Rising Star Bakery

Team SelectStar

Gabriel Monroy	00001065839
Harrison Ng	00001069572
Mrinalini Shekhawat	00001065421
Roopam Sinha	00001026984

CONTENTS

1. Our Business Application

Rising Star Bakery is a new and upcoming whole sale bakery and has enlisted Team SelectStar's services to design and implement the database for its business application.

The business application for Rising Star Bakery provides business management for a wholesale bakery. It is able to do so by allowing for the management of sales, suppliers, inventory, and recipes. The application utilizes a built in audit log to allow for trends to be viewed across price changes. The application revolves around the customers, sales orders and any data necessary to get their transaction from requested goods to products delivered, all while managing the bakeries inventory.

2. Types of Users

The application has four main users which are listed below:

Chefs

The Chefs are the star users of our application. They bake the yummy goods that will be sold by the wholesale bakery.

Procurement Manager

The Procurement Manager is responsible for maintaining the inventory and purchasing raw ingredients required to bake our yummy goods. He will also manage list of suppliers.

Sales Reps

The Sales reps manage customer information and sales orders. They must know how to smile!

Business Analyst

The Business Analyst is responsible for making sure our business is profitable. He is able to view trends and historical data to ensure that business is healthy.

3. Use Cases for the Application

Below the use cases for the application have been enumerated for each of the users.

Chefs

- Chef would like to determine what to bake based on a sales order id and what product was ordered in that sale.
- Chef will increase number of baked goods available for a given order id and product id after they have been baked.
- Chef will look at a baked good and determine what ingredients are available for this baked good given the recipe.

Procurement Manager

- Procurement Manager would be able to find the supplier selling an ingredient at the lowest price and create a Purchase Order.
- Procurement Manager would be able to update Purchase Order and ingredients, once purchase order is fulfilled.
- Procurement Manager would be able to add new Ingredients
- Procurement Manager would be able to view existing suppliers and update with new supplier details and item prices.

Sales Reps

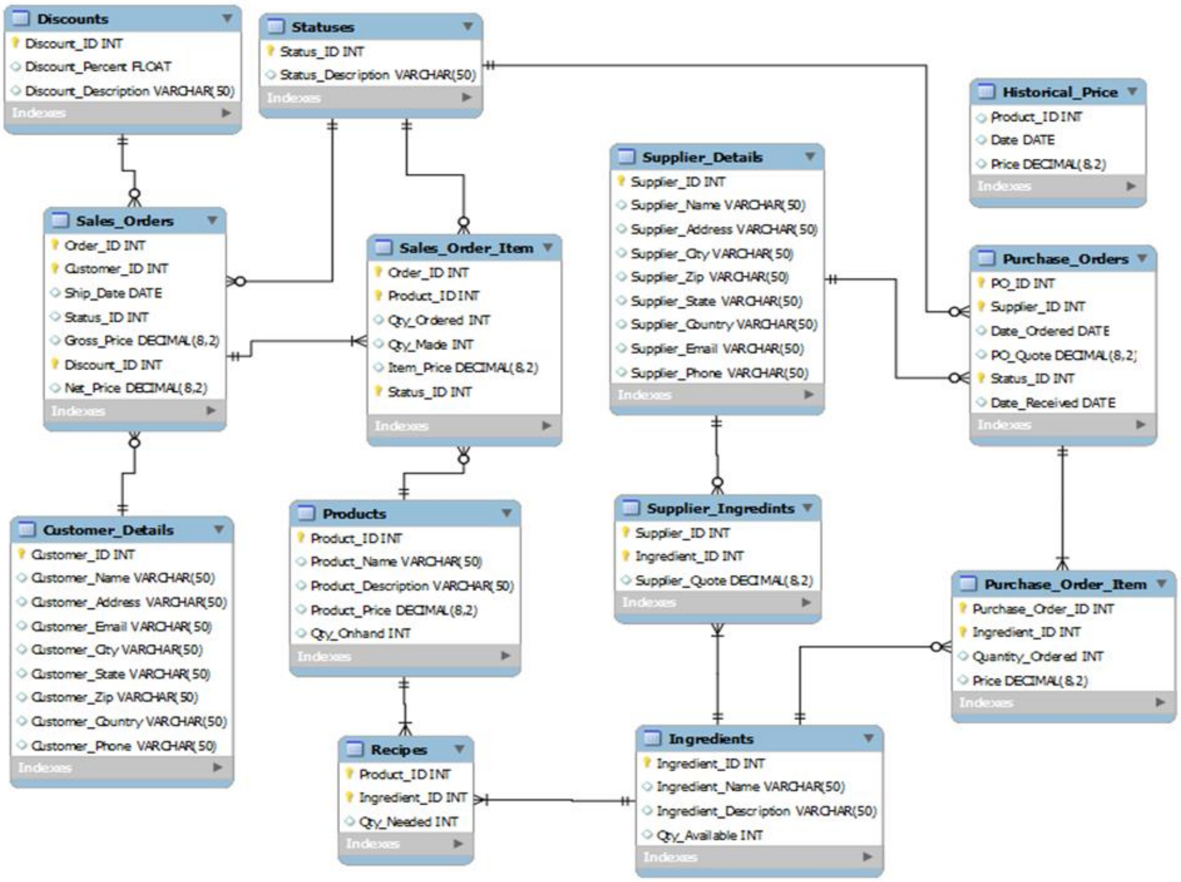
- Sales reps would be able to create and update sales orders
- Sales reps would be able to search existing customers or sales orders with customer ID or order ID
- Sales reps would be able to update a Sales Order
- Sales reps would be able to add a new customer or edit details an existing one
- Sales reps would be able to calculate the total sales for a particular time period

Business Analyst

- The Business Analyst would be able to create quick report views for gross profit, order status, and fulfilled products
- The Business Analyst would be able to view total historical sales by customer
- The Business Analyst would be able to change product sales prices based on demand
- The Business Analyst would be able to view total sales by product to determine whether to continue or discontinue production of the item

4. Our UML Model

Below is the UML model for the business application.



