DATAMAZE - Grubhub Simulation

Enterprise Database Management - Fall 2019

Pawan Ailani

Jordan Baker

Surbhi Chaturvedi

Harrison Heim

Tazeen Munnavar

Tamim Sangrar

# **Table of Contents**

# [**Chapter 1: Requirements Gathering**](#_Requirements_Gathering)**……………………………………………………….3**

# [**Chapter 2: ER Diagram and Data Dictionary**](#_ER_Diagram_and) **……………………………………………..7**

[**Dictionary**](#_DATA_DICTIONARY)**………………………………………………………………………………………..8**

# [**Chapter 3: Normalized Schema**](#_Normalized_Schema)**……………………………………………………………16**

# [**Chapter 4: Complex Queries**](#_Complex_Queries)**……………………………………………………………….21**

[**Complex Query 1**](#_Query_1:)**……………………………………………………………………………...21**

[**Complex Query 2**](#_Query_2:)**………………………………………………………………………..…….23**

[**Complex Query 3**](#_Query_3:)**……………………………………………………………………………...25**

[**Complex Query 4**](#_Query_4:)**……………………………………………………………………………...27**

[**Complex Query 5**](#_Query_5:)**……………………………………………………………………………...28**

[**Complex Query 6**](#_Query_6:)**…………………………………………………………………………..….29**

[**Complex Query 7**](#_Query_7:)**……………………………………………………………………………...31**

[**Complex Query 8**](#_Query_8:)**……………………………………………………………………………...33**

[**Complex Query 9**](#_Query_9:)**……………………………………………………………………………...35**

[**Complex Query 10**](#_Query_10:)**…………………………………………………………………………….36**

[**Complex Query 11**](#_Query_11:)**…………………………………………………………………………….38**

[**Complex Query 12**](#_Query_12:)**…………………………………………………………………………….40**

# [**Chapter 5: Complex Triggers and Procedures**](#_Complex_Triggers_and)**………………………………………….41**

[**Payments**](#_Payments_Trigger)**…………………………………………………………………….…………………41**

[**Customer Favorites**](#_Customer_Favorites_Procedure)**…………………………………………………………………………..45**

[**Password**](#_Password_Hash_Trigger)**……………………………………………………………………………………….47**

[**Password Check**](#_Password_Check_Procedure)**…………………………………………………..…………………………..48**

# [**Chapter 6: User Interface**](#_User_Interface)**…………………………………………………………………...49**

[**Demo**](#_Demo)**…………………………………………………………………………………………....49**

# [**Chapter 7: Conclusion**](#_Conclusion)**………………………………………………………………………50**

# **[Appendix](#_Supplemental_Information)……………………………………………………………………………………….52**

[**SQL Creation**](#_SQL_Statements_to)**…………………………………………………………………………………..52**

[**Citations**](#_Works_Citied_1)**………………………………………………………………………………………...71**

# **[Requirements Gathering](#_Table_of_Contents)**

Our client is Grubhub, a food delivery aggregator that partners with restaurants to offer fast delivery service. The client has amassed many customers. They access the Grubhub app to place orders. For each customer, Grubhub records their name (first and last), phone number, address (street, city, state, and zip), and payment method. A customer may have more than one payment method saved, and only one phone number. Within the app, customers may choose to favorite certain restaurants that they frequent, and Grubhub would like to track that information as well.

Customers place orders from a wide variety of restaurant chains. Each chain may have many different restaurant locations. In our case, each restaurant is an instance of a chain. For each chain, Grubhub tracks the name, unique ID, and website. When a customer places an order, it is sent from the customer to the restaurant account. Each restaurant has one account. Grubhub keeps a record of restaurant account information such as bank account number, username, password, and unique account ID. Customers may place one order at any given time.

Restaurants are divided into three categories: Delivery, Pickup, and Catering. Some restaurants offer all three, while some only offer one. For each restaurant, Grubhub needs to record its category, address (street, city, state, zip), phone number, and hours of operation. For the delivery restaurants, they capture the delivery cost and minimum order amount. For catering restaurants, they track the required down payment, headcount, allowed lead time, minimum and maximum party sizes, and the price per head.

Each restaurant serves one or more cuisines. Information is collected about each cuisine. A restaurant must have at least one cuisine, and each cuisine must be associated with one restaurant. The regions associated with each type of cuisine are also tracked. For each chain, Grubhub stores the items it offers. Each item contains an item name, an item description, categories, unique ID, dietary restrictions, and the price of the item. Restaurants may only offer items that its parent chain offers. They do not have to offer the entire list of items.

Grubhub stores a lot of information once an order is placed. That includes the date of the order, the location of the restaurant, and any special instructions the order may entail. The total cost, items included, payment details, order status, and any promotion codes used are also tracked. Each order is assigned a unique ID.

Food is delivered to customers via delivery drivers. For each driver, their name (first and last), email address, social security number, phone number, and date of birth are recorded. Their address (street, city, state, zip), as well as the maximum range that they would be willing to deliver to (in miles), is also tracked. Grubhub records the driver’s license number, their vehicle insurance provider number, and the driver’s bank account number as well. Once a driver begins their shift, their delivery activity is tracked along with their activity status (whether the driver is online or offline), clock in time and date, and clock out time and date. Additionally, the number of deliveries made, and the number of miles covered also need to be recorded.

For each order, payment is required. Grubhub needs to record information for each payment. These details include the profit earned and the amounts that go to the customer, driver, and restaurant.

Once the customer has received their order, the app allows them to fill out a rating form. The customer can rate both the food and the driver for any particular order. The customer can choose to leave one rating or no rating at all. Only one rating is allowed per order. Each individual rating will be left by one customer at most, and they can only do so upon placing an order. The customer can rate a maximum of five stars and a minimum of one star. Customers are also allowed to leave their own comments. Each rating has a unique rating ID.

Apart from contracted drivers, Grubhub has almost 3000 employees. These employees are split across multiple departments, including, but not limited to, sales, customer service, marketing, finance, software development, and human resources. If you are an employee, you must belong to one department, and conversely, each department will have at least one employee. For all employees, a unique employee ID number is assigned. Their names, date of birth, phone number, email, and start dates (to determine the length of tenure) are also stored. Annual sales target information is tracked for sales employees to determine their yearly bonuses. For customer service employees, a record of their customer review rating is maintained. Employees from the finance department may manage multiple restaurant accounts, but it is not necessary that they manage any. Each restaurant account must be managed by one finance employee.

