored in the created the tables, as well as changes to the tables themselves.

UPDATING VALUES IN SERVICE TABLE

The following statements are used to update all occurances of "Mens Shirt" in the services table to "Mens' Shirts" in order to illustrate how entry data may be corrected. The change is then verified with a query of the service table.

```
ALTER TABLE SERVICE

ALTER COLUMN ServiceDescription Varchar(100);

UPDATE SERVICE

    SET ServiceDescription='Mens'' Shirts'

    WHERE ServiceDescription='Mens Shirt'; ;


SELECT * FROM SERVICE;
```

ServiceID	ServiceDescription	UnitPrice
1100	Mens' Shirts	9.99
1101	WOMENS SHIRT	9.99
1102	MENS PANTS	7.99
1103	WOMENS PANTS	7.99
1104	WOMENS DRESS	5.99

DELETING ENTRIES FROM TABLE INVOICE

The following illustrates deletion of entries from the table INVOICE. Due to a constraint requiring a cascade deletion of related entries in the table INVOICE_ITEM, a deletion request triggers a cascade deletion of child instances in INVOICE_ITEM.

```
DELETE FROM INVOICE
      WHERE InvoiceNumber='1000';
--Automatically deletes all of its invoice items too since cascade
    delete was enabled
```

CREATION OF VIEWS

OrderSummaryView

The following illustrates creation of a view containing the following fields:

INVOICE.InvoiceNumber, INVOICE.DateIn, INVOICE.DateOut,
INVOICE_ITEM.ItemNumber, INVOICE_ITEM. Service,and
INVOICE_ITEM.ExtendedPrice.

```
CREATE VIEW OrderSummaryView
AS (SELECT InvoiceNumber, DateIn,DateOut, ItemNumber, ServiceID,
    ExtendedPrice
     FROM INVOICE INNER JOIN INVOICE_ITEM
     ON INVOICE.InvoiceNumber=INVOICE_ITEM.InvoiceNumber);
```

The following illustrate the contents of the tables INVOICE and INVOICE_ITEM

INVOICE:

InvoiceNumber	CustomerID	DateIn	DateOut	Subtotal	Tax	TotalAmount
1011	115	2016-11-12	2016-11-17	11.98	1.198	13.178
1013	120	2016-11-12	2016-11-17	11.98	1.198	13.178

INVOICE_ITEM:

InvoiceNumber	ItemNumber	ServiceID	Quantity	UnitPrice	ExtendedPrice
1011	3	1104	2	5.99	11.98
1013	3	1104	2	5.99	11.98

The following illustrates the view OrderSummaryView for comparison to INVOICE and INVOICE_ITEM

```
SELECT * FROM OrderSummaryView
```

```
InvoiceNumber      DateIn  DateOut        ItemNumber    ServiceID      ExtendedPrice

1011  2016-11-12   2016-11-17     3        1104    11.98

1013  2016-11-12   2016-11-17     3        1104    11.98
```

OrderSummaryView

The view OrderSummaryView contains the following fields

INVOICE.InvoiceNumber,CUSTOMER.FirstName, CUSTOMER.LastName, CUSTOMER.Phone, INVOICE.DateIn,INVOICE.DateOut, INVOICE.SubTotal, INVOICE_ITEM.ItemNumber, INVOICE_ITEM.Service,and INVOICE_ITEM.ExtendedPrice

```
CREATE VIEW CustomerOrderSummaryView

AS (SELECT INVOICE.InvoiceNumber, FirstName, LastName, DateIn,
    DateOut, Subtotal, ItemNumber, ServiceID, ExtendedPrice

      FROM INVOICE

         INNER JOIN CUSTOMER

         ON CUSTOMER.CustomerID=INVOICE.CustomerID

               INNER JOIN INVOICE_ITEM

               ON INVOICE.InvoiceNumber=INVOICE_ITEM.InvoiceNumber);
```

```
InvoiceNumber      FirstName       LastName       DateIn  DateOut        Subtotal
      ItemNumber   ServiceID       ExtendedPrice

1011  Venica       Aldrin  2016-11-12     2016-11-17     11.98    3       1104     11.98

1013  Ranjeet      Singh   2016-11-12     2016-11-17     11.98    3       1104     11.98
```