5. Enumerating Queries, Creating and Executing SQL Queries

Below we have enumerated the queries, in English, as well as their respective SQL queries.

Chefs

A. Determine what to bake based on sales order item table

```
SELECT P.PRODUCT_NAME, SOI.QTY_ORDERED
FROM SALES_ORDER_ITEM SOI, PRODUCTS P
WHERE ORDER_ID = 2 AND SOI.PRODUCT_ID = P.PRODUCT_ID
```

PRODUCT_NAME	QTY_ORDERED
Fruit Rainbow Muffins	10
Death by Chocolate Cake	40
Wowzer Bite Cookies	5
Al Sconepone	10
Grey Brownies	5

B. Increase number of baked goods available per sales order item

Before:

```
SELECT * FROM SALES_ORDER_ITEM
WHERE ORDER_ID = 1 AND PRODUCT_ID = 1
```

ORDER_ID	PRODUCT_ID	QTY_ORDERED	QTY_MADE	ITEM_PRICE	STATUS_ID
1	1	50	0	100	1

Update Query:

```
UPDATE SALES_ORDER_ITEM
SET QTY_MADE = 5
WHERE ORDER_ID = 1 AND PRODUCT_ID= 1
```

After:

```
SELECT * FROM SALES_ORDER_ITEM
WHERE ORDER_ID = 1 AND PRODUCT_ID = 1
```

ORDER_ID	PRODUCT_ID	QTY_ORDERED	QTY_MADE	ITEM_PRICE	STATUS_ID
1	1	50	5	100	1

C. Look at baked good recipe and determine what ingredients are available for this baked good.

```
SELECT P.PRODUCT_NAME, I.INGREDIENT_NAME, I.QTY_AVAILABLE
FROM PRODUCTS P, INGREDIENTS I, RECIPES R
WHERE R.PRODUCT_ID = P.PRODUCT_ID
AND R.INGREDIENT_ID = I.INGREDIENT_ID
AND P.PRODUCT_ID = 3
```

PRODUCT_NAME	INGREDIENT_NAME	QTY_AVAILABLE
Wowzer Bite Cookies	Flour	50
Wowzer Bite Cookies	Sugar	65
Wowzer Bite Cookies	Eggs	25
Wowzer Bite Cookies	Baking Powder	15
Wowzer Bite Cookies	Whipping Cream	55
Wowzer Bite Cookies	Butter	35

Procurement Manager

A. Find supplier with the lowest price for ingredient 10 and create a purchase order

Step 1: Find the best price

```
SELECT SUPPLIER_ID
FROM SUPPLIER_INGREDIENTS
WHERE INGREDIENT_ID = 10
AND SUPPLIER_QUOTE <= ALL
          (SELECT SUPPLIER_QUOTE
          FROM SUPPLIER_INGREDIENTS
          WHERE INGREDIENT_ID = 10)
```

SUPPLIER_ID
10

Step 2: Insert into PURCHASE_ORDERS

Before:

```
SELECT * FROM PURCHASE_ORDERS
```

PO_ID	SUPPLIER_ID	DATE_ORDERED	PO_QUOTE	STATUS_ID	DATE_RECEIVED
1	3	05-03-2014	280	1	NULL
2	4	28-02-2014	569	1	NULL
3	10	08-03-2014	185	6	NULL
4	1	09-03-2014	99	6	NULL
5	7	05-01-2014	320	5	NULL

After:

```
INSERT INTO
PURCHASE_ORDERS (SUPPLIER_ID, DATE_ORDERED, PO_QUOTE, STATUS_ID,
DATE_RECEIVED)
VALUES (10,'03-05-2014',NULL,1,NULL)
```

PO_ID	SUPPLIER_ID	DATE_ORDERED	PO_QUOTE	STATUS_ID	DATE_RECEIVED
1	3	05-03-2014	280	1	NULL
2	4	28-02-2014	569	1	NULL
3	10	08-03-2014	185	6	NULL
4	1	09-03-2014	99	6	NULL
5	7	05-01-2014	320	5	NULL
6	10	05-03-2014	NULL	1	NULL

Step 3: Insert into PO_ITEMS

Before:

```
SELECT * FROM PO_ITEMS
```

PO_ID	INGREDIENT_ID	QTY_ORDERED	PRICE
1	6	100	280
2	2	100	100
2	4	100	469
3	7	100	185
4	1	100	99
5	6	100	320

```
INSERT INTO
PO_ITEMS (PO_ID,INGREDIENT_ID,QTY_ORDERED,PRICE)
VALUES (6,10,100,
(100*(SELECT SUPPLIER_QUOTE FROM SUPPLIER_INGREDIENTS WHERE INGREDIENT_ID =
10 AND SUPPLIER_ID = 10)))
```

After:

```
SELECT * FROM PO_ITEMS
```

PO_ID	INGREDIENT_ID	QTY_ORDERED	PRICE
1	6	100	280
2	2	100	100
2	4	100	469
3	7	100	185
4	1	100	99
5	6	100	320
6	10	100	298

Step 4: Update PURCHASE_ORDERS with price

Before:

```
SELECT * FROM PURCHASE_ORDERS
WHERE PO_ID = 6
```

PO_ID	SUPPLIER_ID	DATE_ORDERED	PO_QUOTE	STATUS_ID	DATE_RECEIVED
6	10	05-03-2014	NULL	1	NULL

```
UPDATE PURCHASE_ORDERS
SET PO_QUOTE = (SELECT SUM(PRICE) FROM PO_ITEMS WHERE PO_ID = 6 GROUP BY
PO_ID)
WHERE PO_ID = 6
```