The marketing department is responsible for developing perks. Perks are then applied to orders in the form of promotion codes. Each perk must be developed by one marketing employee. Only one code may be applied per order. Perks are associated with individual restaurants. Restaurants may offer many perks, but they are not required to offer any. If the perks are new, it is possible that they may have not yet been applied to any orders. For each perk, Grubhub captures the perk name, promotion code, promotion amount, start date, and end date. Each perk is assigned a unique ID.

Employees from the human resources department are responsible for listing job postings to fill open positions. An HR employee may list many job postings, but they are not all required to do so. Every job posting must be posted by an HR employee. There are times when no job postings exist for a particular job. Every posting is associated with only one job. Each job posting contains the department associated, the status of the job position (open or closed), date when the job was posted, and the job description.

Potential employees (referred to as applicants) apply for job postings. Each job posting is an instance of a job. It is possible for an applicant to submit multiple applications. The applicant’s first name, last name, date of birth, age, and the highest degree of qualification is captured. The application then goes back to HR for review. Not all HR employees are involved in the hiring process, so it is possible that a single HR employee may never review an application. It is also possible for a single HR employee to review many applications.

Event teams organize events to generate leads for new restaurant accounts. These teams are made up of one human resources employee, one marketing employee, and at least three salespeople. For each event, the event location, number of attendees, and its unique ID is captured. The same event team may organize multiple events, but the teams are not typically the same. Each event is created by only one team.

The leads generated from these events need to be tracked so they can be pursued. Events typically generate multiple leads but sometimes generate none. Leads are only generated from events. For each lead, the name of the prospective restaurant, the name, and title of the person to be contacted and their contact information (email address and phone) are recorded. A salesperson may pursue multiple leads, but they are not all required to do so. Each individual lead will only be pursued by one salesperson.

The marketing department runs campaigns to promote these events. Each campaign will promote at least one event but may promote many. Not every event is promoted by an external campaign. They also host internal campaigns to boost sales numbers. For each campaign, its name, the information about the employees involved, and event(s) that are being promoted are recorded every event a unique ID is assigned. For internal campaigns, the office location and room number are recorded. External campaigns have their type and medium captured.

Occasionally when placing orders, customers experience problems. They can report these problems to Grubhub. Since multiple customers may report the same problem, the problem type is recorded. One customer may report many different problems, but it is possible that they report none. The status, date created, date resolved, and priority are captured for each problem type. Each problem ticket must be reported by a customer, which includes problem description, category, and resolution. For each problem ticket, there may be multiple instances of each problem type. Each problem instance is resolved by a member of the customer service team.

Grubhub has partnered with various corporations like Yelp to boost deliveries made by the app. Salespeople target corporate partners to generate these partnerships. They can target multiple partners at one time. Grubhub stores the name of the partner, the services provided by the partner, and also assigns the corporate partner their unique id. Each corporate partner must have a partner contract, and for each partner contract, Grubhub records the contract ID, the contract status, the start date of the contract, and the date of contract expiration.

# **[ER Diagram and Data Dictionary](#_Table_of_Contents)**

Once we established our requirements, we began to create an ER Diagram for our mini world. In total, we ended up with 27 strong entities, 13 subclasses and a weak entity, and many attributes. We model the order process from start to finish and all of the related entities. Furthermore, we modeled how the information would flow through the corporate body of Grubhub, such as employees and their departments, marketing campaigns to attract new partners, and a job portal. Please double-click the ER below to view the Visio diagram, as well as the Data Dictionary, to be taken to excel.



