Bulls Eat

A Food Recommendation and Inventory Management System

ISM 6218 – FALL 2017

Authored by:

Kunal Singh (U33337780)
Aradhna Tiwari (U21785108)
Soumil Dhar (U74918076)

USF UNIVERSITY OF SOUTH FLORIDA.

Contents

USF UNIVERSITY OF SOUTH FLORIDA.

Executive Summary:

Bulls Eat is an online food delivery system that provides a customer to view the catalog and choose the items to order. Database has been designed for the inventory management component of the system to have a check on the current quantity of ingredients v/s the refill threshold.

With an aim of providing a better customer experience as well as to generate revenue, a recommendation system has been built upon using various techniques learnt in the course curriculum. The recommendation system aims at recommending the menu items to be ordered by a customer.

The database design is built in depth on certain parts of the system and normalization principles are used to maintain the relationships between two or more tables and control the redundancy. The data quality is maintained by enforcing integrity and check constraints. The system is optimized and tuned to achieve high performance.

Topic Area	Description	Points
Database Design	This part should include a logical database design (for the relational model), using normalization to control redundancy and integrity constraints for data quality.	30
Query Writing	This part is another chance to write SQL queries, explore transactions, and even do some database programming for stored procedures.	30
Performance Tuning	In this section, you can capitalize and extend your prior experiments with indexing, optimizer modes, partitioning, parallel execution and any other techniques you want to further explore.	25
Other Topics	Here you are free to explore any other topics of interest. Suggestions include DBA scripts, database security, interface design, data visualization, data mining, and NoSQL databases.	15

USF UNIVERSITY OF SOUTH FLORIDA.

Project Requirements:

- The schema for Bulls Eat has been created in Oracle using an existing database DB233.
- Different accesses have been built upon to enforce security.
- Reusability has been implemented using stored procedures for performing repetitive tasks.
- Automation using triggers has been implemented in both data population as well as calculations needed for an update.
- Web scraping is used to generate part of data.
- Backup of the tables have been taken to ensure the system availability.

Part 1: Database Design:

A. Design Assumptions:

Bulls Eat has various modules and sub modules integrated together to create the system but due to complexity reasons, these components have not been designed and only certain parts have been built in depth to showcase its associated role. Several assumptions have been made which are listed below:

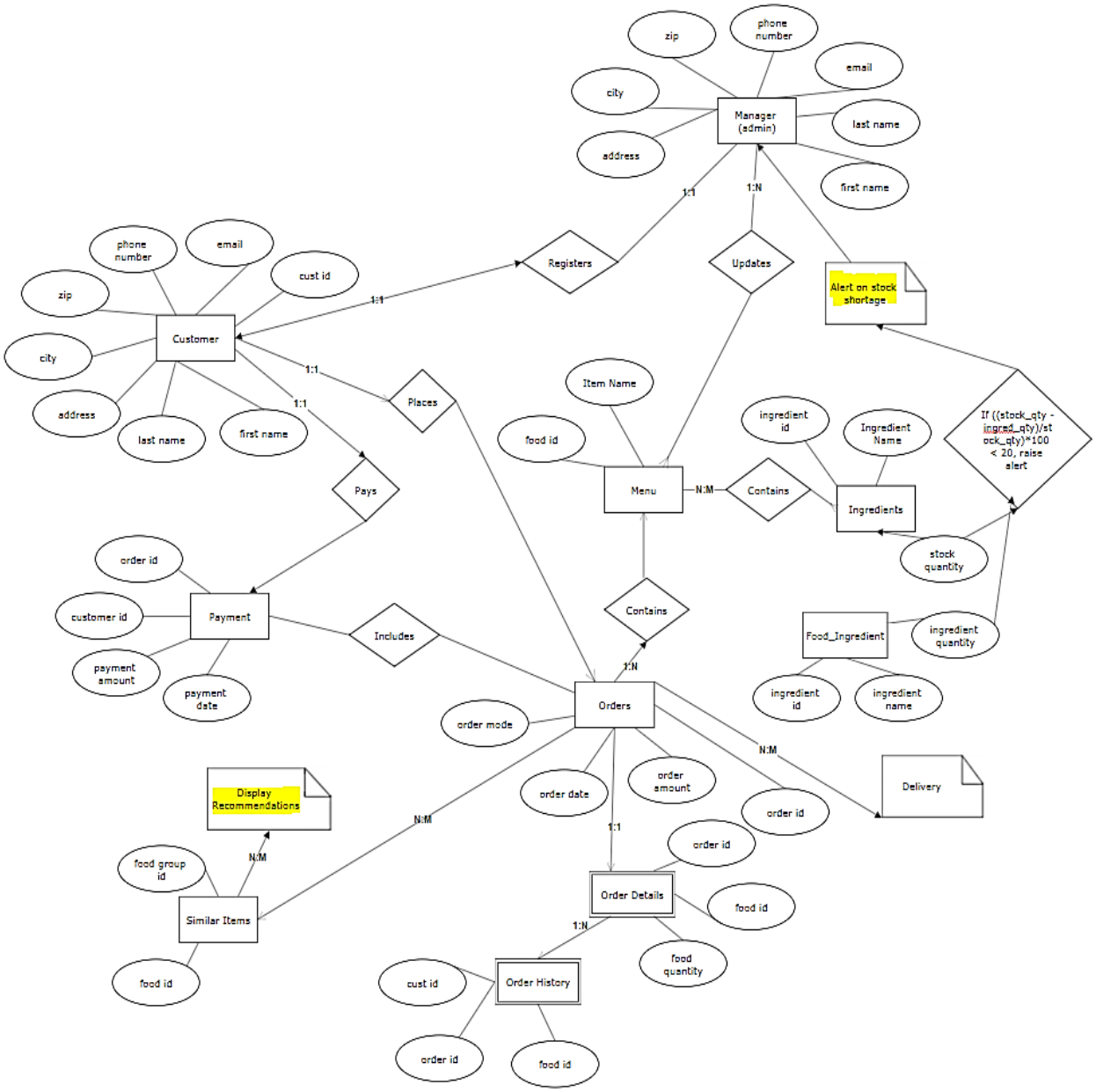
- The menu items listed belong to a single outlet.
- All the recipes are prepared by a team of chefs which are not included in the design.
- Refilling of the inventory is notified but the supplier details are not included in the design.
- Measurements can sometimes be vague, for example in a recipe description one tends to quantify an item using measurements like teaspoon, tablespoon, cup, a pinch etc. To reduce the complexity of mapping these measurements to their absolute values we have used the standard units of measurement.
- The delivery component is not included in the design as the focus of the system plays around the items ordered and the items that one can likely to order.
- It allows our customers to register with the system and their corresponding profile gets created with the demographic details provided.
- This basic profile is enhanced as the customer places an order choosing one or more items from the menu and provides ratings to an item.
- It allows our system manager (admin) to manage inventory via automatic email notification which is sent when the current quantity reaches the refill threshold value. The manager can also monitor transactions and delivery.
- The recommender system will help assist both existing customers as well as new customers to help place an order.

B. Entity Relationship Diagram:

High – Level Diagram:

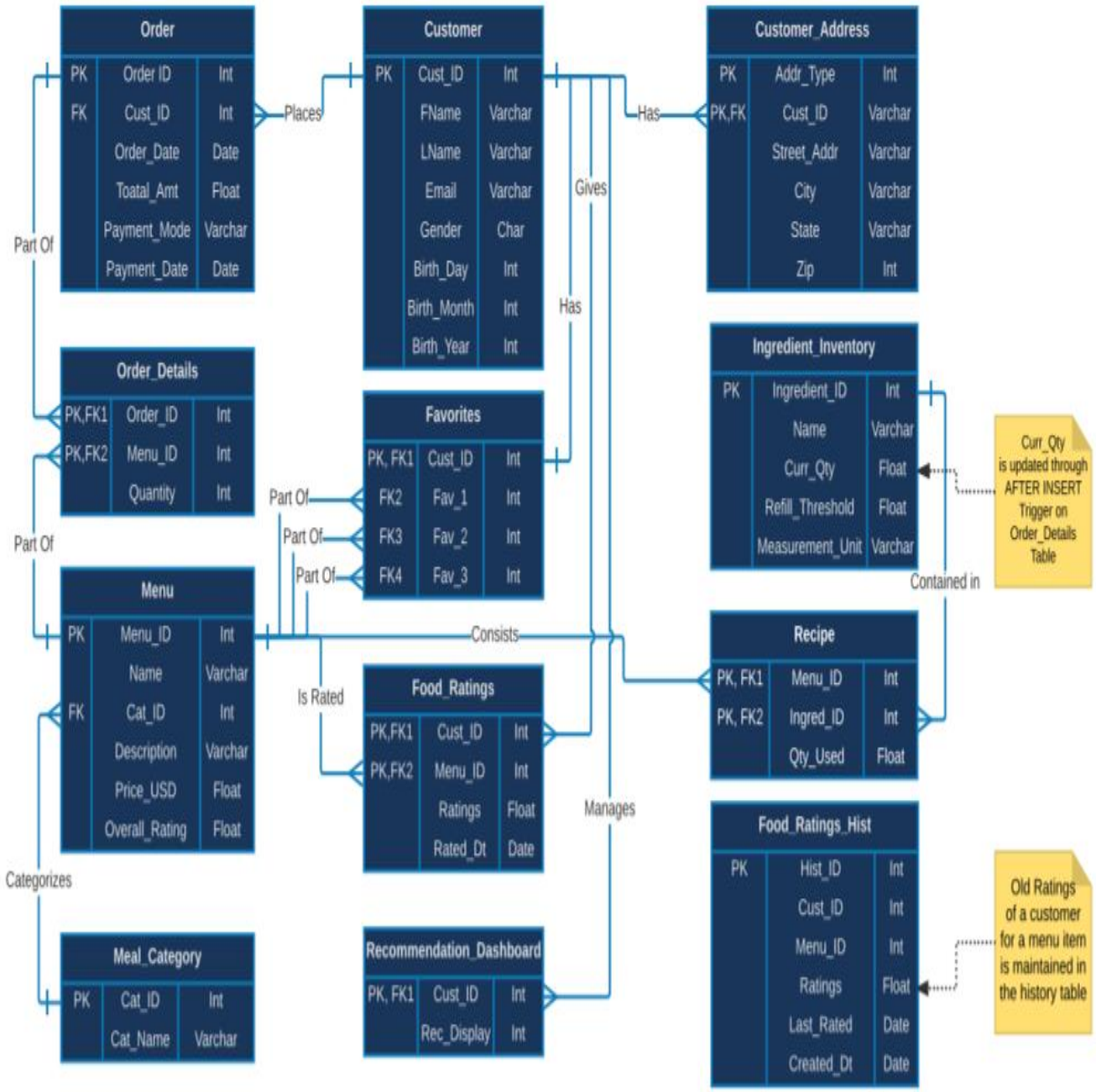
Bulls Eat is a food delivery outlet which has an inventory component to maintain the ingredients quantity and displays the recommendations of the menu items to be ordered by a customer using various methodologies of recommendation. Below is a high level initial design which was built to understand and identify the general components of the system.

The components built in depth and part of the project has been highlighted below:



Low – Level Diagram:

As highlighted in the high-level design, the inventory component as well as the recommendation components are the prime focus and have been built in depth as shown below in the low-level design:



USF UNIVERSITY OF SOUTH FLORIDA.

C. Data Dictionary:

The data dictionary of the tables below describes an in-depth definition of the tables along with the inter relationships and number of rows contained in each of the tables.

Customer:

The customer table comprises of the demographic information of the customers who have ordered food from Bulls Eat.

Column Name	Data Type	Size	Constraint	Description	Example
Cust_ID	Integer	10	Primary Key	Customer ID, auto generated	5
FName	Varchar	50	Nullable Default='FNU'	First Name	Charles
LName	Varchar	50	Nullable Default='LNU'	Last Name	Wise
Email	Varchar	100	Not Null	Email Address	charles.wise@juno.com
Gender	Char	1	Not Null	Gender	M
Birth_Day	Integer	2	Nullable	Extracts the day of birth	29
Birth_Month	Integer	2	Nullable	Extracts the month of birth	2
Birth_Year	Integer	4	Nullable	Extracts the year of month	1984

Customer Address:

The customer address table comprises of all the types of addresses of a customer. This holds a many to one relationship with the customer table as a customer can be associated with more than one address type, for example, home, work, others etc.

Column Name	Data Type	Size	Constraint	Description	Example
Cust_ID	Integer	10	Primary Key	Customer ID	5
Addr_Type	Varchar	50	Primary Key	Address Types	Work
Street_Addr	Varchar	100	Nullable	Street Address	38745 CAMBRIDGE DR
City	Varchar	50	Nullable	City Name	ZEPHYRHILLS
State	Char	2	Nullable	State	FL
Zip	Integer	5	Nullable	Zip Code	33540

Ingredient Inventory:

The Ingredient Inventory comprises of all the Ingredients for the recipe of each menu item. It also maintains their current quantity in stock and the threshold quantity below which the ingredient is ordered for refill. The current quantity is updated automatically as in when a new order is placed by a customer. This is implemented by using an AFTER INSERT Trigger described in the later section.

Column Name	Data Type	Size	Constraint	Description	Example
INGREDIENT_ID	Integer	10	Primary Key	Ingredient ID	100
INGRED_NAME	Varchar	100	Nullable	Ingredient Name	Milk
CURR_QTY	Float		Nullable	Current Quantity	100
REFILL_THRESHOLD	Float		Nullable	Refill Threshold Quantity	20
MEASUREMENT_UNIT	Char	20	Nullable	State	Litre

Recipe:

The recipe table is used to map the menus to the required ingredients. As there is a many to many relationship between the menu items and the ingredients, using the normalization principles this table takes care of implementing the relation.