After:

```
SELECT * FROM PURCHASE_ORDERS
WHERE PO_ID = 6
```

PO_ID	SUPPLIER_ID	DATE_ORDERED	PO_QUOTE	STATUS_ID	DATE_RECEIVED
6	10	05-03-2014	298	1	NULL

B. Update purchase orders details when a PO is fulfilled

Step 1: Set the date received in PURCHASE_ORDERS

Before:

```
SELECT * FROM PURCHASE_ORDERS
WHERE PO_ID = 1
```

PO_ID	SUPPLIER_ID	DATE_ORDERED	PO_QUOTE	STATUS_ID	DATE_RECEIVED
1	3	05-03-2014	280	1	NULL

```
UPDATE PURCHASE_ORDERS
SET DATE_RECEIVED = GETDATE(),
STATUS_ID = 4
WHERE PO_ID = 1
```

After:

```
SELECT * FROM PURCHASE_ORDERS
WHERE PO_ID = 1
```

PO_ID	SUPPLIER_ID	DATE_ORDERED	PO_QUOTE	STATUS_ID	DATE_RECEIVED
1	3	05-03-2014	280	4	12-03-2014

Step 2: Update the INGREDIENTS table with new ingredients

Before:

```
SELECT * FROM INGREDIENTS
WHERE INGREDIENT_ID = 6
```

INGREDIENT_ID	INGREDIENT_NAME	INGREDIENT_DESCRIPTION	QTY_AVAILABLE
6	Milk	Full Fat Milk	10

```
UPDATE INGREDIENTS
SET QTY_AVAILABLE = 110
WHERE INGREDIENT_ID = 6
```

After:

```
SELECT * FROM INGREDIENTS
WHERE INGREDIENT_ID = 6
```

INGREDIENT_ID	INGREDIENT_NAME	INGREDIENT_DESCRIPTION	QTY_AVAILABLE
6	Milk	Full Fat Milk	110

Step 3: Update PURCHASE_ORDERS to cancel the PO

Before:

```
SELECT * FROM PURCHASE_ORDERS
WHERE PO_ID = 5
```

PO_ID	SUPPLIER_ID	DATE_ORDERED	PO_QUOTE	STATUS_ID	DATE_RECEIVED
5	7	05-01-2014	320	5	NULL

```
UPDATE PURCHASE_ORDERS
SET STATUS_ID = 6
WHERE PO_ID = 5
```

After:

```
SELECT * FROM PURCHASE_ORDERS
WHERE PO_ID = 5
```

PO_ID	SUPPLIER_ID	DATE_ORDERED	PO_QUOTE	STATUS_ID	DATE_RECEIVED
5	7	05-01-2014	320	6	NULL

C. Insert new data into Ingredients table

```
INSERT INTO
 INGREDIENTS (INGREDIENT_NAME, INGREDIENT_DESCRIPTION, QTY_AVAILABLE)
 VALUES ('Food Color', 'Package of Red, Blue and Green', 10)
```

D. Insert into Supplier Details

```
INSERT INTO
SUPPLIER_DETAILS (SUPPLIER_NAME, SUPPLIER_ADDRESS, SUPPLIER_CITY,
SUPPLIER_STATE, SUPPLIER_COUNTRY, SUPPLIER_ZIP, SUPPLIER_EMAIL, SUPPLIER_PHONE)
VALUES ('Simons Corp','3652 Santa Clara
Avenue','Campbell','CA','95008','USA','claire@simons.com','408-559-6677')
```

E. Insert into Supplier Ingredients

```
INSERT INTO
SUPPLIER_INGREDIENTS(SUPPLIER_ID, INGREDIENT_ID, SUPPLIER_QUOTE)
VALUES (11,1,4.5)
```

Sales Reps

A. Create new sales orders

Case 1: When discount is applied when creating new order

Step 1: Adding a new sales order to table SALES_ORDER

```
INSERT INTO SALES_ORDERS (CUSTOMER_ID, SHIP_DATE, DISCOUNT_ID)
VALUES ('2', '04-02-2014', '2')
```

Step 2: Adding new line items to SALES_ORDER_ITEM per product

```
INSERT INTO SALES_ORDER_ITEM (ORDER_ID, PRODUCT_ID, QTY_ORDERED, QTY_MADE,
ITEM_PRICE)
VALUES ('11', '1', 50, 0, 50*(SELECT PRODUCT_PRICE FROM PRODUCTS WHERE PRODUCT_ID
= '1'))

INSERT INTO SALES_ORDER_ITEM (ORDER_ID, PRODUCT_ID, QTY_ORDERED, QTY_MADE,
ITEM_PRICE)
VALUES ('11', '2', 20, 0, 20*(SELECT PRODUCT_PRICE FROM PRODUCTS WHERE PRODUCT_ID
= '2'))
```

Step 3: Updating NET_PRICE and GROSS_PRICE in table SALES_ORDER

Before:

```
SELECT * FROM SALES_ORDERS
WHERE ORDER_ID = 11
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
11	2	02-04-2014	12-03-2014	1	NULL	2	NULL

```
UPDATE SALES_ORDERS
SET GROSS_PRICE = (SELECT SUM(ITEM_PRICE)FROM SALES_ORDER_ITEM WHERE
ORDER_ID = 11),
NET_PRICE = (SELECT SUM(ITEM_PRICE) FROM SALES_ORDER_ITEM WHERE ORDER_ID = 11)
-((SELECT SUM(ITEM_PRICE) FROM SALES_ORDER_ITEM WHERE ORDER_ID = 11)* 0.01
*(SELECT DISCOUNT_PERCENT FROM DISCOUNTS WHERE DISCOUNT_ID = 2))
WHERE ORDER_ID = 11
```

After:

```
SELECT * FROM SALES_ORDERS
WHERE ORDER_ID = 11
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
11	2	02-04-2014	12-03-2014	1	535	2	508.25

Case 2: Discount is updated after calculating GROSS_PRICE

Step 1: Adding a new sales order to table SALES_ORDER

```
INSERT INTO SALES_ORDERS (CUSTOMER_ID, SHIP_DATE)
VALUES ('3', '04-02-2014')
```