|  |  |  |
| --- | --- | --- |
| **[DATA DICTIONARY](#_Table_of_Contents)** | | |
| ***Schema Construct*** | ***Data Type*** | ***Constraint*** |
| **APPLICANTS** | *Relation representing the entity class APPLICANTS* | |
| APPLICANTID | VARCHAR2(10 BYTE) | Primary Key |
| HIGHESTDEGREE | VARCHAR2(50 BYTE) |  |
| FIRSTNAME | VARCHAR2(50 BYTE) |  |
| LASTNAME | VARCHAR2(50 BYTE) |  |
| DOB | DATE |  |
| AGE | NUMBER (3,0) |  |
| **APPLICATIONS** | *Relation representing the entity class APPLICATIONS* | |
| APPLICATIONID | VARCHAR2(10 BYTE) | Primary Key |
| APP\_RESUME | BLOB |  |
| APPLICANTID | VARCHAR2(50 BYTE) | Foreign Key, references PRODUCTS |
| **APPLY\_TO** | *Relation representing the entity class APPLY\_TO* | |
| APPLICATIONID | VARCHAR2(10 BYTE) | Foreign Key, references APPLICATIONS |
| JOBPOSTINGID | VARCHAR2(10 BYTE) | Foreign Key, references JOB\_POSTINGS |
| **CAMPAIGNS** | *Relation representing the entity class CAMPAIGNS* | |
| CAMPAIGNID | VARCHAR2(10 BYTE) | Primary Key |
| CAMPAIGNNAME | VARCHAR2(50 BYTE) |  |
| **CATERING\_RESTAURANT** | *Relation representing the entity class CATERING\_RESTAURANTS, subclass of RESTAURANTS* | |
| RESTID | VARCHAR2(10 BYTE) | Foreign Key, references RESTAURANTS |
| PRICEPERHEAD | NUMBER (5,0) | Not NULL |
| MAXPARTYSIZE | NUMBER (5,0) |  |
| MINPARTYSIZE | NUMBER (5,0) |  |
| REQUIREDLEADTIME | NUMBER (5,0) |  |
| DOWNPAYMENTAMT | NUMBER (5,0) |  |
| **CHAINS** | Relation representing the entity class CHAINS | |
| CHAINID | VARCHAR2(10 BYTE) | Primary Key |
| WEBSITE | VARCHAR2(50 BYTE) |  |
| CHAINNAME | VARCHAR2(50 BYTE) | Not NULL |
| **COMPRISE\_HUMAN\_RESOURCES** | *Relation representing the entity class COMPRISE\_HUMAN\_RESOURCES* | |
| EVENTTEAMID | VARCHAR2(10 BYTE) | Foreign Key, references EVENT\_TEAMS |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references HUMAN\_RESOURCES\_EMPLOYEES |
| **COMPRISE\_MARKETING** | *Relation representing the entity class COMPRISE\_MARKETING* | |
| EVENTTEAMID | VARCHAR2(10 BYTE) | Foreign Key, references EVENT\_TEAMS |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references MARKETING\_EMPLOYEES |
| **COMPRISE\_SALES** | *Relation representing the entity class COMPRISE\_SALES* | |
| EVENTTEAMID | VARCHAR2(10 BYTE) | Foreign Key, references EVENT\_TEAMS |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references SALES\_EMPLOYEES |
| **CORPORATE\_PARTNERS** | *Relation representing the entity class CORPORATE\_PARTNERS* | |
| CORPPARTID | VARCHAR2(10 BYTE) | Primary Key |
| CORPPARTNAME | VARCHAR2(50 BYTE) | Not NULL |
| SERVICEPROVIDED | VARCHAR2(50 BYTE) | Not NULL |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| **CREATES** | *Relation representing the entity class CREATES* | |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| CAMPAIGNID | VARCHAR2(10 BYTE) | Foreign Key, references CAMPAIGNS |
| **CUISINES** | *Relation representing the entity class CUISINES* | |
| CUISINEID | VARCHAR2(10 BYTE) | Primary Key |
| CUISINETYPE | VARCHAR2(50 BYTE) |  |
| REGION | VARCHAR2(50 BYTE) |  |
| **CUSTOMER\_CATEGORIES** | *Relation representing the entity class CUSTOMER\_CATEGORIES* | |
| CUST\_CATEGORY | CHAR (5 BYTE) | Primary Key |
| CATEGORY\_NAME | VARCHAR2(20 BYTE) |  |
| POINTS\_REQUIRED | NUMBER (4,0) |  |
| SERVICE\_DISCOUNT | NUMBER (4,3) | CHECK (Service\_Discount >= 0 and service\_discount < 1.0) |
| **CUSTOMER\_SERVICE\_EMPLOYEES** | *Relation representing the entity class CUSTOMER\_SERVICE\_EMPLOYEES, subclass of EMPLOYEES* | |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| CUSTREVIEWRATING | VARCHAR2(5 BYTE) |  |
| **CUSTOMERS** | *Relation representing the entity class CUSTOMERS* | |
| CUSTID | VARCHAR2(10 BYTE) | Primary Key |
| FIRSTNAME | VARCHAR2(50 BYTE) | Not NULL |
| LASTNAME | VARCHAR2(50 BYTE) | Not NULL |
| STREET | VARCHAR2(50 BYTE) | Not NULL |
| CITY | VARCHAR2(50 BYTE) | Not NULL |
| ZIP | VARCHAR2(5 BYTE) | Not NULL |
| CUSTSTATE | VARCHAR2(2 BYTE) | Not NULL |
| PHONENUM | NUMBER (10,0) | Not NULL, CHECK (phoneNum not like '%[^0-9]%'),CHECK (LENGTH(phoneNum) = 10) |
| CUSTEMAIL | VARCHAR2(100 BYTE) |  |
| **CUSTOMER\_FAVORITES** | *Relation representing the entity class CUSTOMERS\_FAVORITES* | |
| CUSTID | VARCHAR2(10 BYTE) | Foreign Key, references CUSTOMERS |
| FAVORITES | VARCHAR2(50 BYTE) |  |
| **CUSTOMERS\_PAYMETHOD** | *Relation representing the entity class CUSTOMERS\_PAYMETHOD* | |
| CUSTID | VARCHAR2(10 BYTE) | Foreign Key, references CUSTOMERS |
| PAYMETHOD | VARCHAR2(50 BYTE) |  |
| **DELIVERY\_RESTAURANTS** | *Relation representing the entity class DELIVERY\_RESTAURANTS* | |
| RESTID | VARCHAR2(10 BYTE) | Foreign Key, references RESTAURANTS |
| DELCOST | NUMBER (10,2) |  |
| MINORDERPRICE | NUMBER (10,0) |  |
| **DEVELOP** | *Relation representing the entity class DEVELOP, employees and perks relation* | |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| PERKID | VARCHAR2(10 BYTE) | Foreign Key, references PERKS |
| **DRIVER\_DELIVERIES** | *Relation representing the entity class DRIVER\_DELIVERIES* | |
| DELIVERYID | VARCHAR2(50 BYTE) | Primary Key |
| CLOCKINDATE | DATE |  |
| CLOCKINTIME | VARCHAR2(10 BYTE) |  |
| CLOCKOUTDATE | DATE |  |
| CLOCKOUTTIME | VARCHAR2(10 BYTE) |  |
| MILES | NUMBER (10,0) |  |
| NUMDELIVERY | NUMBER (10,0) |  |
| DRIVERID | VARCHAR2(10 BYTE) | Foreign Key, references DRIVERS |
| **DRIVER\_RATINGS** | *Relation representing the entity class DRIVER\_RATINGS* | |