CustomerOrderHistoryView

The following illustrates creation of a view referencing another view. The created view contains the following fields: INVOICE_ITEM.ItemNumber and INVOICE_ITEM.Service. The view groups orders by CUSTOMER.LastName, CUSTOMER.FirstName and INVOICE.InvoiceNumber in that order. Finally the view sums and averages INVOICE_ITEM.ExtendedPrice for each order for each customer.

```
SELECT InvoiceNumber, FirstName, LastName, DateIn, DateOut,
    Subtotal, SUM(ExtendedPrice) AS TotalPrice,
    AVG(ExtendedPrice)AS AverageExtendedPrice

FROM CustomerOrderSummaryView

GROUP BY InvoiceNumber, FirstName, LastName, DateIn, DateOut,
    Subtotal;
```

CustomerOrderCheckView

The following illustrates creation of the view CustomerOrderCheckView, which uses CustomerOrderHistoryView and shows any customers for whom the sum of INVOICE_ITEM.ExtendedPrice is not equal to INVOICE.Subtotal.

```
CREATE VIEW CustomerOrderCheckView AS

SELECT FirstName, LastName, SUM(Subtotal) AS SubtotalSum,
    SUM(TotalPrice)AS ExtendedPriceSum

FROM CsutomerOrderHistoryView

GROUP BY FirstName,LastName

HAVING SUM(Subtotal)<> SUM(TotalPrice);
```

REVIEW OF TRIGGER REQUIREMENTS

Relationship between CUSTOMER AND INVOICE tables:

This a M-O Relationship. Placing CustomerID, which is the primary key of CUSTOMER as foreign key of INVOICE ensures the referential integrity constraint upon INSERTION,MODIFICATION or DELETION.

The same rules apply for maintaining RIC in SERVICE AND INVOICE_ITEM tables too.

Relationship between INVOICE AND INVOICE_ITEM tables:

Triggers are required only for populating INVOICE_ITEM and INVOICE tables because they exhibit M-M relationship. Every Invoice should have at least one item belonging to the invoice and every item of an invoice should exactly belong to that invoice only. The plan to implement this minimum cardinality constraint in our design is as below

INSERT TRIGGER:

InvoiceNumber is the primary key of INVOICE which is placed as the foreign key in INVOICE_ITEM. In the INSERT TRIGGER of INVOICE_ITEM, before entering the first record for a particular InvoiceNumber, add a record pertaining to that InvoiceNumber on INVOICE. For subsequent insertions on INVOICE_ITEM it is made sure that there exists a record for the given InvoiceNumber in INVOICE. Similarly, in the INSERT TRIGGER of INVOICE, it is specified that the insertion for an INVOICE can happen only from INSERT TRIGGER of INVOICE_ITEM upon the insertion of its first record. Otherwise, insertion is prohibited and only modifications of non-primary key fields are permitted.

UPDATE TRIGGER:

An update of key on INVOICE automatically updates INVOICE_ITEM. An update of foreign key on INVOICE_ITEM is permitted as long as there exists the new key as primary key value on INVOICE .If not, the update is prohibited by DBMS. A trigger is only required on UPDATE of INVOICE_ITEM to recalculate the INVOICE price/amount for the old and new InvoiceNumbers

DELETE TRIGGER:

Deleting a record in INVOICE subsequently deletes corresponding record(s) on INVOICE_ITEM . Deletion of INVOICE_ITEM records happen as usual as long as it is not the last child for a parent , when it is , the parent record corresponding to that InvoiceNumber is also deleted as there is no meaning in having an INVOICE that doesn't have any INVOICE_ITEM in it. Also, every time an INVOICE_ITEM is deleted the price of that order has to be recalculated.