Step 2: Adding a new line items to SALES_ORDER_ITEM per product

```
INSERT INTO SALES_ORDER_ITEM (ORDER_ID, PRODUCT_ID, QTY_ORDERED, QTY_MADE,
ITEM_PRICE)
VALUES ('12', '1', 60, 0, 60*(SELECT PRODUCT_PRICE FROM PRODUCTS WHERE PRODUCT_ID
= '1' ))
```

```
INSERT INTO SALES_ORDER_ITEM (ORDER_ID, PRODUCT_ID, QTY_ORDERED, QTY_MADE,
ITEM_PRICE)
VALUES ('12', '2', 10, 0, 10*(SELECT PRODUCT_PRICE FROM PRODUCTS WHERE PRODUCT_ID
= '2' ))
```

Step 3: Updating GROSS_PRICE in table SALES_ORDER

Before:

```
SELECT * FROM SALES_ORDERS WHERE ORDER_ID = 12
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
12	3	02-04-2014	12-03-2014	1	NULL	1	NULL

```
UPDATE SALES_ORDERS
SET GROSS_PRICE = (SELECT SUM(ITEM_PRICE) FROM SALES_ORDER_ITEM WHERE
ORDER_ID = 12)
WHERE ORDER_ID = 12
```

After:

```
SELECT * FROM SALES_ORDERS WHERE ORDER_ID = 12
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
12	3	02-04-2014	12-03-2014	1	442.5	1	NULL

Step 4: Check GROSS_PRICE in table SALES_ORDER

```
SELECT GROSS_PRICE FROM SALES_ORDERS WHERE ORDER_ID = 12
```

GROSS_PRICE
442.5

Step 5: Update DISCOUNT_ID in SALES_ORDERS

Before:

```
SELECT * FROM SALES_ORDERS
WHERE ORDER_ID = 12
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
12	3	02-04-2014	12-03-2014	1	442.5	1	NULL

```
UPDATE SALES_ORDERS
SET DISCOUNT_ID = 2
WHERE ORDER_ID = 12
```

After:

```
SELECT * FROM SALES_ORDERS
WHERE ORDER_ID = 12
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
12	3	02-04-2014	12-03-2014	1	442.5	2	NULL

Step 6: Update NET_PRICE in SALES_ORDERS

Before:

```
SELECT * FROM SALES_ORDERS
WHERE ORDER_ID = 12
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
12	3	02-04-2014	12-03-2014	1	442.5	2	NULL

```
UPDATE SALES_ORDERS
SET NET_PRICE = (SELECT SUM(ITEM_PRICE)
FROM SALES_ORDER_ITEM WHERE ORDER_ID = 12)
- ((SELECT SUM(ITEM_PRICE) FROM SALES_ORDER_ITEM WHERE ORDER_ID = 12)
* 0.01 * (SELECT DISCOUNT_PERCENT FROM DISCOUNTS WHERE DISCOUNT_ID = 2))
WHERE ORDER_ID = 12
```

After:

```
SELECT * FROM SALES_ORDERS
WHERE ORDER_ID = 12
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
12	3	02-04-2014	12-03-2014	1	442.5	2	420.38

Case 3: Default Discount

Step 1: Adding a new sales order to table SALES_ORDERS

```
INSERT INTO SALES_ORDERS (CUSTOMER_ID, SHIP_DATE)
VALUES ('2', '04-02-2014')
```

Step 2: Adding a new line items to SALES_ORDER_ITEM per product

```
INSERT INTO SALES_ORDER_ITEM (ORDER_ID, PRODUCT_ID, QTY_ORDERED, QTY_MADE,
ITEM_PRICE)
VALUES ('13', '1', 50, 0, 50*(SELECT PRODUCT_PRICE FROM PRODUCTS WHERE PRODUCT_ID
= '1' ))

INSERT INTO SALES_ORDER_ITEM (ORDER_ID, PRODUCT_ID, QTY_ORDERED, QTY_MADE,
ITEM_PRICE)
VALUES ('13', '2', 20, 0, 20*(SELECT PRODUCT_PRICE FROM PRODUCTS WHERE PRODUCT_ID
= '2' ))
```

Step 3: Updating NET_PRICE and GROSS_PRICE in table SALES_ORDERS

Before:

```
SELECT * FROM SALES_ORDERS
WHERE ORDER_ID = 13
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
13	2	02-04-2014	12-03-2014	1	NULL	1	NULL

```
UPDATE SALES_ORDERS
SET GROSS_PRICE = (SELECT SUM(ITEM_PRICE)FROM SALES_ORDER_ITEM WHERE
ORDER_ID = 13),
NET_PRICE = (SELECT SUM(ITEM_PRICE) FROM SALES_ORDER_ITEM WHERE ORDER_ID = 13)
-((SELECT SUM(ITEM_PRICE) FROM SALES_ORDER_ITEM WHERE ORDER_ID = 13)* 0.01
*(SELECT DISCOUNT_PERCENT FROM DISCOUNTS WHERE DISCOUNT_ID = 1))
WHERE ORDER_ID = 13
```

After:

```
SELECT * FROM SALES_ORDERS
WHERE ORDER_ID = 13
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
13	2	02-04-2014	12-03-2014	1	535	1	535

B. Search existing customers or sales orders with customer ID or order ID

```
SELECT * FROM CUSTOMER_DETAILS WHERE CUSTOMER_ID = 1
```

```
SELECT * FROM SALES_ORDERS WHERE CUSTOMER_ID = 1
```

```
SELECT * FROM SALES_ORDERS WHERE ORDER_ID = 1
```

```
SELECT * FROM SALES_ORDERS
WHERE STATUS_ID = 4
```

C. Update quantity ordered for a sales order

Step 1: Update SALES_ORDER_ITEM

Before:

```
SELECT * FROM SALES_ORDER_ITEM
WHERE ORDER_ID = 1 AND PRODUCT_ID = 1
```

ORDER_ID	PRODUCT_ID	QTY_ORDERED	QTY_MADE	ITEM_PRICE	STATUS_ID
1	1	50	5	100	1

```
UPDATE SALES_ORDER_ITEM
SET QTY_ORDERED = 80
WHERE ORDER_ID = 1 AND PRODUCT_ID = 1
```