Column Name	Data Type	Size	Constraint	Description	Example
MENU_ID	Integer	10	Primary Key	Menu ID	15
INGRED_ID	Integer	10	Primary Key	Ingredient ID	1
QTY_USED	Float		Nullable	Quantity used for Menu ID	2

Menu:

The menu table comprises of the menu items available along with the corresponding unit price in dollars. Several parameters are defined to categorize the menu item.

Column Name	Data Type	Size	Constraint	Description	Example
MENU_ID	Integer	10	Primary Key	Menu ID	10
NAME	Varchar	100	Nullable	Item Name	Grilled Chicken
CAT_ID	Integer	10	Nullable	Meal Category Id	101
DESCRIPTION	Varchar	100	Nullable	Item Quantity Description	Full
PRICE_USD	Float		Nullable	Item price	10.0
OVERALL_RATING	Float		Nullable	Item Rating	7.5
FLAVOR	Varchar	50	Nullable	Flavor	Mild

Meal Category:

This is a lookup table which comprises of the food items categories like Appetizers, Sides, Beverages etc.

Column Name	Data Type	Size	Constraint	Description	Example
CAT_ID	Integer	10	Primary Key	Meal Category ID	104
CAT_NAME	Varchar	100	Nullable	Category Name	Desserts

Orders:

This is a transactional table which maintains the order records for a customer and comprises of the payment information and the order date. The total amount is populated using an AFTER INSERT trigger in the order details table. This is described in the later section.

Column Name	Data Type	Size	Constraint	Description	Example
ORDER_ID	Integer	20	Primary Key	Order ID	10001
CUST_ID	Integer	10	Yes	Customer ID	5
ORDER_DATE	Date		Nullable	Date of Order	16-NOV-17
TOTAL_AMT	Float		Nullable	Total Amount of the Order in Dollars	220.54
PAYMENT_MODE	Varchar	50	Nullable	Payment Mode	Credit Card
PAYMENT_DATE	Date		Nullable	Payment Date	16-NOV-17

Order Details:

An order comprises of number of items from the menu, on the other hand, a menu item can be a part of more than one orders which showcases a many-to-many relationship between orders and menu. Using the normalization principles, Order Details table is introduced to implement this relationship.

Column Name	Data Type	Size	Constraint	Description	Example
ORDER_ID	Integer	20	Primary Key	Order ID	10001
MENU_ID	Integer	10	Primary Key	Menu ID	1
QUANTITY	Integer	10	Nullable	Quantity ordered	3

Favorites:

Favorites table maintains the top three favorites of a customer. This is determined based on the best three ratings given by a customer to menu items.

Column Name	Data Type	Size	Constraint	Description	Example
CUST_ID	Integer	10	Primary Key	Customer ID	1
FAV_1	Integer	10	Nullable	1st Favorite Menu ID	1
FAV_2	Integer	10	Nullable	2nd Favorite Menu ID	2
FAV_3	Integer	10	Nullable	3rd Favorite Menu ID	3

Food Ratings:

This table has ratings of various menu items given by customers. Since there is a primary key relationship defined on customer ID and menu ID columns, only one entry is available for a customer and menu id combination.

Column Name	Data Type	Size	Constraint	Description	Example
CUST_ID	Integer	10	Primary Key	Customer ID	1
MENU_ID	Integer	10	Primary Key	Menu ID	1
RATINGS	Float		Nullable	Ratings	6.5
RATED_DT	Date		Nullable	Rated Date	12-OCT-16

Food Ratings History:

This table is populated by a trigger – BEFORE INSERT on Food Ratings table. If there exist a customer id and menu id entry matching the new entry, the old entry is deleted from the food ratings table and pushed in the history table.

Column Name	Data Type	Size	Constraint	Description	Example
HIST_ID	Integer	10	Primary Key	History ID	1
CUST_ID	Integer	10	Nullable	Customer ID	1
MENU_ID	Integer	10	Nullable	Menu ID	1
RATINGS	Float		Nullable	Ratings	6.5
LAST_RATED	Date		Nullable	Rated Date	12-OCT-16

CREATED_DT	Date		Default: Sysdate	Entry Created Date	19-NOV-17

Recommendation Dashboard:

This table maintains user preferences for recommendations. By default, a customer gets 10 recommendations of the items to be ordered. This number can be restricted as per the customer's choice and is maintained in this table.

Column Name	Data Type	Size	Constraint	Description	Example
CUST_ID	Integer	10	Nullable	Customer ID	1
REC_DISPLAY	Integer	10	Nullable	Number of recommendations to be displayed	3

D. Data Integrity:

This section describes the schema of few of the tables using the inline DDL used as well as the various types of constraints added.

Customer:

Schema:

```
--------------------------------------------------------
--  DDL for Table CUSTOMER
--------------------------------------------------------

  CREATE TABLE "DB233"."CUSTOMER"
   (
    "CUST_ID" NUMBER(10,0),
    "FNAME" VARCHAR2(50 BYTE) DEFAULT 'FNU',
    "LNAME" VARCHAR2(50 BYTE) DEFAULT 'LNU',
    "EMAIL" VARCHAR2(100 BYTE),
    "GENDER" VARCHAR2(10 BYTE),
    "BIRTH_DAY" NUMBER,
    "BIRTH_MONTH" NUMBER,
    "BIRTH_YEAR" NUMBER
   )
--------------------------------------------------------
--  DDL for Index PK_CUSTOMER
--------------------------------------------------------

  CREATE UNIQUE INDEX "DB233"."PK_CUSTOMER" ON "DB233"."CUSTOMER" ("CUST_ID");
--------------------------------------------------------
```

Constraints:

```
--------------------------------------------------------
--  Constraints for Table CUSTOMER
--------------------------------------------------------
  ALTER TABLE "DB233"."CUSTOMER" ADD CONSTRAINT "PK_CUSTOMER"
  PRIMARY KEY ("CUST_ID");
  ALTER TABLE "DB233"."CUSTOMER" MODIFY ("EMAIL" NOT NULL ENABLE);
  ALTER TABLE "DB233"."CUSTOMER" MODIFY ("GENDER" NOT NULL ENABLE);
```

Few highlights of this schema:

- The customer table is defined using a singleton primary key – CUST_ID.
- Default values are defined for First Name and Last Name columns as shown above.
- Not Null constraints are defined on email and gender columns.

Ingredient Inventory:

Schema:

```
--------------------------------------------------------
--  DDL for Table INGREDIENT_INVENTORY
--------------------------------------------------------
  CREATE TABLE "DB233"."INGREDIENT_INVENTORY"
   (    "INGREDIENT_ID" NUMBER(*,0),
    "INGRED_NAME" VARCHAR2(100 BYTE),
    "CURR_QTY" FLOAT(126),
    "REFILL_THRESHOLD" FLOAT(126),
    "MEASUREMENT_UNIT" VARCHAR2(20 BYTE)
   ) ;
--------------------------------------------------------
--  DDL for Index PK_INGREDIENT_INVENTORY
--------------------------------------------------------
  CREATE UNIQUE INDEX "DB233"."PK_INGREDIENT_INVENTORY"
  ON "DB233"."INGREDIENT_INVENTORY" ("INGREDIENT_ID") ;
--------------------------------------------------------
```

Constraints:

```
--------------------------------------------------------
--  Constraints for Table INGREDIENT_INVENTORY
--------------------------------------------------------
  ALTER TABLE "DB233"."INGREDIENT_INVENTORY" ADD CONSTRAINT "MAINTAIN_QTY"
  CHECK (CURR_QTY>=REFILL_THRESHOLD) ENABLE;
  ALTER TABLE "DB233"."INGREDIENT_INVENTORY"
  ADD CONSTRAINT "PK_INGREDIENT_INVENTORY" PRIMARY KEY ("INGREDIENT_ID");
```

Few highlights of the schema:

- This table comprises of the ingredients required for making the recipes listed in the menu.
- Singleton primary key – INGREDIENT_ID is defined with the Auto-Unique Index Creation.
- Check constraint is enabled to ensure that at any point of time the current quantity does not go below the refill threshold value defined.

Food Ratings:

Schema and Constraints:

```
CREATE TABLE "DB233"."FOOD_RATINGS"
 ("CUST_ID" NUMBER(*,0),
  "MENU_ID" NUMBER(*,0),
  "RATINGS" FLOAT(126),
  "RATED_DT" DATE,
   CONSTRAINT "PK_FOOD_RATINGS" PRIMARY KEY ("CUST_ID", "MENU_ID"),
   CONSTRAINT "FK_CUST_ID_RATINGS" FOREIGN KEY ("CUST_ID")
    REFERENCES "DB233"."CUSTOMER" ("CUST_ID") ENABLE,
   CONSTRAINT "FK_MENU_MENU_ID" FOREIGN KEY ("MENU_ID")
    REFERENCES "DB233"."MENU" ("MENU_ID") ENABLE
 );
```

Few highlights of the table:

- This table comprises of the ratings of customers provided for various items in the menu.
- Combined columns are used to define primary key i.e. CUST_ID and MENU_ID
- Referential integrity is maintained by having foreign keys defined on both CUST ID and MENU ID columns, referential tables being CUSTOMER and MENU respectively.

Checking Constraint Status:

DBA query to check the constraints that are currently enabled for the system is highlighted below:

```
select CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME, R_CONSTRAINT_NAME, STATUS from user_constraints where status = 'ENABLED';
```

Query Result | Query Result 1

SQL | Fetched 50 rows in 3.749 seconds

	CONSTRAINT_NAME	CONSTRAINT_TYPE	TABLE_NAME	R_CONSTRAINT_NAME	STATUS
1	FK_CAT_ID_MENU	R	MENU	PK_MEAL_CATEGORY	ENABLED
2	FK_FAV_3_MENU	R	FAVORITES	PK_MENU	ENABLED
3	FK_FAV_2_MENU	R	FAVORITES	PK_MENU	ENABLED
4	FK_FAV_1_MENU	R	FAVORITES	PK_MENU	ENABLED
5	FK2_MENU_ID_MENU	R	ORDER_DETAILS	PK_MENU	ENABLED
6	FK1_MENU_ID_MENU	R	RECIPE	PK_MENU	ENABLED
7	FK_CUST_ID_RATINGS	R	FOOD_RATINGS	PK_CUSTOMER	ENABLED
8	FK_CUST_ID_CUSTOMER	R	ORDERS	PK_CUSTOMER	ENABLED
9	FK_CUST_ID_ADDR	R	CUSTOMER_ADDRESS	PK_CUSTOMER	ENABLED
10	FK1_ORDER_ID_ORDERS	R	ORDER_DETAILS	PK_ORDERS	ENABLED
11	FK2_INGRED_ID_INVENTORY	R	RECIPE	PK_INGREDIENT_INVENTORY	ENABLED
12	SYS_C0070202	C	CUSTOMER	(null)	ENABLED
13	SYS_C0070201	C	CUSTOMER	(null)	ENABLED
14	SYS_C0070147	P	FOOD_RECO	(null)	ENABLED
15	PK_RECIPE	P	RECIPE	(null)	ENABLED
16	PK_ORDER_DETAILS	P	ORDER_DETAILS	(null)	ENABLED
17	PK_ORDERS	P	ORDERS	(null)	ENABLED
18	PK_MENU	P	MENU	(null)	ENABLED
19	PK_MEAL_CATEGORY	P	MEAL_CATEGORY	(null)	ENABLED

Explanation:

This query fetches all the constraints that are currently enabled and give the details of the constraint type. For instance, constraint type – 'R' denotes referential constraint (foreign key) and also highlights the corresponding referential column and table details.

E. Data Generation and Loading:

To create Bulls Eat dataset we have used various different methods that will be highlighted in this section:

Populating the ID column:

Many tables like menu, ingredient inventory, orders etc. have primary key defined on a singleton column which should be an **auto-generated unique number**.

This is achieved by:

1. Creating a sequence with start value and incremented by value.
2. Creating a BEFORE INSERT trigger to populate the ID value using the sequence.

This is shown below for orders table:

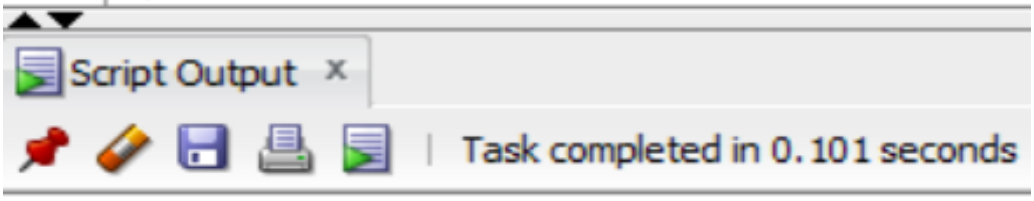
```
-- SEQUENCE CREATED TO POPULATE ORDER ID --
CREATE SEQUENCE  "DB233"."ORDERS_SEQUENCE"
MINVALUE 10001 MAXVALUE 1000000000
INCREMENT BY 1 START WITH 17661;
--------------------------------------------------------
--  DDL for Trigger TR_POPULATE_ORDER_ID
--------------------------------------------------------
CREATE OR REPLACE TRIGGER "DB233"."TR_POPULATE_ORDER_ID"
BEFORE INSERT ON ORDERS
FOR EACH ROW
BEGIN
  SELECT ORDERS_SEQUENCE.NEXTVAL
  INTO   :new.order_id
  FROM   dual;
END;
/
```

Here, the next value of the ID will be populated as 17661. As before inserting in the Orders Table, the trigger picks the next value of the sequence and populates the ORDER_ID with the same.