| RATINGID | VARCHAR2(10 BYTE) | Primary Key |
| NUMSTARS | NUMBER (1,0) |  |
| **DRIVERS** | *Relation representing the entity class DRIVERS* | |
| DRIVERID | VARCHAR2(10 BYTE) | Primary Key |
| CITY | VARCHAR2(50 BYTE) |  |
| DRIVERSTATE | VARCHAR2(2 BYTE) |  |
| STREET | VARCHAR2(50 BYTE) |  |
| ZIP | VARCHAR2(5 BYTE) | Not NULL |
| BANKACCNUM | NUMBER (10,0) | Not NULL |
| DOB | DATE | Not NULL |
| DRIVERREWARDS | VARCHAR2(50 BYTE) |  |
| EMAIL | VARCHAR2(50 BYTE) | CHECK (email like '%@%.%') |
| FIRSTNAME | VARCHAR2(50 BYTE) | Not NULL |
| LASTNAME | VARCHAR2(50 BYTE) | Not NULL |
| SSN | VARCHAR2(11 BYTE) | Not NULL, UNIQUE |
| MAXRANGE | NUMBER (35,0) |  |
| PHONENUM | VARCHAR2(12 BYTE) | Not NULL, CHECK (phoneNum not like '%[^0-9]%'),CHECK (LENGTH(phoneNum) = 10), CHECK (LENGTH(phoneNum) = 10) |
| INSURANCEPNUM | VARCHAR2(20 BYTE) | Not NULL |
| LICNUM | VARCHAR2(10 BYTE) | Not NULL |
| **EMPLOYEES** | *Relation representing the entity class EMPLOYEES* | |
| EMPID | VARCHAR2(10 BYTE) | Primary Key |
| EMPFIRSTNAME | VARCHAR2(50 BYTE) | Not NULL |
| PHONENUM | NUMBER (10,0) | Not NULL, Should be 10 digits, CHECK (phoneNum not like '%[^0-9]%') |
| SSN | VARCHAR2(11 BYTE) | Not NULL, UNIQUE |
| STREET | VARCHAR2(50 BYTE) |  |
| CITY | VARCHAR2(50 BYTE) |  |
| EMPSTATE | VARCHAR2(50 BYTE) |  |
| ZIP | VARCHAR2(5 BYTE) | Not NULL |
| EMAIL | VARCHAR2(50 BYTE) | Not NULL, CHECK (email LIKE '%@%.%'), CHECK (LENGTH(phoneNum)=10) |
| JOINDATE | DATE | Not NULL |
| DOB | DATE | Not NULL |
| AGE | NUMBER (3,0) | CHECK (age >= 18) |
| EMPCOMP | NUMBER (10,0) | Not NULL |
| WAGETYPE | VARCHAR2(10 BYTE) |  |
| TENURE | VARCHAR2(50 BYTE) |  |
| **EVENT\_TEAMS** | *Relation representing the entity class EVENT\_TEAMS* | |
| EVENTTEAMID | VARCHAR2(10 BYTE) | Primary Key |
| EVENTTEAMNAME | VARCHAR2(50 BYTE) | Not NULL |
| CREATIONDATE | DATE |  |
| **EVENTS** | *Relation representing the entity class EVENTS* | |
| EVENTID | VARCHAR2(10 BYTE) | Primary Key |
| STREET | VARCHAR2(50 BYTE) |  |
| STATE | VARCHAR2(2 BYTE) |  |
| CITY | VARCHAR2(50 BYTE) |  |
| ZIP | VARCHAR2(5 BYTE) | Not NULL |
| NUMATTEND | NUMBER (10,0) |  |
| EVENTTEAMID | VARCHAR2(10 BYTE) | Foreign Key, references EVENT\_TEAMS |
| **EXTERNAL\_CAMPAIGN** | *Relation representing the entity class EXTERNAL\_CAMPAIGNS, subclass of CAMPAIGN* | |
| CAMPAIGNID | VARCHAR2(10 BYTE) | Foreign Key, references CAMPAIGNS |
| EC\_TYPE | VARCHAR2(50 BYTE) |  |
| EC\_MEDIUM | VARCHAR2(50 BYTE) |  |
| **FILLED\_JOBS** | Relation representing the entity class FILLED\_JOBS |  |
| JOBID | VARCHAR2(10 BYTE) | Foreign Key, references JOBS |
| FILLDATE | DATE | Not NULL |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| **FINANCE\_EMPLOYEES** | *Relation representing the entity class FINANCE\_EMPLOYEES, subclass of EMPLOYEES* | |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| CFACERTIFIED | CHAR (1 BYTE) |  |
| **FOOD\_RATINGS** | *Relation representing the entity class FOOD\_RATINGS, subclass of RATINGS* | |
| RATINGID | VARCHAR2(10 BYTE) | Foreign Key, references RATINGS |
| NUMFOODSTARS | VARCHAR2(1 BYTE) |  |
| **HUMAN\_RESOURCES\_EMPLOYEES** | *Relation representing the entity class HUMAN\_RESOURCES\_EMPLOYEES, subclass of EMPLOYEES* | |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| COMMUNICATION\_SKILLS | VARCHAR2(50 BYTE) |  |
| **INTERNAL\_CAMPAIGN** | *Relation representing the entity class INTERNAL\_CAMPAIGNS, subclass of CAMPAIGN* | |
| CAMPAIGNID | VARCHAR2(10 BYTE) | Foreign Key, references CAMPAIGNS |
| ROOMNUM | VARCHAR2(50 BYTE) |  |
| OFFICELOC | VARCHAR2(50 BYTE) |  |
| **ITEMS** | *Relation representing the entity class ITEMS* | |
| ITEMID | VARCHAR2(10 BYTE) | Primary Key |
| PRICE | NUMBER (5,2) |  |
| ITEMDESCRIPTION | VARCHAR2(500 BYTE) |  |
| DIETARYRESTRICTIONS | VARCHAR2(50 BYTE) |  |
| RESTID | VARCHAR2(10 BYTE) | Foreign Key, references RESTAURANTS |
| **JOB\_POSTINGS** | *Relation representing the entity class JOB\_POSTINGS* | |
| JOBPOSTINGID | VARCHAR2(10 BYTE) | Primary Key |
| JOBPOSTDATE | DATE |  |
| JOBDESCRIPTION | VARCHAR2(50 BYTE) | Not NULL |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| **JOBS** | *Relation representing the entity class JOBS* | |
| JOBID | VARCHAR2(10 BYTE) | Primary Key |
| JOBPOSITION | VARCHAR2(50 BYTE) | Not NULL |
| JOBSTATUS | VARCHAR2(50 BYTE) |  |
| **LEADS** | Relation representing the entity class LEADS | |
| LEADID | VARCHAR2(10 BYTE) | Primary Key |
| LEADNAME | VARCHAR2(50 BYTE) | Not NULL |
| LEADCONTACTFNAME | VARCHAR2(50 BYTE) | Not NULL |
| LEADCONTACTLNAME | VARCHAR2(50 BYTE) | Not NULL |
| LEADCONTACTTITLE | VARCHAR2(50 BYTE) |  |
| LEADCONTACTPHONE | NUMBER (10,0) | Not NULL, CHECK (leadContactPhone not like '% [^0-9]%'),CHECK (LENGTH(leadContactPhone) = 10) |
| EVENTID | VARCHAR2(10 BYTE) | Foreign Key, references EVENTS |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| **MARKETING\_EMPLOYEES** | *Relation representing the entity class MARKETING\_EMPLOYEES, subclass of EMPLOYEES* | |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| YEARSEXP | NUMBER (2,0) |  |
| **OFFERS** | *Relation representing the entity class OFFERS* | |
| PERKID | VARCHAR2(10 BYTE) | Foreign Key, references PERKS |
| RESTID | VARCHAR2(10 BYTE) | Foreign Key, references RESTAURANTS |
| **OPEN\_JOBS** | *Relation representing the entity class OPEN\_JOBS* | |
| JOBID | VARCHAR2(10 BYTE) | Foreign Key, references JOBS |
| CLOSEDATE | DATE | Not NULL |