After:

```
SELECT * FROM SALES_ORDER_ITEM
WHERE ORDER_ID = 1 AND PRODUCT_ID = 1
```

ORDER_ID	PRODUCT_ID	QTY_ORDERED	QTY_MADE	ITEM_PRICE	STATUS_ID
1	1	80	5	100	1

Step 2: Update NET_PRICE and GROSS_PRICE

Before:

```
SELECT * FROM SALES_ORDERS
WHERE ORDER_ID = 1
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
1	1	29-03-2014	03-03-2014	1	375	1	356.25

```
UPDATE SALES_ORDERS
SET GROSS_PRICE = (SELECT SUM(ITEM_PRICE) FROM SALES_ORDER_ITEM WHERE
ORDER_ID = 1),
NET_PRICE = (SELECT SUM(ITEM_PRICE) FROM SALES_ORDER_ITEM WHERE ORDER_ID = 1)
-((SELECT SUM(ITEM_PRICE) FROM SALES_ORDER_ITEM WHERE ORDER_ID = 1)*
0.01 * (SELECT DISCOUNT_PERCENT FROM DISCOUNTS WHERE DISCOUNT_ID = 1
))
WHERE ORDER_ID = 1
```

After:

```
SELECT * FROM SALES_ORDERS
WHERE ORDER_ID = 1
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	ORDER_DATE	STATUS_ID	GROSS_PRICE	DISCOUNT_ID	NET_PRICE
1	1	29-03-2014	03-03-2014	1	475	1	475

D. Insert new customer

```
INSERT INTO CUSTOMER_DETAILS (CUSTOMER_NAME, CUSTOMER_ADDRESS,
CUSTOMER_CITY, CUSTOMER_STATE, CUSTOMER_ZIP, CUSTOMER_COUNTRY,
CUSTOMER_EMAIL, CUSTOMER_PHONE)
VALUES ('Cold Stone Bakery', '757 ', 'El Camino Real', 'CA', '94087', 'USA',
'enquiry@coldstonecreamery.com', '4087394420')
```

E. Update customer details

Before:

```
SELECT CUSTOMER_EMAIL FROM CUSTOMER_DETAILS
WHERE CUSTOMER_NAME = 'Safeway'
```

CUSTOMER_EMAIL
help@safeway.com

```
UPDATE CUSTOMER_DETAILS
SET CUSTOMER_EMAIL = 'info@safeway.com'
WHERE CUSTOMER_NAME = 'Safeway'
```

After:

```
SELECT CUSTOMER_EMAIL FROM CUSTOMER_DETAILS
WHERE CUSTOMER_NAME = 'Safeway'
```

CUSTOMER_EMAIL
info@safeway.com

F. Calculate total sales to date in current month

```
SELECT SUM(GROSS_PRICE) AS TOTAL_SALES_FOR_PERIOD
FROM SALES_ORDERS
WHERE ORDER_DATE > '03-01-2014' AND ORDER_DATE < GETDATE()
```

Business Analyst

Views

A. Create a gross profit view report, Revenue - Cogs = Gross Profit.

```
CREATE VIEW [MONTH_GROSS_PROFIT] AS
SELECT *, REVENUE - COGS AS [GROSS_PROFIT]
FROM
(SELECT SUM(GROSS_PRICE) AS REVENUE
FROM SALES_ORDERS
WHERE SHIP_DATE >= DATEADD(DAY,-30,GETDATE())) AS INFLOW,
(SELECT SUM(PO_QUOTE) AS COGS
FROM PURCHASE_ORDERS
WHERE DATE_ORDERED >= DATEADD(DAY,-30,GETDATE())) AS OUTFLOW

SELECT * FROM MONTH_GROSS_PROFIT
```

REVENUE	COGS	GROSS_PROFIT
4900	1133	3767

B. Create a quick order status view for fast lookup of order status

```
CREATE VIEW [ORDER_STATUS] AS
SELECT ORDER_ID, CUSTOMER_ID, SHIP_DATE, NET_PRICE, STATUS_DESCRIPTION
FROM SALES_ORDERS, STATUSES
WHERE SALES_ORDERS.STATUS_ID = STATUSES.STATUS_ID

SELECT * FROM ORDER_STATUS
```

ORDER_ID	CUSTOMER_ID	SHIP_DATE	NET_PRICE	STATUS_DESCRIPTION
1	1	29-03-2014	356.25	Open
2	2	29-03-2014	332.5	Open
3	3	22-03-2014	950	Open
4	4	22-03-2014	1021.25	Open
5	5	22-03-2014	522.5	Open
6	1	14-03-2014	237.5	Pending Fulfillment
7	1	14-03-2014	332.5	Ready to Ship
8	1	10-03-2014	332.5	Fulfilled
9	1	15-03-2014	237.5	On Hold
10	1	05-04-2014	332.5	Canceled

C. Create view of fulfilled products for analysis on historical data

```
CREATE VIEW [FULFILLED_PRODUCTS] AS
SELECT SO.ORDER_ID AS ORDER_ID, SO.CUSTOMER_ID AS CID, C.CUSTOMER_NAME,
P.PRODUCT_ID AS PID,
PRODUCT_NAME, QTY_ORDERED, SHIP_DATE
FROM SALES_ORDERS SO, SALES_ORDER_ITEM SOI, PRODUCTS P, STATUSES ST,
CUSTOMER_DETAILS C
WHERE SO.ORDER_ID = SOI.ORDER_ID
AND P.PRODUCT_ID = SOI.PRODUCT_ID
AND ST.STATUS_ID = SO.STATUS_ID
AND C.CUSTOMER_ID = SO.CUSTOMER_ID
AND SO.STATUS_ID = 4

SELECT * FROM FULFILLED_PRODUCTS
```

ORDER_ID	CID	CUSTOMER_NAME	PID	PRODUCT_NAME	QTY_ORDERED	SHIP_DATE
8	1	Safeway	1	Fruit Rainbow Muffins	50	10-03-2014
8	1	Safeway	2	Death by Chocolate Cake	20	10-03-2014