IMPLEMENTATION OF INSERT TRIGGERS ON SQL SERVER 2008

Trigger on INVOICE_ITEM

```
TRIGGER ON INVOICE_ITEM

ALTER TRIGGER pricecalculation ON INVOICE_ITEM

INSTEAD OF INSERT

AS

DECLARE @ExtPrice Money

DECLARE @ordernum int

DECLARE @itemnum int

DECLARE @UPrice Money

DECLARE @id int

DECLARE @Quantity int

declare @subtotal MONEY

declare @parentorder int

     set @ordernum= (select InvoiceNumber from inserted i);

     set @id= (select ServiceID from inserted i);

     set @Quantity= (select Quantity from inserted i);

     set @itemnum= (select ItemNumber from inserted i);

     set @UPrice= (select UnitPrice from SERVICE
                     where SERVICE.ServiceID=@id);
     set @ExtPrice= @UPrice * @Quantity;
     set @parentorder=(select count(*) from INVOICE
                     where InvoiceNumber=@ordernum);
     if @parentorder=0
          begin
                INSERT INTO INVOICE
                     values (@ordernum,0,'1/1/1','1/1/1',0,0,0);
          end
     INSERT INTO INVOICE_ITEM
       values(@ordernum,@itemnum,@id,@Quantity,@UPrice,@ExtPrice);
          set @subtotal=(select Subtotal from INVOICE
                where InvoiceNumber=@ordernum);
     update INVOICE set Subtotal=@subtotal+@ExtPrice
          where InvoiceNumber=@ordernum;
     update INVOICE set Tax= INVOICE.Subtotal * 0.1
          where InvoiceNumber=@ordernum;
     update INVOICE set TotalAmount = INVOICE.Subtotal + INVOICE.Tax
          where InvoiceNumber=@ordernum;
GO
```

Trigger on Invoice

```
TRIGGER ON INVOICE
ALTER TRIGGER insertion_rules ON INVOICE
INSTEAD OF INSERT
AS
declare @ordercount int
declare @ordernum int
declare @custid int
declare @datein DATE
declare @dateout DATE
set @custid=(select CustomerID from inserted i);
set @ordernum=(select InvoiceNumber from inserted i);
if @custid = 0
      begin
      set @custid=RAND()*(130-110)+110
            if @custid%5 <> 0
            begin
                  set @custid=@custid-(@custid%5)
            end
      set @datein=getdate();
      set @dateout=getdate() + 5 ;
      INSERT INTO INVOICE values
(@ordernum,@custid,@datein,@dateout,0,0,0);
      end
else
      begin
      set @ordercount=(select count(*)
                            from INVOICE_ITEM
                            where InvoiceNumber=@ordernum);
      if @ordercount=0
            Begin
            RAISERROR('Manual Insertion of an INVOICE prohibited.
            Atleast one record must exist in INVOICE_ITEM table to
            auto-generate the invoice for given InvoiceNumber',16,1);
            end
      else
            Begin
            RAISERROR('Data already exists for given InvoiceNumber.
            Use UPDATE to make any changes',16,1);
            end
      end
```

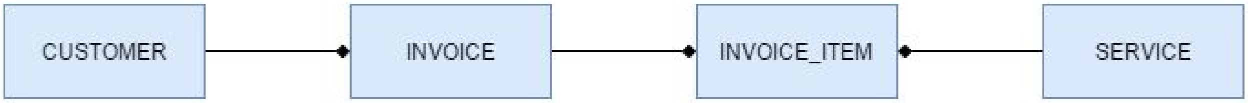
```
GO
```

IMPLEMENTATION OF DELETE TRIGGER

```
ALTER TRIGGER child_deletion ON INVOICE_ITEM
AFTER DELETE AS
     declare @ordercount int
     declare @ordernum int
     declare @subtotal int
     declare @tax int
     declare @amount int

     set @ordernum=(select distinct InvoiceNumber
                      from deleted d);
     print @ordernum
     set @ordercount=(select count(*) from INVOICE_ITEM
                           where InvoiceNumber=@ordernum);
     print @ordercount
     if @ordercount=0
           begin
                print @ordernum
                DELETE FROM INVOICE where InvoiceNumber=@ordernum
           End
     else
           begin
                set @subtotal=(select Subtotal
                                from INVOICE
                                where InvoiceNumber=@ordernum)-
                                (select ExtendedPrice
                                from INVOICE_ITEM
                                where InvocieNumber=@ordernum);
                set @tax = @subtotal*0.1
                set @amount = @subtotal+@tax
                update INVOICE
                set Subtotal=@subtotal,Tax=@tax,TotalAmount=@amount
                where InvoiceNumber=@ordernum;
           End
GO
```