| JOBPOSTINGID | VARCHAR2(10 BYTE) | Foreign Key, references JOB\_POSTINGS |
| **ORDERS** | *Relation representing the entity class ORDERS* | |
| ORDERID | VARCHAR2(10 BYTE) | Primary Key |
| SPECINSTRUCT | VARCHAR2(50 BYTE) |  |
| STATUS | VARCHAR2(50 BYTE) |  |
| ORDERDATE | DATE |  |
| TOTALCOST | NUMBER (5,2) |  |
| RESTID | VARCHAR2(10 BYTE) | Foreign Key, references RESTAURANTS |
| PERKID | VARCHAR2(10 BYTE) | Foreign Key, references PERKS |
| CUSTID | VARCHAR2(10 BYTE) | Foreign Key, references CUSTOMERS |
| DELIVERYID | VARCHAR2(10 BYTE) | Foreign Key, references DRIVER\_DELIVERIES |
| **ORDERS\_ITEMS** | *Relation representing the entity class ORDERS\_ITEMS* | |
| ORDERID | VARCHAR2(10 BYTE) | Foreign Key, references ORDERS |
| ITEMS | VARCHAR2(50 BYTE) | Foreign Key, references ITEMS |
| **PARTNER CONTRACTS** | *Relation representing the entity class ORDERS\_ITEMS* | |
| CONTRACTID | VARCHAR2(10 BYTE) | Primary Key |
| CONTRACTEXPDATE | DATE | Not NULL |
| CONTRACTSTARTDATE | DATE | Not NULL |
| STATUS | VARCHAR2(20 BYTE) |  |
| CORPPARTID | VARCHAR2(10 BYTE) | Foreign Key, references CORPORATE\_PARTNERS |
| **PAYMENTS** | *Relation representing the entity class PAYMENTS* | |
| PAYMENTID | VARCHAR2(10 BYTE) | Primary Key |
| CUSTPAIDAMT | NUMBER (10,2) |  |
| PROFITEARNED | NUMBER (10,2) |  |
| DRIVERRCVDAMT | NUMBER (10,2) |  |
| RESTRCVDAMT | NUMBER (10,2) |  |
| ORDERID | VARCHAR2(10 BYTE) | Foreign Key, references ORDERS |
| **PERKS** | *Relation representing the entity class PERKS* | |
| PERKID | VARCHAR2(10 BYTE) | Primary Key |
| PROMOAMOUNT | NUMBER (2,0) |  |
| PROMOCODE | VARCHAR2(10 BYTE) |  |
| PERKNAME | VARCHAR2(50 BYTE) |  |
| PROMOENDDATE | DATE |  |
| PROMOSTARTDATE | DATE |  |
| **PICKUP\_RESTAURANTS** | *Relation representing the entity class PICKUP\_RESTAURANTS, subclass of RESTAURANTS* | |
| RESTID | VARCHAR2(10 BYTE) | Foreign Key, references RESTAURANTS |
| PICKUPTIME | VARCHAR2(10 BYTE) |  |
| **PROBLEM\_TICKET** | *Relation representing the entity class PROBLEM\_TICKET* | |
| TICKETID | VARCHAR2(10 BYTE) | Primary Key |
| STATUS | VARCHAR2(15 BYTE) |  |
| DATERESOLVED | DATE | Not NULL |
| DATECREATED | DATE | Not NULL |
| PROBID | VARCHAR2(10 BYTE) | Foreign Key, references PROBLEMS\_CATEGORY |
| CUSTID | VARCHAR2(10 BYTE) | Foreign Key, references CUSTOMERS |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| **PROBLEM\_TYPES** | *Relation representing the entity class PROBLEM\_TYPES* | |
| PROBID | VARCHAR2(10 BYTE) | Primary Key |
| PROBRESOLUTION | VARCHAR2(255 BYTE) |  |
| PROBDESCRIPTION | VARCHAR2(50 BYTE) | Not NULL |
| **PROBLEMS\_CATEGORY** | *Relation representing the entity class PROBLEM\_CATEGORY* | |
| PROBID | VARCHAR2(10 BYTE) | Primary Key |
| PROBCATEGORY | VARCHAR2(50 BYTE) |  |
| **PROMOTE** | *Relation representing the entity class PROMOTE* | |
| EVENTID | VARCHAR2(10 BYTE) | Foreign Key, references EVENTS |
| CAMPAIGNID | VARCHAR2(10 BYTE) | Foreign Key, references CAMPAIGNS |
| **RATINGS** | *Relation representing the entity class RATINGS* | |
| RATINGID | VARCHAR2(10 BYTE) | Primary Key |
| CUSTCOMMENTS | VARCHAR2(255 BYTE) |  |
| ORDERID | VARCHAR2(10 BYTE) | Foreign Key, references ORDERS |
| **RESTAURANT\_ACCOUNTS** | *Relation representing the entity class RESTAURANT\_ACCOUNTS* | |
| ACCTID | VARCHAR2(10 BYTE) | Primary Key |
| ACC\_USERNAME | VARCHAR2(20 BYTE) | Not NULL |
| ACC\_PASSWORD | VARCHAR2(20 BYTE) | Not NULL |
| BANKACCTNUM | VARCHAR2(10 BYTE) |  |
| EMPID | VARCHAR2(10 BYTE) |  |
| RESTID | VARCHAR2(10 BYTE) |  |
| **RESTAURANT\_HOURS** | *Relation representing the entity class RESTAURANT\_HOURS* | |
| RESTID | VARCHAR2(10 BYTE) | Foreign Key, references RESTAURANT\_HOURS |
| WEEKDAY | VARCHAR2(10 BYTE) |  |
| OPENTIME | VARCHAR2(10 BYTE) |  |
| CLOSETIME | VARCHAR2(10 BYTE) |  |
| **RESTAURANTS** | *Relation representing the entity class RESTAURANTS* | |
| RESTID | VARCHAR2(10 BYTE) | Primary Key |
| RESTNAME | VARCHAR2(50 BYTE) | Not NULL |
| REST\_CATEGORY | VARCHAR2(50 BYTE) |  |
| PHONENUM | NUMBER (12,0) | CHECK (phoneNum not like '%[^0-9]%'),CHECK (LENGTH(phoneNum) = 10) |
| STREET | VARCHAR2(50 BYTE) |  |
| CITY | VARCHAR2(50 BYTE) |  |
| RESTSTATE | VARCHAR2(2 BYTE) |  |
| ZIP | VARCHAR2(5 BYTE) | Not NULL |
| CHAINID | VARCHAR2(10 BYTE) | Foreign Key, references CHAINS |
| **REVIEW\_BY** | *Relation representing the entity class REVIEW\_BY* | |
| APPLICATIONID | VARCHAR2(10 BYTE) | Foreign Key, references APPLICATIONS |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| **SALES\_EMPLOYEES** | *Relation representing the entity class SALES\_EMPLOYEES, subclass of EMPLOYEES* | |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| SALESTARGET | NUMBER (10,0) |  |
| **SCHEDULE** | *Relation representing the entity class SCHEDULE* | |
| SCHEDULEID | VARCHAR2(10 BYTE) | Primary Key |
| SCHEDULE\_DATE | DATE |  |
| STARTTIME | VARCHAR2(10 BYTE) |  |
| ENDTIME | VARCHAR2(10 BYTE) |  |
| DRIVERID | VARCHAR2(10 BYTE) | Foreign Key, references DRIVERS |
| **SERVE** | *Relation representing the entity class SERVE* | |
| CUISINEID | VARCHAR2(10 BYTE) | Foreign Key, references CUISINE |
| RESTID | VARCHAR2(10 BYTE) | Foreign Key, references RESTAURANTS |
| **SOFTWARE\_DEV\_EMPLOYEES** | *Relation representing the entity class SOFTWARE\_DEV\_EMPLOYEES, subclass of EMPLOYEES* | |
| EMPID | VARCHAR2(10 BYTE) | Foreign Key, references EMPLOYEES |
| SPECIALITY | VARCHAR2(20 BYTE) |  |