Queries

A. View total historical sales by customer

```
SELECT ORDER_ID, SALES_ORDERS.CUSTOMER_ID AS CID, CUSTOMER_NAME, SHIP_DATE,
GROSS_PRICE, DISCOUNT_ID, NET_PRICE, STATUS_ID
FROM SALES_ORDERS, CUSTOMER_DETAILS
WHERE CUSTOMER_DETAILS.CUSTOMER_ID = SALES_ORDERS.CUSTOMER_ID
AND CUSTOMER_DETAILS.CUSTOMER_NAME LIKE 'SAFEWAY'
```

ORDER_ID	CID	CUSTOMER_NAME	SHIP_DATE	GROSS_PRICE	DISCOUNT_ID	NET_PRICE	STATUS_ID
1	1	Safeway	29-03-2014	375	1	356.25	1
6	1	Safeway	14-03-2014	250	1	237.5	2
7	1	Safeway	14-03-2014	350	1	332.5	3
8	1	Safeway	10-03-2014	350	1	332.5	4
9	1	Safeway	15-03-2014	250	1	237.5	5
10	1	Safeway	05-04-2014	350	1	332.5	6

B. Change finished product sales prices based on demand

```
SELECT SALES_ORDERS.ORDER_ID, CUSTOMER_ID, SALES_ORDER_ITEM.PRODUCT_ID,
PRODUCTS.PRODUCT_NAME, QTY_ORDERED, SHIP_DATE
FROM SALES_ORDERS, SALES_ORDER_ITEM, PRODUCTS
WHERE SALES_ORDERS.ORDER_ID = SALES_ORDER_ITEM.ORDER_ID
AND PRODUCTS.PRODUCT_ID = SALES_ORDER_ITEM.PRODUCT_ID
AND PRODUCTS.PRODUCT_ID = 2
```

ORDER_ID	CUSTOMER_ID	PRODUCT_ID	PRODUCT_NAME	QTY_ORDERED	SHIP_DATE
1	1	2	Death by Chocolate Cake	20	29-03-2014
2	2	2	Death by Chocolate Cake	40	29-03-2014
3	3	2	Death by Chocolate Cake	20	22-03-2014
4	4	2	Death by Chocolate Cake	40	22-03-2014
5	5	2	Death by Chocolate Cake	30	22-03-2014
7	1	2	Death by Chocolate Cake	20	14-03-2014
8	1	2	Death by Chocolate Cake	20	10-03-2014
9	1	2	Death by Chocolate Cake	20	15-03-2014
10	1	2	Death by Chocolate Cake	20	05-04-2014

Before:

```
SELECT * FROM PRODUCTS
WHERE PRODUCT_ID = 2
```

PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESCRIPTION	PRODUCT_PRICE	QTY_ONHAND
2	Death by Chocolate Cake	Every Chocolate in the world. Twice.	5	0

```
UPDATE PRODUCTS
SET PRODUCT_PRICE = 14.25
WHERE PRODUCT_ID = 2
```

After:

```
SELECT * FROM PRODUCTS
WHERE PRODUCT_ID = 2
```

PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESCRIPTION	PRODUCT_PRICE	QTY_ONHAND
2	Death by Chocolate Cake	Every Chocolate in the world. Twice.	14.25	0

C. View total sales by product to determine whether to continue or discontinue production of the item

```
SELECT PRODUCTS.PRODUCT_ID, PRODUCT_NAME, SUM(QTY_ORDERED) AS
TOTAL_ORDERED
FROM SALES_ORDERS, SALES_ORDER_ITEM, PRODUCTS
WHERE SALES_ORDERS.ORDER_ID = SALES_ORDER_ITEM.ORDER_ID
AND SHIP_DATE > '2014-03-22'
AND SALES_ORDER_ITEM.PRODUCT_ID = PRODUCTS.PRODUCT_ID
GROUP BY PRODUCTS.PRODUCT_ID, PRODUCT_NAME
```

PRODUCT_ID	PRODUCT_NAME	TOTAL_ORDERED
1	Fruit Rainbow Muffins	110
2	Death by Chocolate Cake	80
3	Wowzer Bite Cookies	45
4	Al Sconepone	20
5	Grey Brownies	10

8. Database Dictionary

TABLES

TABLE 1: CUSTOMER_DETAILS

COLUMNS	TYPE	DATA_TYPE	CHECK
CUSTOMER_ID	NOT NULL	INT	Autogenerate
CUSTOMER_NAME	NOT NULL	VARCHAR(50)	N/A
CUSTOMER_ADDRESS	NOT NULL	VARCHAR(50)	N/A
CUSTOMER_CITY	NOT NULL	VARCHAR(50)	N/A
CUSTOMER_STATE	NOT NULL	VARCHAR(50)	N/A
CUSTOMER_ZIP	NOT NULL	VARCHAR(50)	N/A
CUSTOMER_COUNTRY	NOT NULL	VARCHAR(50)	N/A
CUSTOMER_EMAIL	NULL	VARCHAR(50)	N/A
CUSTOMER_PHONE	NOT NULL	VARCHAR(50)	N/A

PRIMARY_KEY
CUSTOMER_ID

TABLE 2: STATUSES

COLUMNS	TYPE	DATA_TYPE	CHECK
STATUS_ID	NOT NULL	INT	Autogenerate
STATUS_DESCRIPTION	NOT NULL	VARCHAR(50)	N/A

PRIMARY KEY
STATUS_ID

INDEX
STATUS_INDEX ON STATUS_ID

TABLE 3: DISCOUNT

COLUMNS	TYPE	DATA_TYPE	CHECK
DISCOUNT_ID	NOT NULL	INT	Autogenrate
DISCOUNT_PERCENT	NOT NULL	FLOAT	N/A
DISCOUNT_DESCRIPTION	NOT NULL	VARCHAR(50)	N/A