REDESIGN OF THE DATABASE

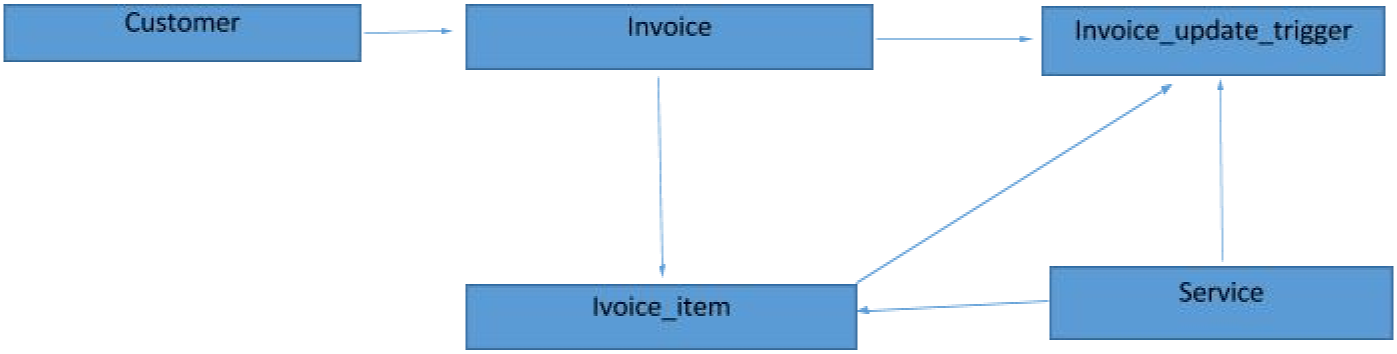
The following illustrates the dependencies for all each table in a revised form of the database



The dependency graph can be extended for views and other database constructs, such as stored procedures and triggers. Any additional object like a view, stored procedure or trigger shall be represented as a box in the graph, and directed arrows are drawn from all such objects that can affect the newly added object.

For this purpose, let us consider a trigger Invoice_update_trigger that makes certain validation checks before allowing an invoice to be updated. Consider further that the triggers queries from the tables INVOICE, INVOICE_ITEM and SERVICE.

The extended dependency graph can be drawn as:



CHANGING TABLE NAMES

The following steps describe renaming of the INVOICE to CUST_INVOICE.

(1) Create a new table named CUST_INVOICE.
(2) Copy all the data from INVOICE to CUST_INVOICE.
(3) Add the foreign key constraint between INVOICE and CUST_ORDER.
(4) Drop INVOICE.

Change the name of the INVOICE table to CUST_INVOICE

As there are two objects dependent on the INVOICE table, namely, the table INVOICE_ITEM and the trigger INVOICE_UPDATE_TRIGGER.

1) Create a new table CUST_INVOICE. Use the same column definitions as the table INVOICE.
2) Define the constraints on table CUST_INVOICE. By giving them names differnet from the constraints names on table INVOICE. Define InvoiceNumber as the primary key. Define a foreign key constraint on column CustomerID of table CUST_INVOICE referencing primary key CustomerID of table CUSTOMER.
3) Copy data from INVOICE to CUST_INVOICE.
4) Change the foreign key refernce in table INVOICE_ITEM to point to column InvoiceNumber in the new tabe CUST_INVOICE.
5) Drop the table INVOICE