Data generation using Hacks:

1. Populating Flavor column using Mod function:

```
UPDATE MENU
SET FLAVOR = 'HOT'
where mod(menu_id,19) =0;
```




```
10 rows updated.
```

2. Populating Payment Date column using Date Arithmetic:

```
UPDATE ORDERS
SET PAYMENT_DATE = SYSDATE - CUST_ID - 10,
ORDER_DATE = SYSDATE - CUST_ID - 10;
commit;
select * from orders fetch first 10 rows only;
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.027 seconds

	ORDER_ID	CUST_ID	ORDER_DATE	TOTAL_AMT	PAYMENT_MODE	PAYMENT_DATE
1	10001	1	08-NOV-17	36.57	Paypal	08-NOV-17
2	10002	3	06-NOV-17	58.84	Cash	06-NOV-17
3	10004	5	04-NOV-17	2.98	Credit Card	04-NOV-17
4	10005	7	02-NOV-17	220.54	Cash	02-NOV-17
5	10006	8	01-NOV-17	13.78	Credit Card	01-NOV-17
6	10007	10	30-OCT-17	10.29	Paypal	30-OCT-17
7	10008	12	28-OCT-17	33.06	Cash	28-OCT-17
8	10009	13	27-OCT-17	28.47	Paypal	27-OCT-17
9	10010	14	26-OCT-17	32.54	Credit Card	26-OCT-17
10	10011	15	25-OCT-17	45.35	Cash	25-OCT-17

3. Populating tables from existing tables:

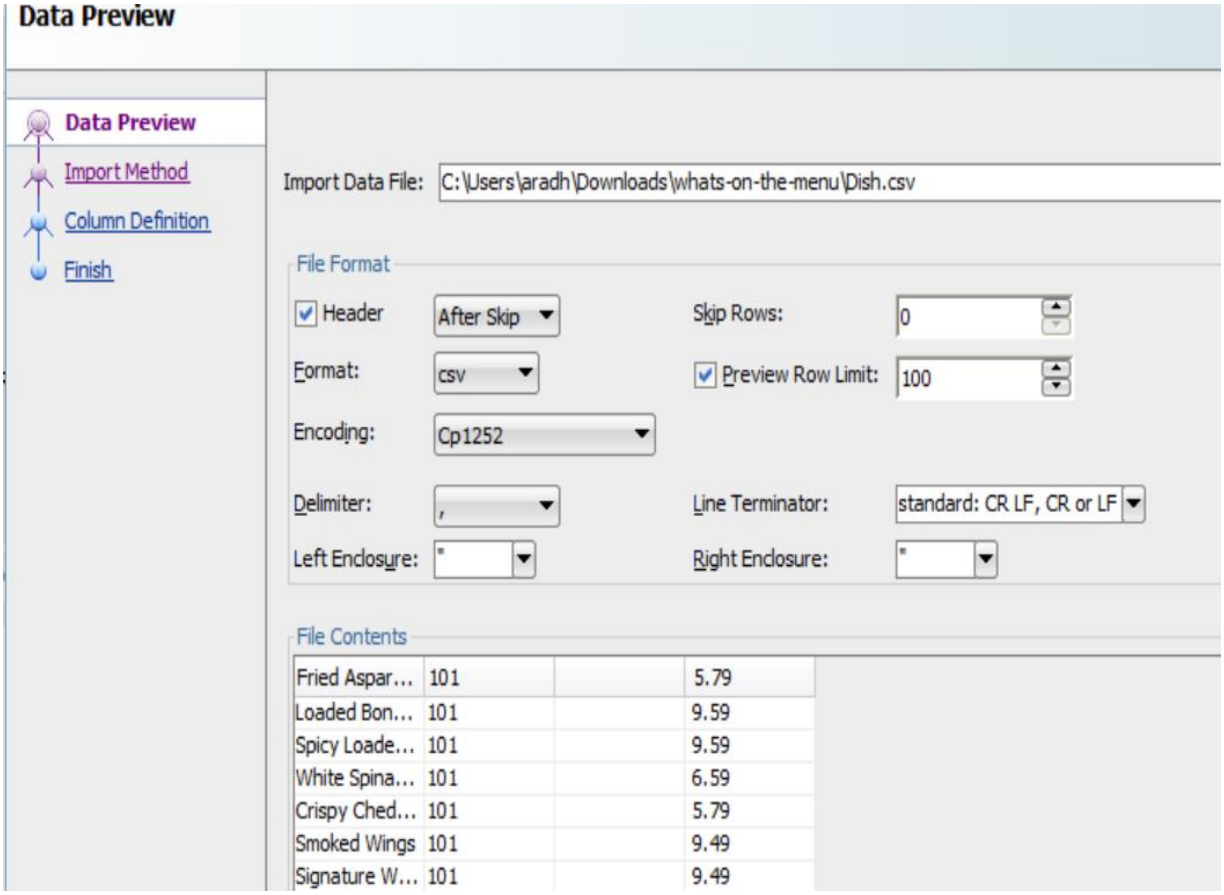
```
insert into customer
select fan_id as cust_id, fname, lname, email,gender,
birth_day, birth_month, birth_year from fans;
commit;
```

4. Populating tables using Random Values:

```
insert into order_details
select round(dbms_random.value(10001,17760)) ORDER_ID,
       round(dbms_random.value(1,200)) MENU_ID ,
       round(dbms_random.value(1,5)) QUANTITY
from dual;
```

Populated data using MS Excel:

The data was generated in csv format using www.mockaroo.com



Populated data using complex query:

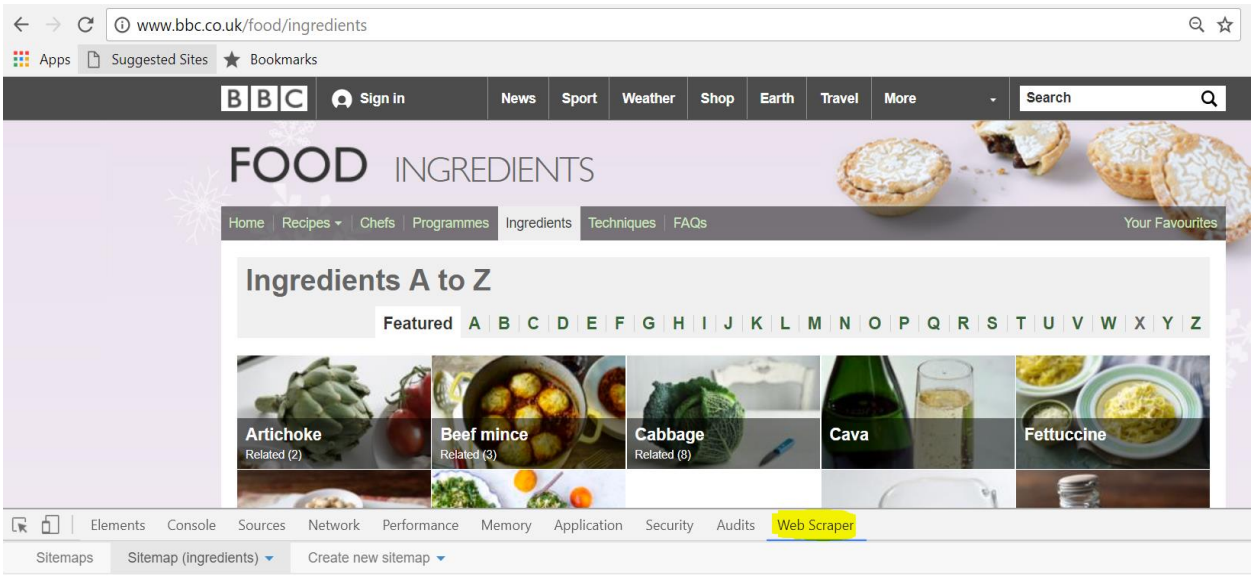
```
update orders t1
set total_amt = (select sum(total)
                 from
                     (select o.order_id ODR, m.menu_id,quantity*price_usd total
                      from orders o
                      inner join order_details od
                      on o.ORDER_ID = od.ORDER_ID
                      inner join menu m
                      on od.menu_id=m.menu_id )o
                      where t1.ORDER_ID = o.ODR
                 group by o.ODR);
```

Populating data using insert query:

```
set define off;
set verify off;

insert into recipe (menu_id, ingred_id, qty_used)
values (15, 1, 2);

insert into recipe (menu_id, ingred_id, qty_used)
values (15, 2, 5);

insert into recipe (menu_id, ingred_id, qty_used)
values (15, 3, 2);

insert into recipe (menu_id, ingred_id, qty_used)
values (15, 36, 1);
```

Populated data using Web Scraper – Chrome Extension:

Ingredients dataset was created using Web Scraper extension available for chrome browser to fetch the ingredients name from www.bbc.co.uk/food/ingredients



letters	letters-href	ing_element
E	http://www.bbc.co.uk/food/ingredients/by/letter/e	Exotic fruit
C	http://www.bbc.co.uk/food/ingredients/by/letter/c	Chocolate cake
L	http://www.bbc.co.uk/food/ingredients/by/letter/l	Lamb shoulder
M	http://www.bbc.co.uk/food/ingredients/by/letter/m	Macadamia
B	http://www.bbc.co.uk/food/ingredients/by/letter/b	Breadcrumbs
P	http://www.bbc.co.uk/food/ingredients/by/letter/p	Pumpernickel bread

Summary of Implemented Tables:

Table Name	No. Of Rows
CUSTOMER	11655
CUSTOMER_ADDRESS	17658
FAVORITES	11655
FOOD_RATINGS	30152
INGREDIENT_INVENTORY	117
MENU	200
MEAL_CATEGORY	16
ORDERS	7655
ORDER_DETAILS	12044
RECIPE	536
RECOMMENDATION_DASHBOARD	11655

Part 2 - Query Writing:

The system was queried to test the capability of answering different types of questions.

#1 Finding the Top 10 most liked menu items:

```
SELECT Y.NAME, X.TOTAL_LIKES
  FROM (SELECT *
          FROM (SELECT MENU_ID,SUM(F1CNT) + SUM(F2CNT) + SUM(F3CNT) TOTAL_LIKES
                  FROM (SELECT FAV_1 MENU_ID, COUNT(DISTINCT CUST_ID) F1CNT,
                               0 AS F2CNT,0 AS F3CNT
                          FROM FAVORITES GROUP BY FAV_1
                          UNION
                          SELECT FAV_2 MENU_ID, 0 AS F1CNT,
                                 COUNT(DISTINCT CUST_ID) F2CNT,0 AS F3CNT
                          FROM FAVORITES GROUP BY FAV_2
                          UNION
                          SELECT FAV_3 MENU_ID,0 AS F1CNT,
                                 0 AS F2CNT, COUNT(DISTINCT CUST_ID) F3CNT
                          FROM FAVORITES GROUP BY FAV_3)
                 WHERE MENU_ID IS NOT NULL
                 GROUP BY MENU_ID)
        ORDER BY TOTAL_LIKES DESC) X
 INNER JOIN MENU Y
    ON X.MENU_ID = Y.MENU_ID
 WHERE ROWNUM <= 10;
```

	NAME	TOTAL_LIKES
1	Mix & Match Fajitas	145
2	Mango-Chile Tilapia	142
3	Wings	137
4	Spicy Shrimp Tacos	137
5	Chili & House Salad	136
6	Skillet Toffee Fudge Brownie	136
7	Cheese Enchiladas	135
8	Classic Sirloin with Grilled Avocado (cal. 410)	134
9	Diet Coke	133
10	Ancho Salmon	132

#2 Finding the Top 10 most ordered menu items EVER

```
SELECT * FROM
          (SELECT * FROM
                (SELECT M.NAME, COUNT(DISTINCT ORD.ORDER_ID) TOTAL_ORDERS
                  FROM MENU M
                 INNER JOIN ORDER_DETAILS OD
                    ON M.MENU_ID = OD.MENU_ID
                 INNER JOIN ORDERS ORD
                    ON ORD.ORDER_ID = OD.ORDER_ID
                 GROUP BY M.NAME)
            ORDER BY TOTAL_ORDERS DESC)
WHERE ROWNUM <= 10;
```

Query Result 2 ×

SQL | All Rows Fetched: 10 in 0.049 seconds

	NAME	TOTAL_ORDERS
1	Terlingua Chili	178
2	Margarita Grilled Chicken	171
3	Big Mouth Bites	142
4	Craft Beer BBQ Baby Back Ribs	137
5	Cajun Chicken Pasta	137
6	Strawberry Lemonade	135
7	Mango-Chile Chicken	130
8	Dr Pepper BBQ Baby Back Ribs	129
9	Quesadilla Explosion Salad	128
10	Classic Sirloin	127

#3 Finding the Top 10 most ordered menu item of the LAST month

```
SELECT *
  FROM (SELECT *
        FROM (SELECT M.NAME, COUNT(DISTINCT ORD.ORDER_ID) TOTAL_ORDERS
              FROM MENU M
             INNER JOIN ORDER_DETAILS OD
                ON M.MENU_ID = OD.MENU_ID
             INNER JOIN ORDERS ORD
                ON ORD.ORDER_ID = OD.ORDER_ID
             WHERE ORD.ORDER_DATE BETWEEN
                 ADD_MONTHS(TRUNC(SYSDATE, 'MM'), -1) AND
                 TRUNC(SYSDATE, 'MM') - 1
             GROUP BY M.NAME)
       ORDER BY TOTAL_ORDERS DESC)
 WHERE ROWNUM <= 10;
```

	NAME	TOTAL_ORDERS
1	Santa Fe Salad	2
2	Mix & Match Fajita Trio	2
3	Spicy Loaded Boneless Wings	2
4	Sprite	2
5	Prime Rib Tacos	2
6	Honey-Chipotle Shrimp & Sirloin	2
7	Buffalo Chicken Ranch Sandwich	1
8	Classic Nachos	1
9	Cheese Enchiladas	1
10	Classic Turkey Toasted Sandwich	1

#4 Finding the Top 10 most consumed ingredients EVER