# **[Normalized Schema](#_Table_of_Contents)**

Once we had created the ER Diagram and were ready to begin the table insertion, we had to normalize the tables to determine functional dependencies. The normalization process led us to create seven additional tables for many to many relationships, as well as map foreign keys to their corresponding entities. During this process, we also added various check and unique constraints to handle situations where we wanted to prevent particular data from being inserted and duplicate data.

**Normalized Schema**

APPLICATIONS(applicationID, *applicantID*, resume)

FOREIGN KEY applicantID *references* APPLICANTS (applicantID)

APPLICANTS(applicantID, firstName, lastName, DOB, age, highestDegree)

APPLY\_TO(applicationID, jobPostingID)

FOREIGN KEY (applicationID) references APPLICATIONS (applicationID)

FOREIGN KEY (jobPostingID) references JOB\_POSTINGS (jobPostingID)

CAMPAIGNS(campaignID, campaignName)

CATERING\_RESTAURANT(restID, pricePerHead, maxPartySize, minPartySize, requiredLeadTime, downPaymentAmt)

FOREIGN KEY (restID) references RESTAURANTS (restID)

CHAINS(chainID, website, chainName)

COMPRISE\_HUMAN\_RESOURCES(eventTeamID, empID)

FOREIGN KEY (eventTeamID) references EVENT\_TEAMS (eventTeamID)

FOREIGN KEY (empID) references EMPLOYEES (empID)

COMPRISE\_MARKETING(eventTeamID, empID)

FOREIGN KEY (eventTeamID) references EVENT\_TEAMS (eventTeamID)

FOREIGN KEY (empID) references EMPLOYEES (empID)

COMPRISE\_SALES(eventTeamID, empID)

FOREIGN KEY (eventTeamID) references EVENT\_TEAMS (eventTeamID)

FOREIGN KEY (empID) references EMPLOYEES (empID)

CORPORATE\_PARTNERS(corpPartID, name, serviceProvided, empID)

FOREIGN KEY (empID) references SALES (empID)

CREATES(empID, campaignID)

FOREIGN KEY (empID) references EMPLOYEES (empID)

FOREIGN KEY (campaignID) references CAMPAIGNS (campaignID)

CUISINES(cuisineID, cuisineType, region)

CUSTOMERS(custID, firstname, lastname, street, city, Zip, custState, phoneNum, custEmail)

CUSTOMERS\_FAVORITES(custID, favorites)

FOREIGN KEY (custID) references CUSTOMERS (custID)

CUSTOMERS\_PAYMETHOD(custID, payMethod)

FOREIGN KEY (custID) references CUSTOMERS (custID)

CUSTOMER\_SERVICE\_EMPLOYEES(empID, custReviewRating)

FOREIGN KEY (empID) references EMPLOYEES (empID)

DELIVERY\_RESTAURANTS (restID, delCost, minOrderPrice)

FOREIGN KEY (restID) references RESTAURANTS (restID)

DEVELOP(empID, perkID)

FOREIGN KEY (empID) references EMPLOYEES (empID)

FOREIGN KEY (perkID) references PERKS (perkID)

DRIVER\_DELIVERIES(deliveryID, clockInDate, clockInTime, clockOutDate, clockOutTime, miles,numDelivery, *driverID*)

FOREIGN KEY (driverID) references DRIVERS (driverID)

DRIVERS(driverID, city, driverState, street, Zip, bankAcctNum, dob, driverRewards, email, firstName, ssn, maxRange, phoneNum,insurancePNum, lastName, licNum)

DRIVER\_RATINGS(ratingID, numStars)

FOREIGN KEY (ratingID) references RATINGS (ratingID)

EMPLOYEES(empID, name, phoneNum, street, city, empState, zip, email, joinDate, DOB, age, empComp, wageType*,* tenure, departmentName)

EVENTS(eventID, street, state, city, zip, numAttend, *eventTeamID*)

FOREIGN KEY (eventTeamID) references EVENT\_TEAMS (event\_Team\_ID)

EVENT\_TEAMS(eventTeamID, eventTeamName, creationDate)

EXTERNAL\_CAMPAIGN(campaignID, ecType, ecMedium)

FOREIGN KEY(campaignID) references CAMPAIGNS (campaignID)

FILLED\_JOBS(jobID, *empID,* fillDate)

FOREIGN KEY jobID *references* JOBS (jobID)

FOREIGN KEY empID *references* EMPLOYEES (empID)

FINANCE\_EMPLOYEES(empID, CFACertified)

FOREIGN KEY (empID) references EMPLOYEES (empID)

FOOD\_RATINGS(ratingID, numFoodStars)

FOREIGN KEY (ratingID) references RATINGS (ratingID)

HUMAN\_RESOURCES\_EMPLOYEES(empID, communicationSkills)

FOREIGN KEY (empID) references EMPLOYEES (empID)

INTERNAL\_CAMPAIGN(campaignID, roomNum, officeLoc)

FOREIGN KEY(campaignID) references CAMPAIGNS (campaignID)

ITEMS(itemID, price, description, itemName, dietaryRestrictions, *restID, chainID*)

FOREIGN KEY (restID)  references RESTAURANT (restID)

FOREIGN KEY (chainID)  references CHAINS (chainID)

JOB\_POSTINGS(jobPostingID, jobPostDate, jobDescription, *empID*)

FOREIGN KEY (empID) references EMPLOYEES (empID)