IMPLEMENTATION OF TABLE NAME CHANGES

DB used : sql server 2008

```
sp_rename 'INVOICE','CUST_INVOICE';
```

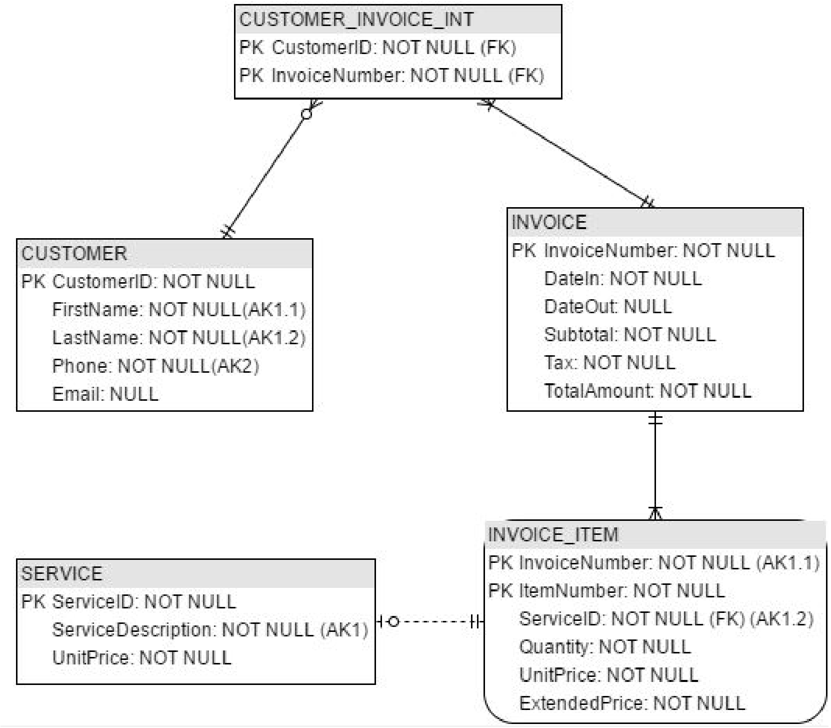
```
ANS 2)C) BY ISHU JUNEJA
CREATE TABLE CUST_INVOICE(
      InvoiceNumber Int NOT NULL,
      CustomerID Int NOT NULL,
      DateIn Date NOT NULL,
      SubTotal Numeric NOT NULL,
      Tax Numeric NOT NULL,
      TotalAmount Numeric NOT NULL,
      CONSTRAINT Cust_invoice_PK
      PRIMARY KEY (InvoiceID)
      CONSTRAINT Cust_Invoice_cust_FK
      FOREIGN KEY (CustomerID) REFERENCES CUSTOMER(CustomerID)
);

INSERT INTO CUST_INVOICE
      (InvoiceNumber, CustomerID, DateIn, DateOut, SubTotal,
Tax, TotalAmount)
SELECT InvoiceNumber, CustomerID, DateIn, DateOut, SubTotal,
TotalAmount From INVOICE;
```

Drop the foreign key reference from Table INVOICE_ITEM to INVOICE and create the new constraint from INVOICE_ITEM to CUST_INVOICE.

```
ALTER TABLE INVOICE _ITEM DROP CONSTRAINT Invoice_Item_Inv_FK;
ALTER TABLE INVOICE_ITEM
      ADD CONSTARINT Invoice_Item_Cust_Order_FK
      FOREIGN KEY(InvoiceNumber)
      REFERENCES CUST_INVOICE(InvoiceNumber);
--Now drop the old table INVOICE;
DROP TABLE INVOICE;
```

TABLE CONSTRAINT MODIFICATIONS

The following describes how to modify table constraints to allow multiple customers for the same invoice. As the database's rule in primary key, it is unique and not allow to duplicate. The one way to solve the multiple customers per order is to create sub table to store the Customer_id nd Invoice_number. Allowing multiple customers per order changes the relationship cardinality between the CUSTOMER and INVOICE tables from 1:N to M:N. In order to ensure referential integrity between CUSTOMER and INVOICE entries, an intersection table CUSTOMER_INVOICE_INT must be created, which stores the primary keys of the CUSTOMER and INVOICE tables as as parts of a composite primary key. First, the table CUSTOMER_INVOICE_INT must be created. Then, the (CustomerID, InvoiceNumber) tuples in the INVOICE table must be inserted into CUSTOMER_INVOICE_INT. The field CustomerID must be removed from the table INVOICE. Finally, relationships must be established between the CUSTOMER table and CUSTOMER_INVOICE_INT and the INVOICE table and CUSTOMER_INVOICE_INT to establish CustomerID and InvoiceNumber as foreign keys in CUSTOMER_INVOICE_INT.