```
SELECT *
  FROM (SELECT INGRED_ID,
               INGRED_NAME,
               QTY_USED,
               ROW_NUMBER() OVER(PARTITION BY NULL ORDER BY QTY_USED DESC) RNK
          FROM (SELECT R.INGRED_ID, I.INGRED_NAME, SUM(R.QTY_USED) QTY_USED
                  FROM RECIPE R
                 INNER JOIN INGREDIENT_INVENTORY I
                    ON R.INGRED_ID = I.INGREDIENT_ID
                 GROUP BY R.INGRED_ID, I.INGRED_NAME))
 WHERE RNK <= 10;
```

	INGRED_ID	INGRED_NAME	QTY_USED	RNK
1	3	Olive Oil	130	1
2	2	Salt	40	2
3	9	Ribs	38	3
4	8	Papparoni	36	4
5	80	Mango	35	5
6	11	Spinach	32	6
7	61	Cheese	31	7
8	14	Green Chillis	30	8
9	38	Soya Sauce	30	9
10	22	Brown Pepper	27	10

#5 Finding the Top 10 most consumed ingredient TODAY

```
SELECT *
  FROM (SELECT INGRED_NAME
        FROM (SELECT DISTINCT ORI.INGRED_NAME, ORI.QTY_USED
              FROM ORDERS ORD
             INNER JOIN (SELECT RI.MENU_ID,
                                RI.INGRED_ID,
                                RI.INGRED_NAME,
                                RI.QTY_USED,
                                OD.ORDER_ID
                           FROM ORDER_DETAILS OD
                          INNER JOIN (SELECT R.MENU_ID,
                                            R.INGRED_ID,
                                            I.INGRED_NAME,
                                            R.QTY_USED
                                       FROM RECIPE R
                                      INNER JOIN INGREDIENT_INVENTORY I
                                         ON R.INGRED_ID = I.INGREDIENT_ID) RI
                             ON OD.MENU_ID = RI.MENU_ID) ORI
                ON ORD.ORDER_ID = ORI.ORDER_ID
             WHERE ORDER_DATE BETWEEN TRUNC(SYSDATE) AND SYSDATE
             ORDER BY ORI.QTY_USED DESC))
 WHERE ROWNUM <= 10;
```

	INGRED_NAME
1	Anchovy
2	Apple
3	Bacon
4	Beef
5	Brocholli
6	Brown Pepper
7	Cabbage
8	Cheese
9	Chips
10	Green Chillis

#6 Finding the Most Consumed Category

```
SELECT CAT_NAME FROM (
SELECT DISTINCT CAT_ID
  FROM (SELECT *
        FROM (SELECT M.NAME, M.CAT_ID, COUNT(DISTINCT ORD.ORDER_ID) TOTAL_ORDERS
              FROM MENU M
             INNER JOIN ORDER_DETAILS OD
                ON M.MENU_ID = OD.MENU_ID
             INNER JOIN ORDERS ORD
                ON ORD.ORDER_ID = OD.ORDER_ID
             GROUP BY M.NAME, M.CAT_ID)
       ORDER BY TOTAL_ORDERS DESC)
 WHERE ROWNUM <= 10
 ) X
 INNER JOIN MEAL_CATEGORY Y
 ON X.CAT_ID = Y.CAT_ID;
```

	CAT_NAME
1	Sides
2	Salads, Soups and Chilis
3	Ribs and Steaks
4	Beverages

#7 Finding the Top 10 Highest Billed Orders

```
SELECT *
  FROM (SELECT * FROM ORDERS ORDER BY TOTAL_AMT DESC)
 WHERE ROWNUM <= 10;
```

	ORDER_ID	CUST_ID	ORDER_DATE	TOTAL_AMT	PAYMENT_MODE	PAYMENT_DATE
1	14556	3793	30-06-07	494.69	Cash	30-06-07
2	16317	6028	17-05-01	441.53	Cash	17-05-01
3	11622	2969	01-10-09	433.38	Cash	01-10-09
4	14316	7627	30-12-96	426.74	Cash	30-12-96
5	13260	5016	23-02-04	424.28	Cash	23-02-04
6	12606	5829	02-12-01	422.73	Cash	02-12-01
7	16900	4686	18-01-05	414.72	Credit Card	18-01-05
8	10830	1458	20-11-13	399.8	Cash	20-11-13
9	14339	5646	03-06-02	398.73	Paypal	03-06-02
10	11754	4223	26-04-06	395.82	Cash	26-04-06

#8 Finding the Top 10 repeating customers

```
SELECT CUST_ID,
       FNAME || ' ' || LNAME AS CUSTOMER_NAME,
       EMAIL,
       GENDER,
       BIRTH_DAY || '-' || BIRTH_MONTH || '-' || BIRTH_YEAR AS DOB
  FROM CUSTOMER
 WHERE CUST_ID IN (SELECT CUST_ID
                     FROM (SELECT *
                             FROM (SELECT CUST_ID, COUNT(*) NO_OF_ORDERS
                                     FROM ORDERS
                                    GROUP BY CUST_ID
                                   HAVING COUNT(*) > 1)
                            ORDER BY NO_OF_ORDERS DESC)
                    WHERE ROWNUM <= 10);
```

	CUST_ID	CUSTOMER_NAME	EMAIL	GENDER	DOB
1	78	MARI WILMOTH	mari.wilmoth@ya.ru	F	26-7-1954
2	70	THOMAS EZELL	thomas.ezell@google.com	M	29-10-1947
3	1113	RICHARD ROGERS	richard.rogers@yahoo.co.in	M	7-5-1964
4	94	MATTHEW ALMAND	matthew.almand@aol.com	M	28-5-1962
5	879	BOBBY MCMURTRY	bobby.mcmurtry@talktalk.co.uk	M	11-12-1976
6	5681	JERRY BOLES	jerry.boles@ntlworld.com	M	24-7-1962
7	434	JOHN EALUM	john.ealum@oi.com.br	M	9-2-1951
8	1146	MACARTHUR STEPP	macarthur.stepp@lavabit.com	M	6-3-1947
9	1078	WILLIAM GROOM	william.groom@laposte.net	M	10-5-1956
10	7512	JOHN STYER	john.styer@ymail.com	M	7-10-1950

#9 Finding Category wise total sales for last month

```
SELECT *
  FROM (SELECT MC.CAT_NAME, SUM(ORD.TOTAL_AMT) TOT_SALES
          FROM MEAL_CATEGORY MC
         INNER JOIN MENU M
            ON MC.CAT_ID = M.CAT_ID
         INNER JOIN ORDER_DETAILS OD
            ON M.MENU_ID = OD.MENU_ID
         INNER JOIN ORDERS ORD
            ON OD.ORDER_ID = ORD.ORDER_ID
         WHERE ORD.ORDER_DATE BETWEEN ADD_MONTHS(TRUNC(SYSDATE, 'MM'), -1) AND
               TRUNC(SYSDATE, 'MM') - 1
         GROUP BY MC.CAT_NAME)
 ORDER BY 2 DESC;
```

	CAT_NAME	TOT_SALES
1	Party Platters	749.94
2	Appetizers	399.91
3	Tacos and Quesadillas	209.57
4	Fajitas and Enchiladas	179.32
5	Beverages	158.43
6	Ribs and Steaks	150.98
7	Salads, Soups and Chilis	87.42
8	Craft Burgers	85.58
9	Sandwiches and Handhelds	47.51
10	Kids Menu	44.82
11	Burritos	43.56
12	Lighter Choices	23.07
13	Sides	22.48
14	Fresh Mex Bowls	15.59

#10 Finding Category & Monthwise Sales

```
SELECT MC.CAT_NAME,
       TO_CHAR(ORD.ORDER_DATE, 'MON') MONTH,
       SUM(ORD.TOTAL_AMT) TOT_SALES
  FROM MEAL_CATEGORY MC
 INNER JOIN MENU M
    ON MC.CAT_ID = M.CAT_ID
 INNER JOIN ORDER_DETAILS OD
    ON M.MENU_ID = OD.MENU_ID
 INNER JOIN ORDERS ORD
    ON OD.ORDER_ID = ORD.ORDER_ID
 GROUP BY MC.CAT_NAME, TO_CHAR(ORD.ORDER_DATE, 'MON');
```

	CAT_NAME	MONTH	TOT_SALES
1	Appetizers	Jan	6335.35
2	Appetizers	Mar	5766.32
3	Craft Burgers	May	2307.07
4	Lighter Choices	Nov	2212.23
5	Lighter Choices	Dec	2560.21
6	Lighter Choices	Apr	2363.94
7	Fajitas and Enchiladas	Feb	2386.31
8	Fajitas and Enchiladas	Nov	2912.36
9	Fresh Mex Bowls	Sep	2323.68
10	Fresh Mex Bowls	Dec	4753.3
11	Tacos and Quesadillas	Jul	2371.18
12	Tacos and Quesadillas	Aug	3928.53
13	Tacos and Quesadillas	Oct	4032.3
14	Ribs and Steaks	Oct	4762.97
15	Ribs and Steaks	Jun	5489.97
16	Chicken and Seafood	Oct	3260.84
17	Chicken and Seafood	Mar	3725.62
18	Chicken and Seafood	Aug	3093.37
19	Sides	Jul	3763.8

Database Programming:

TRIGGER
TR_UPDATE_CURR_QTY_INVENTORY
This trigger update and/or insert the CURR_QTY after an insert into the ORDER_DETAILS table.

```
CREATE OR REPLACE TRIGGER "TR_UPDATE_CURR_QTY_INVENTORY"
AFTER INSERT ON ORDER_DETAILS
FOR EACH ROW
BEGIN
MERGE INTO INGREDIENT_INVENTORY I1
USING
(
SELECT INGREDIENT_ID, CURR_QTY - USED LEFTOUT
FROM INGREDIENT_INVENTORY I
INNER JOIN
        (SELECT INGRED_ID, QTY_USED*QUANTITY USED
         FROM ORDER_DETAILS O INNER JOIN RECIPE R
         ON O.MENU_ID = R.MENU_ID
         WHERE O.MENU_ID = :new.MENU_ID AND ORDER_ID = :new.ORDER_ID
        )U ON I.INGREDIENT_ID = U.INGRED_ID
)I2
ON(I1.INGREDIENT_ID = I2.INGREDIENT_ID)
WHEN MATCHED THEN UPDATE
SET I1.CURR_QTY = I2.LEFTOUT;
END;
/
ALTER TRIGGER "TR_UPDATE_CURR_QTY_INVENTORY" ENABLE;
```

DBMS_SCHEDULER Job: (Not Implemented due to insufficient privileges)
This job aims at running every minute and assess the current stock. If threshold is breached, it will execute the stored procedure prc_alert_notifier described in the next point.

```
begin
  begin
    dbms_scheduler.drop_job(job_name => 'JOB_THRESHOLD_MONITOR');
  exception
    when others then
      null;
  end;
  dbms_scheduler.create_job(job_name        => 'JOB_THRESHOLD_MONITOR',
                            job_type        => 'PLSQL_BLOCK',
                            job_action      => 'BEGIN
                prc_alert_notifier;
                END;',
                            auto_drop       => FALSE,
                            start_date      => (sysdate + 1),
                            repeat_interval => 'FREQ=DAILY;',
                            enabled         => TRUE);
end;
/
```

#1 STORED PROCEDURE:

The below stored procedure will be used to send system-generated email notification to the admin in case the current quantity of an ingredient falls below its allowed threshold.

```
create or replace procedure prc_email(from_name varchar2,
                                       to_name   varchar2,
                                       subject   varchar2,
                                       message   varchar2)
is
  l_mailhost  VARCHAR2(64) := 'bullseat.com';
  l_from      VARCHAR2(64) := from_name;
  l_to        VARCHAR2(64) := to_name;
  l_mail_conn UTL_SMTP.connection;
BEGIN
  l_mail_conn := UTL_SMTP.open_connection(l_mailhost, 25);
  UTL_SMTP.helo(l_mail_conn, l_mailhost);
  UTL_SMTP.mail(l_mail_conn, l_from);
  UTL_SMTP.rcpt(l_mail_conn, l_to);
  UTL_SMTP.open_data(l_mail_conn);
  UTL_SMTP.write_data(l_mail_conn,
                      'Date: ' ||
                      TO_CHAR(SYSDATE, 'DD-MON-YYYY HH24:MI:SS') || Chr(13));
  UTL_SMTP.write_data(l_mail_conn, 'From: ' || l_from || Chr(13));
  UTL_SMTP.write_data(l_mail_conn, 'Subject: ' || subject || Chr(13));
  UTL_SMTP.write_data(l_mail_conn, 'To: ' || l_to || Chr(13));
  UTL_SMTP.write_data(l_mail_conn, '' || Chr(13));
  UTL_SMTP.write_data(l_mail_conn,message);
  UTL_SMTP.close_data(l_mail_conn);
  UTL_SMTP.quit(l_mail_conn);
END;
/
```

#2 STORED PROCEDURE:

This stored procedure will monitor the ingredient stock and throw alert by invoking the prc_email stored procedure when conditions are met.