PRIMARY KEY
DISCOUNT_ID

TABLE 4: SALES_ORDERS

COLUMNS	TYPE	DATA_TYPE	CHECK
ORDER_ID	NOT NULL	INT	Autogenerate
CUSTOMER_ID	NOT NULL	INT	N/A
SHIP_DATE	NOT NULL	DATE TYPE	N/A
ORDER_DATE	NOT NULL	DATE TYPE	N/A
STATUS_ID	NOT NULL	INT	N/A
GROSS_PRICE	NULL	DECIMAL(8,2)	N/A
DISCOUNT_ID	NULL	INT	N/A
NET_PRICE	NULL	DECIMAL(8,2)	N/A

PRIMARY KEY
ORDER_ID

FOREIGN KEY
CUSTOMER_ID references to CUSTOMER_ID of CUSTOMER_DETAILS table
STATUS_ID references STATUS_ID of STATUSES table
DISCOUNT_ID references to DISCOUNT_ID of DISCOUNTS table

TABLE 5: PRODUCTS

COLUMNS	TYPE	DATA_TYPE	CHECK
PRODUCT_ID	NOT NULL	INT	Autogenerate
PRODUCT_NAME	NOT NULL	VARCHAR(50)	N/A
PRODUCT_DESCRIPTION	NOT NULL	VARCHAR(50)	N/A
PRODUCT_PRICE	NOT NULL	DECIMAL(8,2)	N/A
QTY_ONHAND	NOT NULL	INT	N/A

PRIMARY KEY
PRODUCT_ID

TABLE 6: HISTORICAL_PRICE

COLUMNS	TYPE	DATA_TYPE	CHECK
PRODUCT_ID	NOT NULL	INT	N/A
PRICE	NOT NULL	DECIMAL(8,2)	N/A
DATE_CHANGED	NOT NULL	DATE	N/A

TABLE 7: SALES_ORDER_ITEM

COLUMNS	TYPE	DATA_TYPE	CHECK
ORDER_ID	NOT NULL	INT	N/A
PRODUCT_ID	NOT NULL	INT	N/A
QTY_ORDERED	NOT NULL	INT	N/A
QTY_MADE	NOT NULL	INT	N/A
ITEM_PRICE	NOT NULL	DECIMAL(8,2)	N/A
STATUS_ID	NOT NULL	INT	N/A

PRIMARY KEY
PRIMARY KEY CLUSTERED (ORDER_ID, PRODUCT_ID)

FOREIGN KEY
ORDER_ID references to ORDER_ID of SALES_ORDERS table
PRODUCT_ID references to PRODUCT_ID of PRODUCTS table
STATUS_ID references to STATUS_ID of STATUSES table

TABLE 8: INGREDIENTS

COLUMNS	TYPE	DATA_TYPE	CHECK
INGREDIENT_ID	NOT NULL	INT	Autogenerate
INGREDIENT_NAME	NOT NULL	VARCHAR(50)	N/A
INGREDIENT_DESCRIPTION	NOT NULL	VARCHAR(50)	N/A
QTY_AVAILABLE	NOT NULL	INT	N/A

PRIMARY KEY
INGREDIENT_ID

TABLE 9: RECIPE

COLUMNS	TYPE	DATA_TYPE	CHECK
PRODUCT_ID	NOT NULL	INT	N/A
INGREDIENT_ID	NOT NULL	INT	N/A
QTY_NEEDED	NOT NULL	INT	N/A

PRIMARY KEY
PRIMARY KEY CLUSTERED (PRODUCT_ID, INGREDIENT_ID)

FOREIGN KEY
PRODUCT_ID references to PRODUCT_ID of PRODUCTS table
PRODUCT_ID references to INGREDIENT_ID of INGREDIENTS table

TABLE 10: SUPPLIER_DETAILS

COLUMNS	TYPE	DATA_TYPE	CHECK
SUPPLIER_ID	NOT NULL	INT	Autogenerate
SUPPLIER_NAME	NOT NULL	VARCHAR(50)	N/A
SUPPLIER_ADDRESS	NOT NULL	VARCHAR(50)	N/A
SUPPLIER_CITY	NOT NULL	VARCHAR(50)	N/A
SUPPLIER_STATE	NOT NULL	VARCHAR(50)	N/A
SUPPLIER_COUNTRY	NOT NULL	VARCHAR(50)	N/A
SUPPLIER_ZIP	NOT NULL	VARCHAR(50)	N/A
SUPPLIER_EMAIL	NULL	VARCHAR(50)	N/A
SUPPLIER_PHONE	NOT NULL	VARCHAR(50)	N/A

PRIMARY_KEY
SUPPLIER_ID

TABLE 11: SUPPLIER_INGREDIENTS

COLUMNS	TYPE	DATA_TYPE	CHECK
SUPPLIER_ID	NOT NULL	INT	N/A
INGREDIENT_ID	NOT NULL	INT	N/A
SUPPLIER_QUOTE	NOT NULL	DECIMAL(8,2)	N/A

PRIMARY KEY
PRIMARY KEY CLUSTERED (SUPPLIER_ID, INGREDIENT_ID)

FOREIGN KEY
SUPPLIER_ID references to SUPPLIER_ID of SUPPLIER_DETAILS table
INGREDIENT_ID references to INGREDIENT_ID of INGREDIENTS table

TABLE 12: PURCHASE_ORDERS

COLUMNS	TYPE	DATA_TYPE	CHECK
PO_ID	NOT NULL	INT	Autogenerate
SUPPLIER_ID	NOT NULL	INT	N/A
DATE_ORDERED	NOT NULL	DATE	N/A
PO_QUOTE	NULL	DECIMAL(8,2)	N/A
STATUS_ID	NOT NULL	INT	N/A
DATE_RECEIVED	NULL	DATE	N/A

FOREIGN KEY
SUPPLIER_ID references to SUPPLIER_ID of SUPPLIER_DETAILS table
STATUS_ID references to STATUS_ID of STATUSES table