JOBS(jobID, jobPosition, jobStatus, *jobPostingID*)

FOREIGN KEY (jobPostingID) JOB\_POSTINGS (jobPostingID)

LEADS(leadID, leadName, leadContactFName, leadContactLName, leadContactTitle, leadContactPhone, *eventID*, *empID*)

FOREIGN KEY (eventID) references EVENTS (eventID)

FOREIGN KEY (empID) references EMPLOYEES (empID)

MARKETING\_EMPLOYEES(empID, yearsExp)

FOREIGN KEY (empID) references EMPLOYEES (empID)

ORDERS(orderID, specInstruct, status, orderDate, totalCost, *restID, perkID*, *custID, deliveryID*)

FOREIGN KEY (restID) references RESTAURANTS (restID)

FOREIGN KEY (perkID) references PERKS (perkID)

FOREIGN KEY (custID) references CUSTOMERS (custID)

FOREIGN KEY (deliveryID) DRIVER\_DELIVERIES (deliveryID)

OFFERS(perkID, restID)

FOREIGN KEY (perkID) references PERKS (perkID)

FOREIGN KEY (restID) references RESTAURANTS (restID)

OPEN\_JOBS(jobID, closeDate, *jobPostingID*)

FOREIGN KEY jobID *references* JOBS (jobID)

FOREIGN KEY jobPostingID *references* JOB\_POSTINGS (jobPostingID)

ORDERS\_ITEMS(orderID, items)

FOREIGN KEY (orderID) references ORDERS (orderID)

PARTNER\_CONTRACTS(contractID, contractExpDate, contractStartDate, status, *corpPartID*)

FOREIGN KEY (corpPartID) references CORPORATE\_PARTNERS (corpPartID)

PAYMENTS(paymentID, custPaidAmt, profitEarned, driverRcvdAmt, restRcvdAmt, *orderID*)

FOREIGN KEY (orderID) references ORDERS (orderID)

PERKS(perkID, promoAmount, promoCode, perkName, promoEndDate, promoStartDate)

PICKUP\_RESTAURANT(restID, pickupTime)

FOREIGN KEY (restID) references RESTAURANTS (restID)

PROBLEMS\_CATEGORY(probID,probCategory)

FOREIGN KEY (probID) reference PROBLEM\_TYPES (probID)

PROBLEM\_TICKET(ticketID, status, dateResolved, priority, dateCreated, *probID, custID, empID*)

FOREIGN KEY (probID) references PROBLEM\_TYPES (probID)

FOREIGN KEY (custID) references CUSTOMERS (custID)

FOREIGN KEY (empID) references CUSTOMER\_SERVICE\_EMPLOYEES (empID)

PROBLEM\_TYPES(probID, probResolution, probDescription)

PROMOTE(eventID, campaignID)

FOREIGN KEY (eventID) references EVENTS (eventID)

FOREIGN KEY (campaignID) references EXTERNAL\_CAMPAIGNS (campaignID)

RATINGS(ratingID, custComments, *orderID*)

FOREIGN KEY (orderID) references ORDERS (orderID)

RESTAURANTS(restID, restName, category, phoneNum, street, city, restState, zip, *chainID*)

FOREIGN KEY (chainID) references CHAINS (chainID)

RESTAURANT\_HOURS(restID,weekday, openTime, closeTime)

FOREIGN KEY (restID) reference RESTAURANTS (restID)

RESTAURANT\_ACCOUNTS(acctID, userName, password, bankAcctNum, *empID*, *restID*)

FOREIGN KEY (empID) references FINANCE\_EMPLOYEES (empID)

FOREIGN KEY (restID) references RESTAURANTS (restID)

REVIEW\_BY(applicationID, empID)

FOREIGN KEY (applicationID) references  APPLICATIONS (applicationID)

FOREIGN KEY (empID) references HUMAN\_RESOURCES\_EMPLOYEES (empID)

SALES\_EMPLOYEES(empID, salesTarget)

FOREIGN KEY (empID) references EMPLOYEES (empID)

SCHEDULE(scheduleID, date, startTime, endTime, *driverID*)

FOREIGN KEY (driverID) references DRIVERS (driverID)

SERVE(cuisineID, restID)

FOREIGN KEY (cuisineID) references CUISINES (cuisineID)

FOREIGN KEY (restID) references RESTAURANTS (restID)

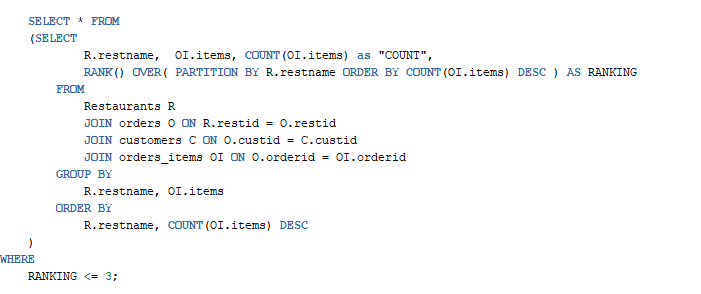
SOFTWARE\_DEV\_EMPLOYEES(empID, speciality)

FOREIGN KEY (empID) references EMPLOYEES (empID)

# **[Complex Queries](#_Table_of_Contents)**

# **[Query 1:](#_Table_of_Contents)**

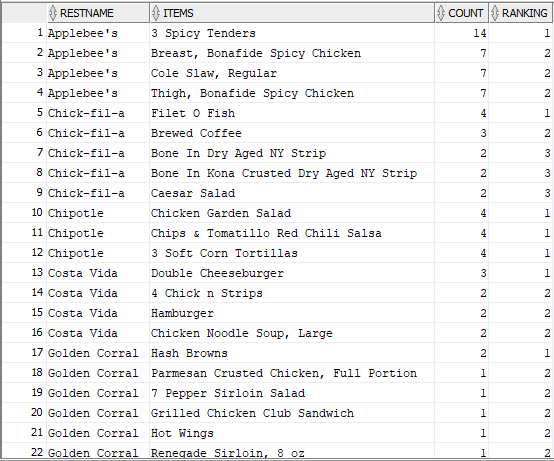
Grubhub requires sales information across all restaurants and customers. For each restaurant in the United States, display the restaurant name, item name, and the number of times that item has been ordered. Rank each item by the number of times it has been ordered and limit the Rank to three.