```
CREATE OR REPLACE PROCEDURE prc_alert_notifier AS
BEGIN
  FOR i IN (SELECT ingred_name,
                   curr_qty,
                   measurement_unit,
                   refill_threshold
              from ingredient_inventory
             where curr_qty <= refill_threshold
             order by ingred_name) loop
    prc_email(from_name => 'system@bullseat.com',
              to_name   => 'admin@bullseat.com',
              subject   => '**Inventory Alert**',
              message   => 'Ingredient ' || i.ingred_name ||
                           ' is below finishing fast.' || chr(13) ||
                           'Refill');
  END LOOP;
END;
/
```

Part 3: Performance Tuning:

#1 Experiment with Independent Index v/s Composite Index

Purpose: The customer table has approximately 11k records and is queried on FNAME and LNAME to see the impact of Independent Index and Composite Index.

Steps:

1. Checking the execution plan for the query without any indexes.

```
select count(*) from customer
where fname = 'DAVID' and lname='FLOYD';
```

Script Output | Query Result | Autotrace | Autotrace 1 | Autotrace 2

SQL HotSpot | 1.835 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				30
SORT		AGGREGATE	1	
TABLE ACCESS	CUSTOMER	FULL	1	30
Filter Predicates				
AND				
LNAME='FLOYD'				
FNAME='DAVID'				

V$STATNAME Name	V$MYSTAT Value
consistent gets	114

2. Creating independent index on LNAME and checking the execution plan.

```
CREATE INDEX IDX_LNAME_CUSTOMER ON CUSTOMER(LNAME);
select count(*) from customer
where fname = 'DAVID' and lname='FLOYD';
```

Script Output | Query Result | Autotrace | Autotrace 1 | Autotrace 2

SQL HotSpot | 1.864 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				3
SORT		AGGREGATE	1	
TABLE ACCESS	CUSTOMER	BY INDEX ROWID BATCHED	1	3
Filter Predicates				
FNAME='DAVID'				
INDEX	IDX_LNAME_CUSTOMER	RANGE SCAN	1	1
Access Predicates				
LNAME='FLOYD'				

V$STATNAME Name	V$MYSTAT Value
consistent gets	9

3. Creating independent index on FNAME and LNAME and checking the execution plan.

```
CREATE INDEX IDX_LNAME_CUSTOMER ON CUSTOMER(LNAME);
CREATE INDEX IDX_FNAME_CUSTOMER ON CUSTOMER(FNAME);
select count(*) from customer
where fname = 'DAVID' and lname='FLOYD';
```

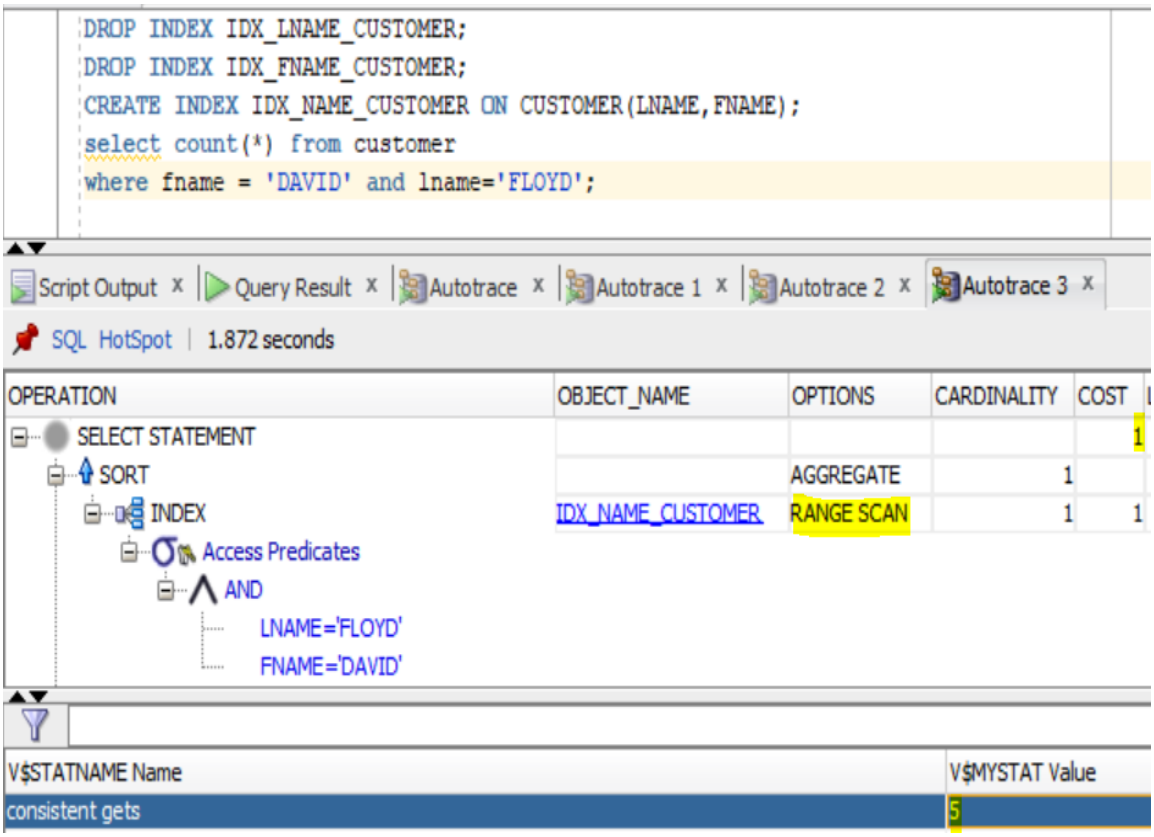
Script Output | Query Result | Autotrace | Autotrace 1 | Autotrace 2 | Autotrace 3

SQL HotSpot | 1.867 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				2
SORT		AGGREGATE	1	
BITMAP CONVERSION		COUNT	1	2
BITMAP AND				
BITMAP CONVERSION		FROM ROWIDS		
INDEX	IDX_LNAME_CUSTOMER	RANGE SCAN	1	1
Access Predicates				
LNAME='FLOYD'				
BITMAP CONVERSION		FROM ROWIDS		
INDEX	IDX_FNAME_CUSTOMER	RANGE SCAN	1	1
Access Predicates				
FNAME='DAVID'				

V$STATNAME Name	V$MYSTAT Value
consistent gets	7

4. Dropped independent index on FNAME and LNAME. Created composite index on (FNAME, LNAME) and checked the execution plan.