TABLE 13: PO_ITEMS

COLUMNS	TYPE	DATA_TYPE	CHECK
PO_ID	NOTNULL	INT	N/A
INGREDIENT_ID	NOT NULL	INT	N/A
QTY_ORDERED	NOT NULL	INT	N/A
PRICE	NOT NULL	DECIMAL(8,2)	N/A

PRIMARY KEY

PRIMARY KEY CLUSTERED (PO_ID, INGREDIENT_ID)

FOREIGN KEY

PO_ID references to PO_ID of PURCHASE_ORDERS table
INGREDIENT_ID references to INGREDIENT_ID of INGREDIENTS table

VIEWS

View 1: MONTH_GROSS_PROFIT

This view allows business analyst to calculate gross margin profit of the business.

```
CREATE VIEW [MONTH_GROSS_PROFIT] AS
SELECT *, REVENUE - COGS AS [GROSS_PROFIT]
FROM
(SELECT SUM(GROSS_PRICE) AS REVENUE
FROM SALES_ORDERS
WHERE SHIP_DATE >= DATEADD(DAY,-30,GETDATE())) AS INFLOW,
(SELECT SUM(PO_QUOTE) AS COGS
FROM PURCHASE_ORDERS
WHERE DATE_ORDERED >= DATEADD(DAY,-30,GETDATE())) AS OUTFLOW
```

View 2: ORDER_STATUS

This view enables the business analyst to view order status based on sales order Id

```
CREATE VIEW [ORDER_STATUS] AS
SELECT ORDER_ID, CUSTOMER_ID, SHIP_DATE, NET_PRICE, STATUS_DESCRIPTION
FROM SALES_ORDERS, STATUSES
WHERE SALES_ORDERS.STATUS_ID = STATUSES.STATUS_ID
```

View 3: FULFILLED_PRODUCTS

This view allows the business analyst to view customer details and product details based on sales order Id

```
CREATE VIEW [FULFILLED_PRODUCTS] AS
SELECT SO.ORDER_ID AS ORDER_ID, SO.CUSTOMER_ID AS CID, C.CUSTOMER_NAME,
P.PRODUCT_ID AS PID,
PRODUCT_NAME, QTY_ORDERED, SHIP_DATE
```

```
FROM SALES_ORDERS SO, SALES_ORDER_ITEM SOI, PRODUCTS P, STATUSES ST,
CUSTOMER_DETAILS C
WHERE SO.ORDER_ID = SOI.ORDER_ID
AND P.PRODUCT_ID = SOI.PRODUCT_ID
AND ST.STATUS_ID = SO.STATUS_ID
AND C.CUSTOMER_ID = SO.CUSTOMER_ID
AND SO.STATUS_ID = 4
```

12. Project Summary

Summarize your experience with this exercise

This project was one of the best experiences for this quarter. We learnt how to communicate our ideas with our teammates and integrate all of these ideas into an amazing DBMS project. We had fun playing with the creative aspects of the project, for example the name and descriptions of the products that the bakery produces. A fun named product that the bakery offers is 'Grey Brownies' with a product description of 'Why do we still call them brownies?'. We also learned how to create a database from scratch and learned the potential problems that can be faced. We had fun working as a team, helping each other improve our understanding of the concepts learned in the class.

What was the hardest part of this project?

The hardest part of this project, without a doubt, was to come up with the database schema. Before we sat down and drew up a basic foundation, all of our understandings of the database and what it should do was greatly varied. We went through multiple iterations of schema and had intense discussions on which tables should stay and which should be removed in order to contain our scope but still be difficult enough for us to learn from it. A very simple schema would prove nothing aside from the fact that we could type commands into a prompt. Too complex a schema would surely have left us in tears during the project due date. We came up with a good balance of tables and functionality for our project after much deliberation and are happy to present what we have.

What problems did you run against in this project?

The better question to ask would be, what problems did we not have. As mentioned before, coming up with a schema proved to be a gauntlet of its own. Aside from that, we had issues throughout the process including the following:

- How do we define the scope?
- How do we setup a trigger table to keep an audit of prices?
- How can we integrate an index in a logical fashion?
- How can we incorporate views that fall within the use cases we’ve defined?

Lastly, we had to ensure that the data we use is realistic.

How did you solve these problems?

The approach that we followed was first we tried to identify the requirements of each user. Answers to the questions, what information in the form of tables and their attributes would each of these users need to perform their day to day functions, what changes would these users probably be required to make etc. Once we had identified the tables and their respective attributes, we revisited our initial queries that we had enumerated to check if they were applicable to our draft database schema design. Unfortunately, the answer to this was no. There were few attributes we had missed. Upon adding these attributes we again revisited the enumerated queries and found that we still had to make further changes to the database schema. Upon trying to figure out the cardinalities and based on inputs from Professor Shailesh, we realized that we were missing two very critical tables. We added tables SALES_ORDER_ITEM and PO_ITEM into the database schema and finally our queries were applicable to the database schema.

Triggers, indexes, and views were all brought up at different times during our project, and were heavily discussed, analyzed, researched, and implemented as needed. They were all treated as enrichment opportunities, and truly helped make this project special.

If you were to do this project again, what methodology would you follow?

As stated before, the main issue stemmed from the fact that we all had a separate idea of what the table should do. Although we had already defined the use cases, there was still a multitude of ways to design the schema to account for these very few use cases. The turning point from confusion to construction was when we finally defined the tables and went through the logical flow from Customer details to supplier details. From there we were able to define the relationships and get everyone on the same page. If we had to do a similar project, we would attempt to get a schema going early to have a point to hold on to for referential understanding. Even if changes had to be made, having a rough schema gives a tangible object for both understanding and criticism.

Suggestions on how to refine this project for the next class

One change that would be beneficial for the future classes would be to give some clarity in terms of the grading rubric. One issue we had was understanding what exactly was meant by an “important” use case. Did this refer to important for the purposes of showcasing a fancy query? Or did this mean important in terms of our database and its functionality. We were not sure and decided to go with the latter.