IMPLEMENTATION OF CONSTRAINT MODIFICATIONS

SQL Statements for Redesign of the Database

```
--Create the new table CUSTOMER_INVOICE_INT

CREATE TABLE CUSTOMER_INVOICE_INT (

      CustomerID INT NOT NULL,

      InvoiceNumber INT NOT NULL,

      CONSTRAINT pk_id PRIMARY KEY(CustomerID, InvoiceNumber),

      CONSTRAINT fk_id FOREIGN KEY(InvoiceNumber)

            REFERENCES CUST_INVOICE(InvoiceNumber),

      CONSTRAINT fk_id1 FOREIGN KEY(CustomerID)

            REFERENCES SERVICE(CustomerID)

   );


--Insert all CustomerID, InvoiceNumber tuples into
CUSTOMER_INVOICE_INT

INSERT INTO CUSTOMER_INVOICE_INT (CustomerID, InvoiceNumber)

      SELECT CustomerID, InvoiceNumber FROM INVOICE;


--Drop the CustomerID reference constraint from INVOICE

ALTER TABLE CUST_INVOICE

      DROP CONSTRAINT Cust_Invoice_cust_FK;


--Drop the CustomerID reference from INVOICE

ALTER TABLE CUST_INVOICE

      DROP COLUMN CustomerID
```

INVESTIGATION OF PRIMARY KEY CHANGES

The following is a consideration of primary key changes for the CUSTOMER table

To change the primary key of CUSTOMER to (FirstName, LastName), we must first determine if the primary key is currently unique within the existing database. If all the (FirstName, LastName) tuples are unique, we must decide whether it is likely that a non-unique tuple will be entered. Considering how common many names are, it is a bad idea to use only (FirstName, LastName) as a primary key. The correlated subquery below can be used to determine whether more than one customer with the same first and last names exist in the current database.

```
--Correlated subquery to determine whether different CustomerID
values
--exist for the same (FirstName,LastName) tuple.  Checks for
duplicate tuples.
SELECT     C1.FirstName, C1.LastName
FROM       CUSTOMER AS C1
WHERE EXISTS
        (SELECT    C2.FirstName, C2.LastName
        FROM  CUSTOMER AS C2
        WHERE C1.FirstName == C2.FirstName
        AND        C1.LastName == C2.LastName
        AND        C1.CustomerID <> C2.CustomerID);
```

2-G. Suppose that (FirstName, LastName) can be made the primary key of CUSTOMER. Make appropriate changes to the table design with this new primary key.

--See question 2-H

2-H. Code all SQL statements necessary to implement the changes described in question 2-G.

```
--Add columns for FirstName and LastName to CUST_INVOICE
--These will be foreign keys for referencing CUSTOMER later
ALTER TABLE CUST_INVOICE
    ADD COLUMN FirstName VARCHAR(50) NULL,
                    LastName VARCHAR(50) NULL;

--Fill the new columns with data from CUSTOMER
INSERT INTO CUST_INVOICE
```

```
    (FirstName, LastName)
    SELECT FirstName, LastName
    FROM CUSTOMER AS C
    WHERE C.CustomerID = CUST_INVOICE.CustomerID;

--Drop the CustomerID FK in CUST_INVOICE,
--Then drop the CustomerID column
ALTER TABLE CUST_INVOICE
    DROP CONSTRAINT FK_CustomerID;
    DROP COLUMN CustomerID

--Drop the CustomerID primary key constraint
--Add a UNIQUE constraint to CustomerID
--Add(FirstName,LastName) as the primary key
ALTER TABLE CUSTOMER
    DROP CONSTRAINT CustomerID,
    ADD CONSTRAINT Cust_AK1 UNIQUE(CustomerID),
    ADD CONSTRAINT Cust_PK PRIMARY KEY (FirstName, LastName)
        ON UPDATE CASCADE
        ON DELETE CASCADE;

--A constraint with FirstName and LastName in CUST_INVOICE
--now existing as foreign keys referencing CUSTOMER
ALTER TABLE CUST_INVOICE
    ADD CONSTRAINT Cust_FK FOREIGN KEY(FirstName, LastName)
        REFERENCES CUSTOMER(FirstName, LastName)
              ON UPDATE NO ACTION;
```