Explanation:

The Query execution cost reduces by using Composite index on (FNAME, LNAME) over Independent indexes on FNAME and LNAME. The summary statistics of the experiment is shown below:

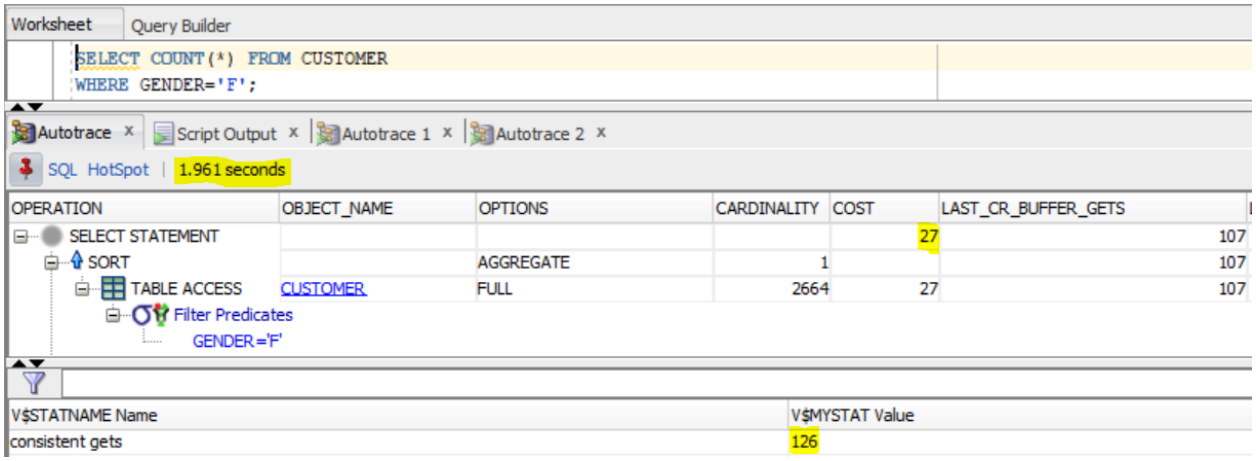
Query: SELECT COUNT (*) FROM CUSTOMER WHERE FNAME='DAVID' AND LNAME='FLOYD'		
	Consistent Gets	Query Cost
Without Index	114	30
One Index on LNAME	9	3
Two Independent Indexes on FNAME, LNAME	7	2
Composite Index on (FNAME, LNAME)	5	1

#2 Experiment with B Tree Index v/s Bitmap Index

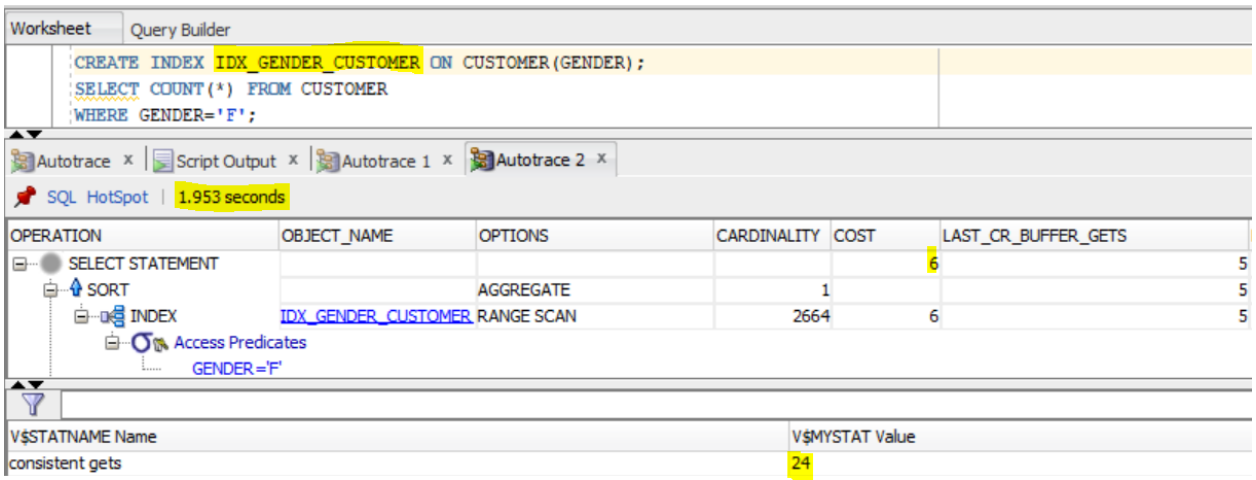
Purpose: The customer table has approximately 11k records and is queried on GENDER to check impact of B Tree Index and Bitmap Index.

Steps:

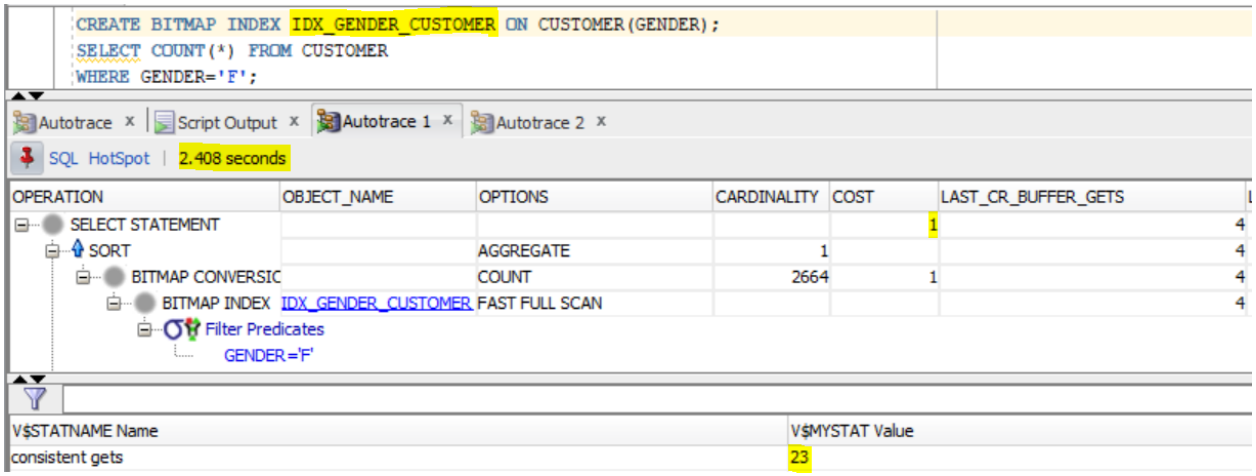
1. Checking the execution plan for the query without any indexes.



2. Checking the execution plan for the query with B Tree index on Gender.



3. Creating Bitmap Index on Gender and checking the execution plan.



Explaination:

We can see that the cost reduces significantly by using Bitmap Index over using B Tree Index as the cardinality of Gender column is low.

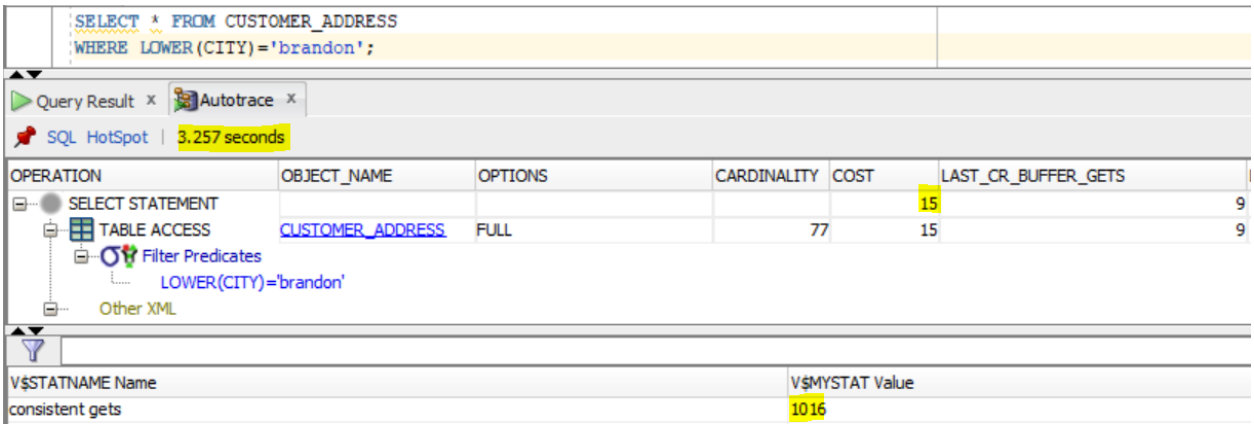
Query: SELECT COUNT (*) FROM CUSTOMER WHERE GENDER='F';			
	Consistent Gets	Execution Time	Query Cost
Without Index	126	1.961 seconds	27
With B Tree Index on Gender	24	1.953 seconds	6
With Bitmap Index on Gender	23	2.408 seconds	1

#3 Experiment with Functional Index

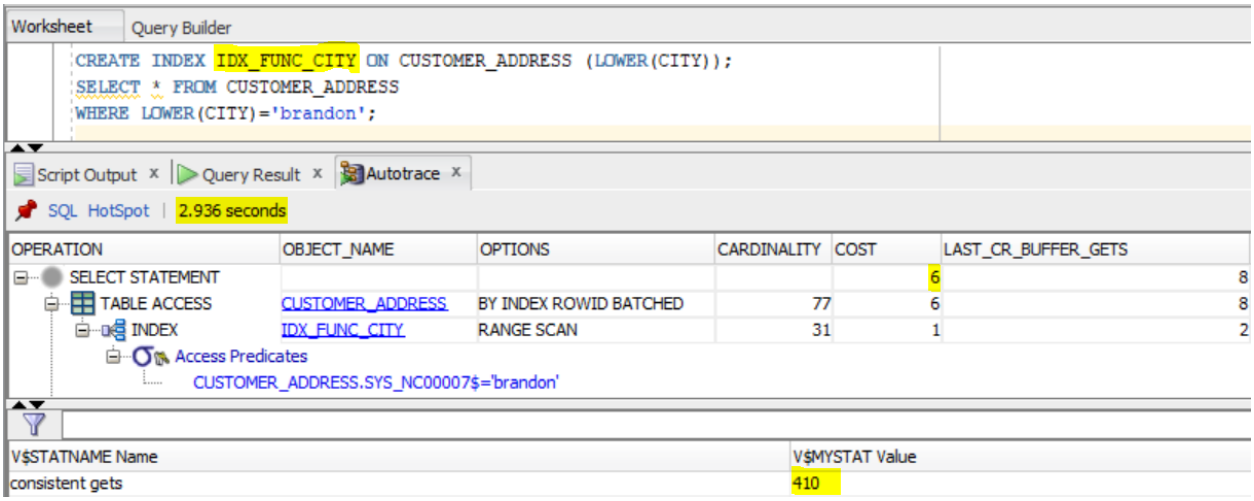
Purpose: To query the Customer Address table with city name in lower case and check the performance after using functional index on Lower(City).

Steps:

1. Checking the execution plan for the query without any indexes.



2. Creating Functional index on City to allow indexing on lowercase names and checking the execution plan.



Explaination:

We could see that functional index on the city column improves the cost by allowing indexing on the city names when queried in lower case.

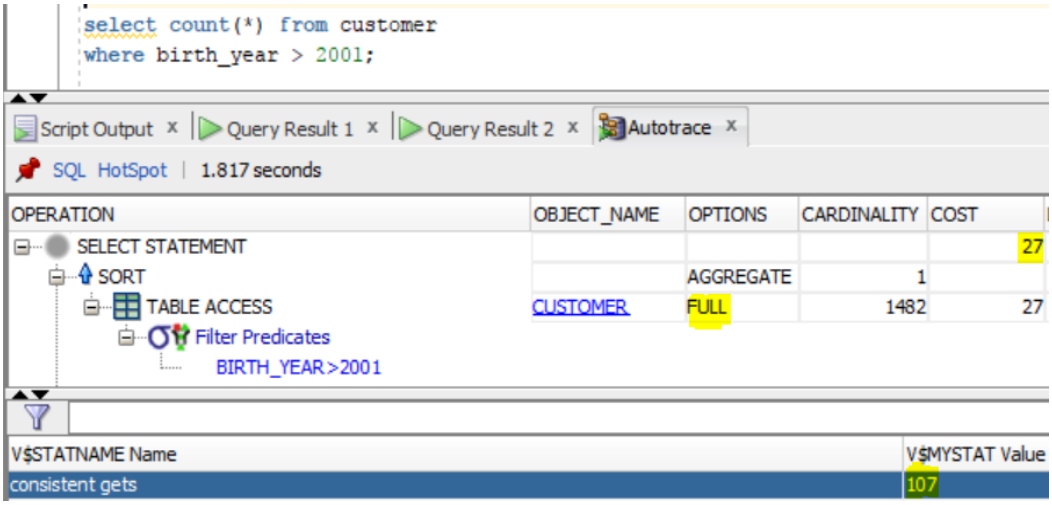
Query: SELECT * FROM CUSTOMER_ADDRESS WHERE LOWER(CITY)='brandon';			
	Consistent Gets	Execution Time	Query Cost
Without Index	1016	3.257 seconds	15
With Functional Index Lower(CITY)	410	2.936 seconds	6

#4 Experiment with Index v/s Partitioned Tables

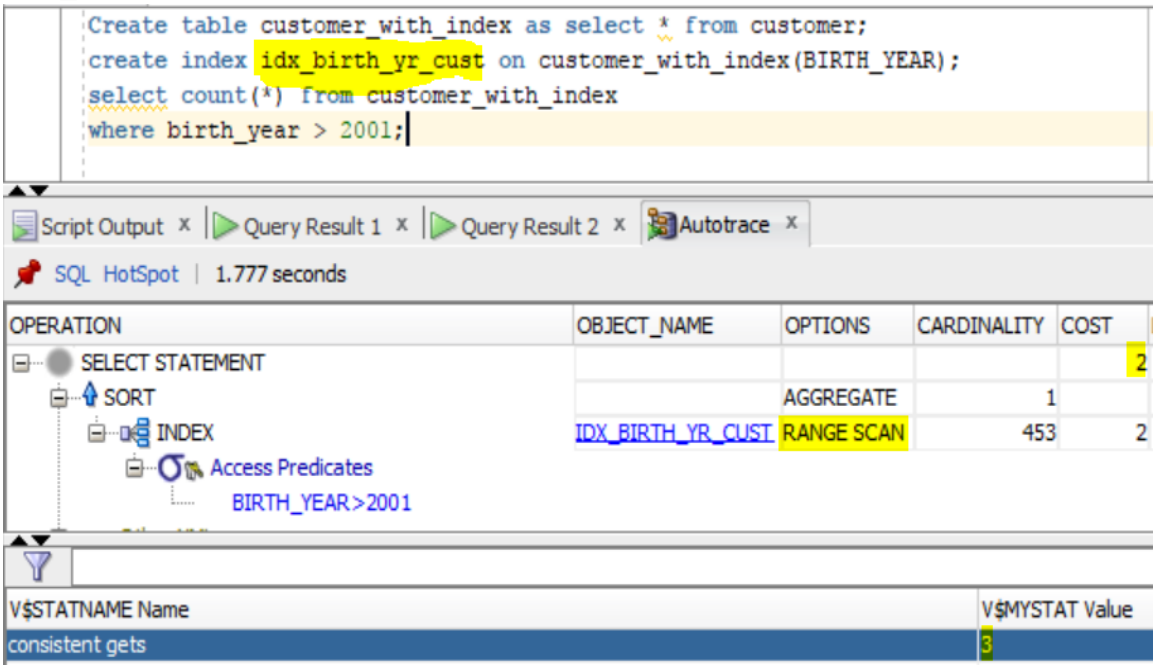
Purpose: The customer table has approximately 11K records and is queried on BIRTH_YEAR column to get the statistics on age category of the customers of Bulls Eat.

Steps:

1. Checked the execution plan, query cost and consistent gets of customer table having no index on birth_year.



2. Copied the customer table into customer_with_index table and created an index on birth_year column to compare the execution plan.



3. Next, we create a new customer_with_partition table with range partition on birth_year column.

```
create table customer_with_partition
(
"CUST_ID" NUMBER(10,0),
"FNAME" VARCHAR2(50 BYTE),
"LNAME" VARCHAR2(50 BYTE),
"EMAIL" VARCHAR2(100 BYTE),
"GENDER" VARCHAR2(10 BYTE),
"BIRTH_DAY" NUMBER,
"BIRTH_MONTH" NUMBER,
"BIRTH_YEAR" NUMBER
)
PARTITION BY RANGE(BIRTH_YEAR)
(PARTITION P1 VALUES LESS THAN(1950),
 PARTITION P2 VALUES LESS THAN(1975),
 PARTITION P3 VALUES LESS THAN(2000),
 PARTITION P4 VALUES LESS THAN(MAXVALUE)
);
INSERT INTO customer_with_partition
SELECT * FROM CUSTOMER;
COMMIT;
```

4. Comparing the execution plan.

```
select count(*) from customer_with_partition
where birth_year > 2001;
```

Script Output | Query Result 1 | Query Result 2 | Autotrace

SQL HotSpot | 1.815 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	PARTITION_START	COST
SELECT STATEMENT					274
SORT		AGGREGATE	1		
PARTITION RANGE		SINGLE	1	4	274
TABLE ACCESS	CUSTOMER_WITH_PARTITION	FULL	1	4	274
Filter Predicates					
BIRTH_YEAR>2001					

V$STATNAME Name	V$MYSTAT Value
consistent gets	53

Explanation:

Since the table is not very big i.e. having table size of more than 2GB, creating partitions on the customer table increases the cost and consistent gets, whereas, creating an index on the column BIRTH_YEAR gives positive results in performance as summarized in the table below:

Query: select count(*) from customer where birth_year > 2001;			
	Consistent Gets	Execution Time	Query Cost
Without Index or Partition	107	1.817 seconds	27
With Index	3	1.777 seconds	2
With Partition	53	1.815 seconds	274

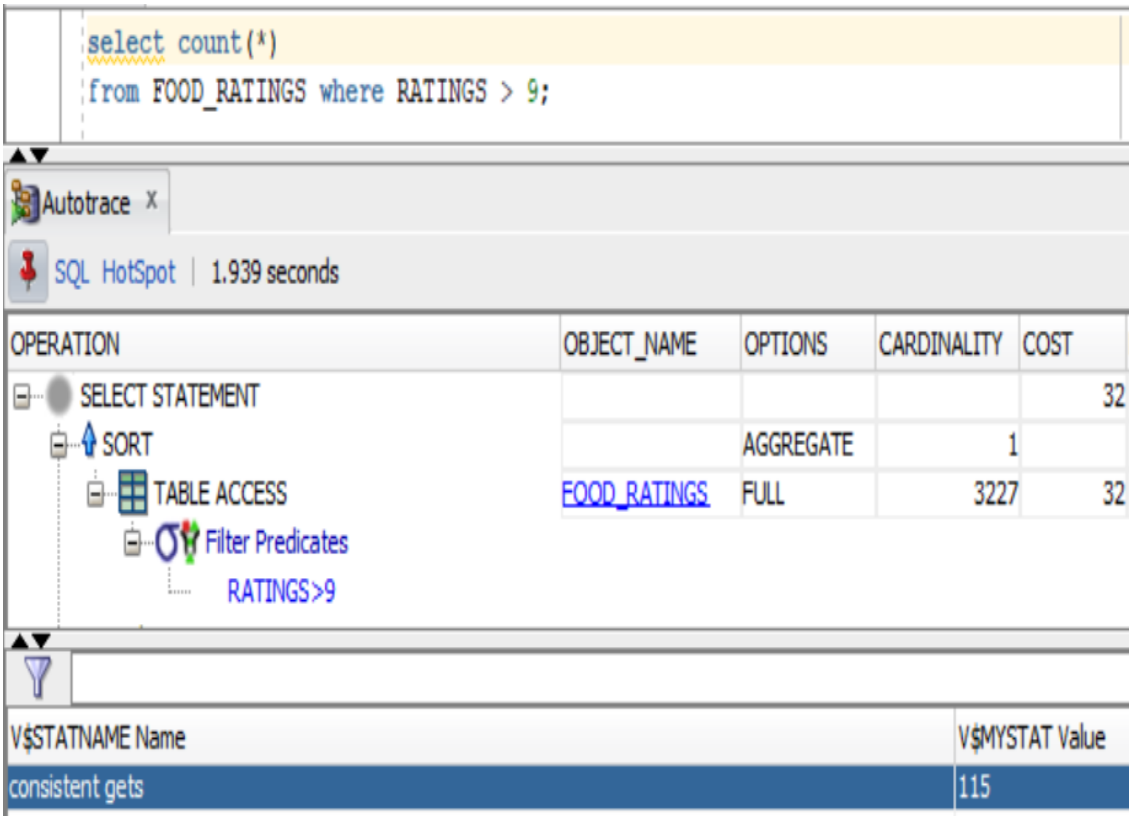
#5 Experiment with Parallelism:

Purpose:

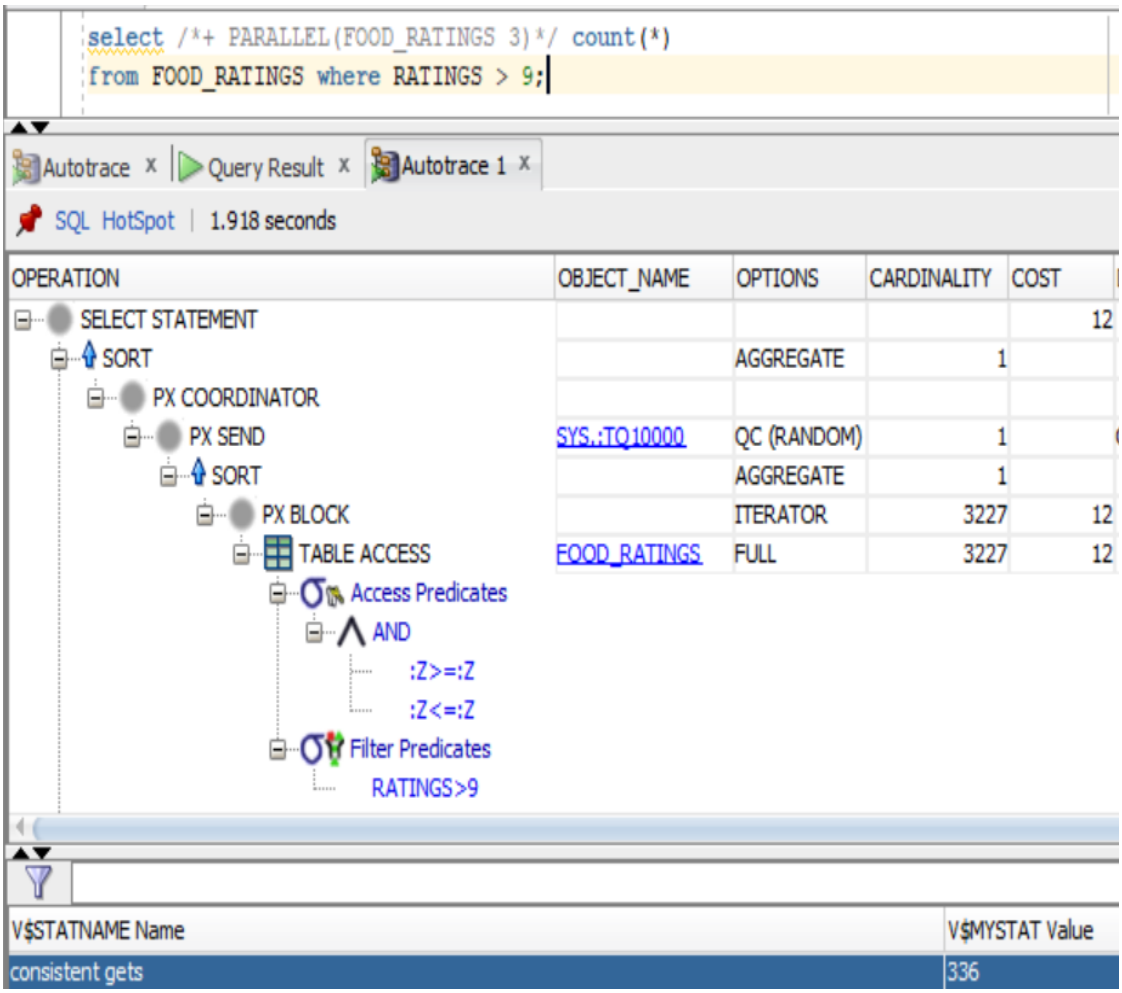
This experiment will be checking the impact of implementing parallelism on a big table i.e. food_ratings table which has 30K records.

Steps:

1. Executing the query without parallelism and checking the execution plan.



2. Executing the same query using query hints for parallelism and comparing the execution plan.



Explanation:

The query performs better with respect to execution time and the query cost after implementing parallelism, however, we see an increase in the consistent gets. Details are summarized below:

Query: select count(*) from food_ratings where ratings>9;			
	Consistent Gets	Execution Time	Query Cost
Without Parallelism	115	1.939 seconds	32
With Parallelism	336	1.918 seconds	12

Part 4 – Recommendation System

A recommender system helps a user discover products or content by predicting the user's preference of each item and then showing them the items they would rate highly. It gives the users personalized recommendations based of similarity of various parameters.

Recommender system uses various information filtering techniques. The two main types of information filtering techniques are:

1)Content Based Filtering - Content-based techniques match content resources to user characteristics. It ignores contributions from other users and the prediction is based on the active user's information.

2)Collaborative Filtering – Collaborative filtering recommends items by identifying other users with similar taste. It uses the opinion of other users to recommend items to the active user. The idea behind collaborative filtering is to recommend new items based on the similarity of users.

We have decided to use Collaborative Filtering in our project 'Bulls Eat' to get food recommendations for users based on the similarity in food ratings. We are using a similarity measure called Cosine-Similarity which calculates the similarity based on 'anti-distance'. The less is the distance between two vectors of user ratings, the more similar they are.

The basic formula for cosine similarity is as below:

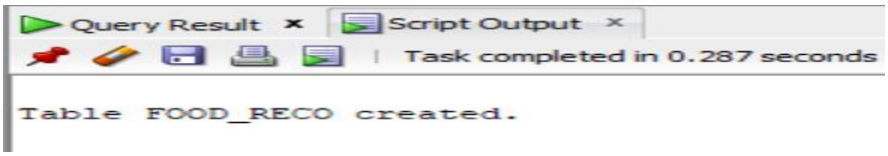
$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Bulls Eat provides food recommendation to its customer based on collaborative filtering recommendation using the food ratings table.

The table food_reco is created to store the similar customer's profile and the score calculated with reference.

Worksheet | Query Builder

```
CREATE TABLE food_reco (
    cust_id NUMBER(4,0) PRIMARY KEY,
    score NUMBER);
```



The collaborative filtering recommendation system has been implemented using two stored procedures as described below:

#1 Stored Procedure - food_recommender_dataset()

- This procedure clears and populate food_reco table for the customer ID being passed in the procedure.
- The data populated in this table comprises of a list of top 30 customers and their respective cosine similarity score to get a list of the customers having similar tastes like that of the customer for who recommendation must be given.
- Note: In collaborative recommendation system – two customers are similar based on their rating pattern.
- The ratings are normalized to achieve good recommendation.

```
create or replace procedure food_recommender_dataset(i_Customer IN NUMBER)
as
 BEGIN
    declare c INTEGER default 0;
    BEGIN
        select NUM_ROWS into c from user_tables where table_name='FOOD_RECO';
        IF c!=0 then
            execute immediate 'truncate table FOOD_RECO';
        END IF;
        insert into FOOD_RECO
        SELECT users.cust_id AS CUST_ID,distances.dist / (SQRT(my.norm) * SQRT(users.norm)) AS SCORE
         FROM (SELECT fr.cust_id, SUM((fr.ratings) * (cust.ratings)) as dist
               FROM (SELECT * FROM FOOD_RATINGS WHERE CUST_ID!= i_Customer) fr
                     INNER JOIN (Select * from FOOD_RATINGS where cust_ID= i_Customer) cust
                     ON fr.menu_id = cust.menu_id
                     GROUP BY fr.cust_id) distances
               INNER JOIN (SELECT cust_id, SUM((ratings) * (ratings)) AS norm
                           FROM food_ratings where cust_ID!=i_Customer
                           GROUP BY cust_ID) users
               ON distances.cust_id = users.cust_id
               CROSS JOIN
               (SELECT SUM((ratings) * (ratings)) AS norm
                FROM FOOD_RATINGS where cust_ID=i_Customer) my
          ORDER BY score DESC
          FETCH FIRST 30 ROWS ONLY;
        END;
 END;
/
```

#2 Stored Procedure - food_recommender()

- This procedure displays recommendation of food items for a customer whose CUST_ID is passed as a parameter.
- The procedure uses food_recommender_dataset() to populate food_reco table and then uses the same to join and fetch menu item names for the customer.
- Number of recommendation displayed are controlled based on REC_DISPLAY column in recommendation_dashboard table which stores user preferences of number of recommendations.

```
create or replace procedure food_recommender(i_Customer IN NUMBER)
as
 begin
  declare v_rows NUMBER;
  begin
       food_recommender_dataset(i_Customer);
       select REC_DISPLAY into v_rows from RECOMMENDATION_DASHBOARD where CUST_ID = i_Customer;
       dbms_output.put_line('Hi! You can try the following items from our menu: ');
       for cur in (select ITEM from
                     (SELECT
                        me.name ITEM, ROUND(SUM(ratings) / COUNT(*), 1) AS score
                      FROM
                        food_reco reco
                        INNER JOIN (SELECT * FROM FOOD_RATINGS WHERE CUST_ID!=i_Customer) fr
                            ON reco.cust_id = fr.cust_id
                        INNER JOIN menu me
                            ON fr.menu_id = me.menu_id
                      WHERE fr.menu_id NOT IN
                                  (SELECT menu_id from FOOD_RATINGS where cust_ID=i_Customer)
                      GROUP BY me.name
                      ORDER BY score DESC
                      FETCH FIRST v_rows ROWS ONLY) a)
       loop
       dbms_output.put_line(cur.ITEM);
       end loop;
   end;
end;
/
```

Results:

The recommendation_dashboard table is queried to get 3 customer IDs along with their recommendation preferences.

```
select * from RECOMMENDATION_DASHBOARD
fetch first 3 rows only;
```

Query Result | SQL | All Rows Fetched: 3 in 0.037 seconds

	CUST_ID	REC_DISPLAY
1	2439	8
2	2440	7
3	2441	1

First, the food_recommender() procedure is executed with cust_id = 2439

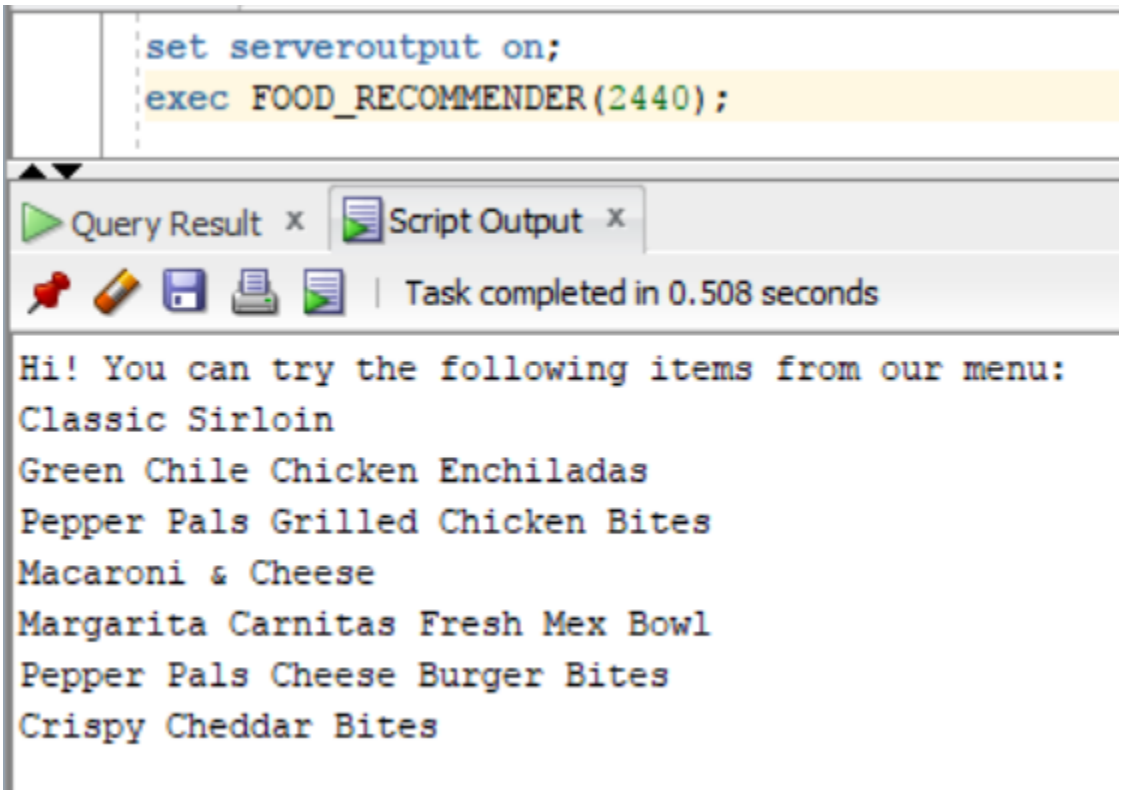
```
set serveroutput on;
exec FOOD_RECOMMENDER(2439);
```

Query Result | Script Output | Task completed in 0.678 seconds

```
Hi! You can try the following items from our menu:
Pepper Pals Grilled Chicken Bites
Chipotle Shrimp Fresh Mex Bowl
Side - Black Beans
Skillet Chocolate Chip Cookie
Smothered Carnitas Burrito
Guacamole Burger
Char-Crusted Sirloin
Pepper Pals Crispy Chicken Crispers
```

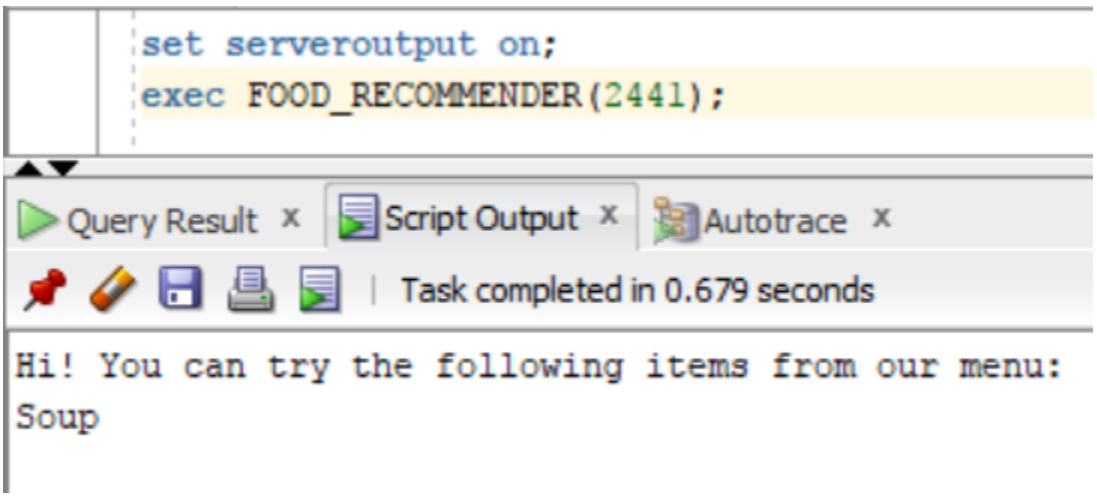
Here, as the customer had 8 recommendations as preference, only 8 recommendations are displayed.

Second, the procedure is executed for CUST_ID = 2440



Here, as the customer had 7 recommendations as preference, only 8 recommendations are displayed.

Likewise, there is only one recommendation displayed for customer ID = 2441



Appendix

Below described are the DDLs of all the tables that have been populated, the corresponding constraints as well as the sequences and trigger definition used for data loading.

Customer Table:

```
CREATE TABLE "CUSTOMER"
 (
"CUST_ID" NUMBER(10,0),
"FNAME" VARCHAR2(50 BYTE),
"LNAME" VARCHAR2(50 BYTE),
"EMAIL" VARCHAR2(100 BYTE),
"GENDER" VARCHAR2(10 BYTE),
"BIRTH_DAY" NUMBER,
"BIRTH_MONTH" NUMBER,
"BIRTH_YEAR" NUMBER
 ) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Primary Key (With Auto-Unique Index) Creation:

```
ALTER TABLE "CUSTOMER" ADD CONSTRAINT "PK_CUSTOMER" PRIMARY KEY ("CUST_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

Customer_Address Table:

```
CREATE TABLE "DB233"."CUSTOMER_ADDRESS"
(
"CUST_ID" NUMBER(*,0),
"ADDR_TYPE" VARCHAR2(50 BYTE),
"STREET_ADDR" VARCHAR2(100 BYTE),
"CITY" VARCHAR2(50 BYTE),
"STATE" VARCHAR2(50 BYTE),
"ZIP" NUMBER(*,0)
) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Primary Key (With Auto-Unique Index) Creation:

```
ALTER TABLE "DB233"."CUSTOMER_ADDRESS" ADD CONSTRAINT "PK_CUSTOMER_ADDRESS"
PRIMARY KEY ("CUST_ID", "ADDR_TYPE")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
```

```
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

Foreign Key Consraint:

```
alter table "ORDERS" add constraint "FK_ORDERS" foreign key ("CUST_ID") references
"CUSTOMER"("CUST_ID");
```

ORDERS table:

```
CREATE TABLE "DB233"."ORDERS"
(
"ORDER_ID" NUMBER(*,0),
"CUST_ID" NUMBER(*,0),
"ORDER_DATE" DATE,
"TOTAL_AMT" FLOAT(126),
"PAYMENT_MODE" VARCHAR2(50 BYTE),
"PAYMENT_DATE" DATE
) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Primary Key (With Auto-Unique Index) Creation:

```
ALTER TABLE "DB233"."ORDERS" ADD CONSTRAINT "PK_ORDERS" PRIMARY KEY ("ORDER_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

Foreign Key Consraint:

```
alter table "ORDERS" add constraint "FK_ORDERS" foreign key ("CUST_ID") references
"CUSTOMER"("CUST_ID");
```

ORDER_DETAILS table:

```
CREATE TABLE "DB233"."ORDER_DETAILS"
(
"ORDER_ID" NUMBER(*,0),
"MENU_ID" NUMBER(*,0),
"QUANTITY" NUMBER(*,0)
) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Primary Key (With Auto-Unique Index) Creation:

```
ALTER TABLE "DB233"."ORDER_DETAILS" ADD CONSTRAINT "PK_ORDER_DETAILS" PRIMARY
KEY ("ORDER_ID", "MENU_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

Foreign Key Constraints:

```
alter table "ORDER_DETAILS" add constraint "FK1_ORDER_DETAILS" foreign key ("ORDER_ID")
references "ORDERS"("ORDER_ID");
alter table "ORDER_DETAILS" add constraint "FK2_ORDER_DETAILS" foreign key
("MENU_ID") references "MENU"("MENU_ID");
```

MENU table:

```
CREATE TABLE "DB233"."MENU"
(
"MENU_ID" NUMBER(*,0),
"NAME" VARCHAR2(250 BYTE),
"CAT_ID" NUMBER(*,0),
"DESCRIPTION" VARCHAR2(250 BYTE),
"PRICE_USD" FLOAT(126),
"OVERALL_RATING" FLOAT(126),
"FLAVOR" VARCHAR2(50 BYTE)
) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Primary Key (With Auto-Unique Index) Creation:

```
ALTER TABLE "DB233"."MENU" ADD CONSTRAINT "PK_MENU" PRIMARY KEY ("MENU_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

Foreign Key Constraints:

```
alter table "MENU" add constraint "FK_MENU" foreign key ("CAT_ID") references
"MEAL_CATEGORY"("CAT_ID");
```

MEAL_CATEGORY table:

```
CREATE TABLE "DB233"."MEAL_CATEGORY"
(
"CAT_ID" NUMBER(*,0),
"CAT_NAME" VARCHAR2(200 BYTE)
) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
```

```
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Primary Key (With Auto-Unique Index) Creation:

```
ALTER TABLE "DB233"."MEAL_CATEGORY" ADD CONSTRAINT "PK_MEAL_CATEGORY" PRIMARY
KEY ("CAT_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

RECIPE table:

```
CREATE TABLE "DB233"."RECIPE"
(
"MENU_ID" NUMBER(*,0),
"INGRED_ID" NUMBER(*,0),
"QTY_USED" FLOAT(126)
) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Primary Key (With Auto-Unique Index) Creation:

```
ALTER TABLE "DB233"."RECIPE" ADD CONSTRAINT "PK_RECIPE" PRIMARY KEY ("MENU_ID",
"INGRED_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

Foreign Key Constraints:

```
alter table "RECIPE" add constraint "FK1_RECIPE" foreign key ("MENU_ID") references
"MENU"("MENU_ID");
alter table "RECIPE" add constraint "FK2_RECIPE" foreign key ("INGRED_ID") references
"INGREDIENT_INVENTORY"("INGREDIENT_ID");
```

INGREDIENT_INVENTORY table:

```
CREATE TABLE "DB233"."INGREDIENT_INVENTORY"
(
"INGREDIENT_ID" NUMBER(*,0),
"INGRED_NAME" VARCHAR2(100 BYTE),
"CURR_QTY" FLOAT(126),
"REFILL_THRESHOLD" FLOAT(126),
"MEASUREMENT_UNIT" VARCHAR2(20 BYTE)
) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
```

```
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Primary Key (With Auto-Unique Index) Creation:

```
ALTER TABLE "DB233"."INGREDIENT_INVENTORY" ADD CONSTRAINT
"PK_INGREDIENT_INVENTORY" PRIMARY KEY ("INGREDIENT_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

FOOD_RATINGS table:

```
CREATE TABLE "DB233"."FOOD_RATINGS"
(
"CUST_ID" NUMBER(*,0),
"MENU_ID" NUMBER(*,0),
"RATINGS" FLOAT(126),
"RATED_DT" DATE
) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Primary Key (With Auto-Unique Index) Creation:

```
ALTER TABLE "DB233"."FOOD_RATINGS" ADD CONSTRAINT "PK_FOOD_RATINGS" PRIMARY KEY
("CUST_ID", "MENU_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

Foreign Key Constraints:

```
alter table "FOOD_RATINGS" add constraint "FK1_FOOD_RATINGS" foreign key ("CUST_ID")
references "CUSTOMER"("CUST_ID");
alter table "FOOD_RATINGS" add constraint "FK2_FOOD_RATINGS" foreign key ("MENU_ID")
references "MENU"("MENU_ID");
```

FAVORITES table

```
CREATE TABLE "DB233"."FAVORITES"
(
"CUST_ID" NUMBER(*,0),
"FAV_1" NUMBER(*,0),
"FAV_2" NUMBER(*,0),
"FAV_3" NUMBER(*,0)
) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
```

```
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Primary Key (With Auto-Unique Index) Creation:

```
ALTER TABLE "DB233"."FAVORITES" ADD CONSTRAINT "PK_FAVORITES" PRIMARY KEY
("CUST_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

Foreign Key Constraints:

```
alter table "FAVORITES" add constraint "FK1_FAVORITES" foreign key ("CUST_ID") references
"CUSTOMER"("CUST_ID");
alter table "FAVORITES" add constraint "FK2_FAVORITES" foreign key ("FAV_1") references
"MENU"("MENU_ID");
alter table "FAVORITES" add constraint "FK3_FAVORITES" foreign key ("FAV_2") references
"MENU"("MENU_ID");
alter table "FAVORITES" add constraint "FK4_FAVORITES" foreign key ("FAV_3") references
"MENU"("MENU_ID");
```

Sequences (Auto-number):

Create Sequence ORDERS_SEQUENCE

```
CREATE SEQUENCE ORDERS_SEQUENCE
  MINVALUE 1
  MAXVALUE 9999999
  START WITH 1
  INCREMENT BY 1
  CACHE 20;
```

Create Sequence MENU_SEQUENCE

```
CREATE SEQUENCE MENU_SEQUENCE
  MINVALUE 1
  MAXVALUE 99999
  START WITH 1
  INCREMENT BY 1
  CACHE 20;
```

Create Sequence CATEGORY_SEQUENCE

```
CREATE SEQUENCE CATEGORY_SEQUENCE
  MINVALUE 1
  MAXVALUE 99999
  START WITH 1
  INCREMENT BY 1
  CACHE 20;
```

Create Sequence INGREDIENTS_SEQUENCE

```
CREATE SEQUENCE INGREDIENTS_SEQUENCE
  MINVALUE 1
  MAXVALUE 99999
  START WITH 1
  INCREMENT BY 1
  CACHE 20;
```

Triggers:

TR_POPULATE_ORDER_ID

This trigger increments the ORDER_ID before an insert into ORDERS table.

```
CREATE OR REPLACE EDITIONABLE TRIGGER "TR_POPULATE_ORDER_ID"
BEFORE INSERT ON ORDERS
FOR EACH ROW
BEGIN
  SELECT ORDERS_SEQUENCE.NEXTVAL
  INTO   :new.order_id
  FROM   dual;
END;
/
ALTER TRIGGER "TR_POPULATE_ORDER_ID" ENABLE;
```

TR_POPULATE_MENU_MENU_ID

This trigger increments the MENU_ID before an insert into the MENU table.

```
CREATE OR REPLACE EDITIONABLE TRIGGER "TR_POPULATE_MENU_MENU_ID"
BEFORE INSERT ON MENU
FOR EACH ROW
BEGIN
SELECT MENU_SEQUENCE.NEXTVAL
INTO   :new.MENU_ID
FROM   dual;
END;
/
ALTER TRIGGER "TR_POPULATE_MENU_MENU_ID" ENABLE;
```

TR_POPULATE_MEAL_CAT_ID

This trigger increments the CAT_ID before an insert into the MEAL_CATEGORY table.

```
CREATE OR REPLACE EDITIONABLE TRIGGER "TR_POPULATE_MEAL_CAT_ID"
BEFORE INSERT ON MEAL_CATEGORY
FOR EACH ROW
BEGIN
  SELECT CATEGORY_SEQUENCE.NEXTVAL
  INTO   :new.cat_id
  FROM   dual;
END;
/
ALTER TRIGGER "TR_POPULATE_MEAL_CAT_ID" ENABLE;
```

TR_POPULATE_INGREDIENT_ID

This trigger increments the INGREDIENT_ID before an insert into the INGREDIENT_INVENTORY table.

```
CREATE OR REPLACE EDITIONABLE TRIGGER "TR_POPULATE_INGREDIENT_ID"
BEFORE INSERT ON INGREDIENT_INVENTORY
FOR EACH ROW
BEGIN
  SELECT INGREDIENTS_SEQUENCE.NEXTVAL
  INTO   :new.INGREDIENT_ID
  FROM   dual;
END;
/
ALTER TRIGGER "TR_POPULATE_INGREDIENT_ID" ENABLE;
```