**Explanation:**

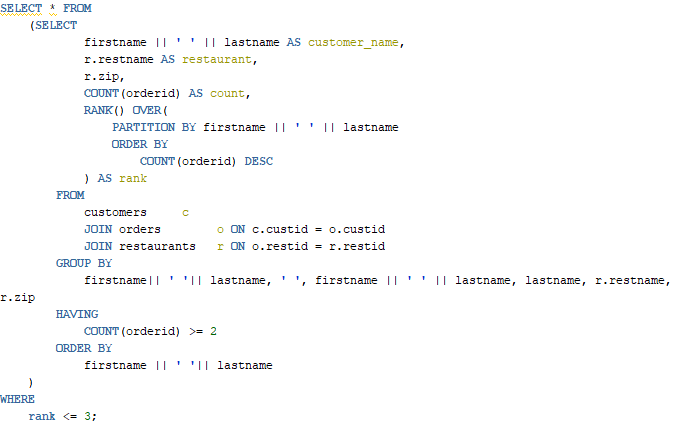
The RANK function to rank the count of item names across all restaurants where food orders have been placed. DESC is used to rank the count in descending order from highest to lowest. The JOINS are made between Orders and Restaurants to map each order that has been placed with that restaurant. Each item ordered per OrderID is mapped in a different table orders\_items. The WHERE clause has a limit of up to 3 because we only want to display three ranks. Lastly, the SELECT statement displays the restaurant name, item name from orders\_items, and the corresponding count of that item across all orders.

**Output:**



# **[Query 2:](#_Table_of_Contents)**

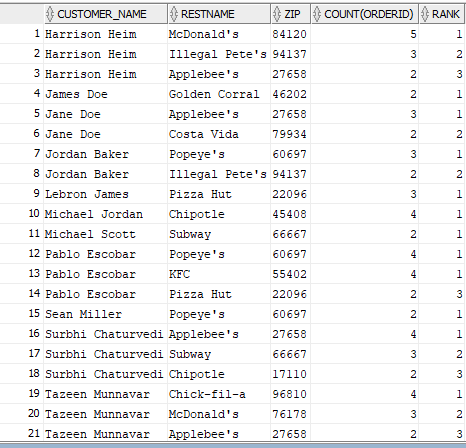
Grubhub collects information on the customer's top restaurant preferences. This information will help Grubhub to create perks and promotional campaigns based on the number of times an order has been placed. For each customer, we display the number of times the order has been placed, and its ranking among orders from other restaurants.



**Explanation**:

We have used RANK to classify count of orders per customer, with the maximum count as ranked no.1. We have joined the customers, orders, restaurants tables, and this query connects every order with its respective restaurant name. We use order by DESC to rank the counts with maximum count as rank 1. We use customers, orders, and restaurants to display.

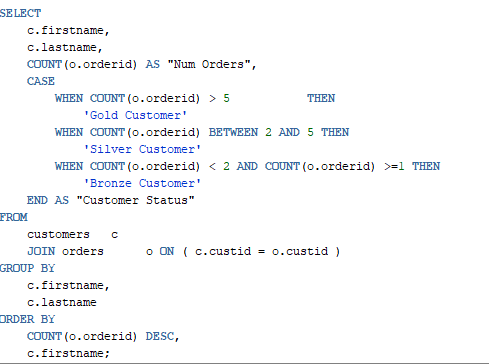
**Output:**



# **[Query 3:](#_Table_of_Contents)**

Grubhub incentivizes customers based on the number of orders they have placed in the past. For every customer, we categorize people into three customer categories:

1. Gold where the minimum number of orders required is 5.
2. Silver where orders should be between 2 and 5.
3. Bronze is awarded to those who have at least one order.



**Explanation:**

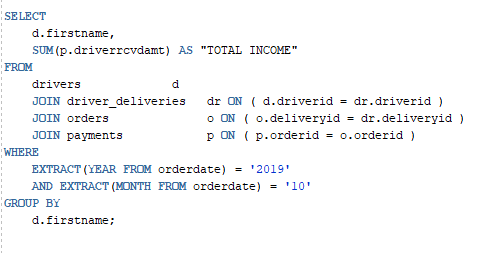
CASE is used to categorize customers into three categories - the first condition checks if the count is greater than 5, and if it is, then adds ‘Gold Customer’ under Customer Status. Similarly, for the other two categories, SILVER and BRONZE is listed for every customer based on the conditions. Two tables - customers and orders are joined on custID to count their orders. We use ORDER BY Num Orders (descending order) and the first name (ascending order) for the output table.

**Output:**



# **[Query 4:](#_Table_of_Contents)**

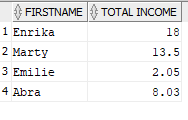
Grubhub drivers can log in and check their monthly income based on the orders they delivered in that calendar year. This example shows the income for all drivers who have delivered orders in October in 2019. From the user interface, the drivers can select the particular month and year to check their income for that month and year.



**Explanation**:

The code extracts the year and date from the orderdate. The JOIN statement joins driver\_deliveries, orders, and payments on driverID, deliveryID, and orderID, respectively.

**Output:**



# **[Query 5:](#_Table_of_Contents)**

Grubhub checks the income of each chain periodically. The query uses PIVOT to display all restaurants as columns and their corresponding income from January 1, 2019.



**Explanation:**

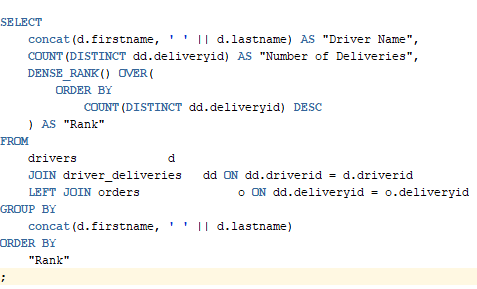
The chains are connected with restaurants on chainID, and each restaurant is connected with orders using restaurantID. The payments table is related to each order using OrderID. ‘Restrcvdamt’ is the variable that stores the amount earned by each restaurant per order. Pivot function selects all variables and values in one column and performs aggregations on remaining column values.

**Output:**



# **[Query 6:](#_Table_of_Contents)**

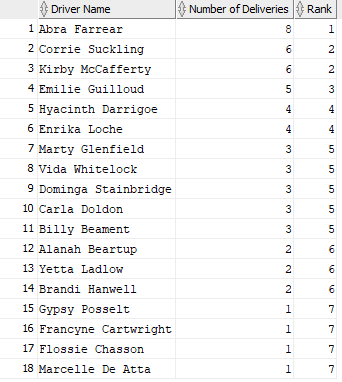
Grubhub ranks each driver by the number of deliveries they do each month. RANK function assigns a position each month. This is used for incentivizing the driver and give rewards at the end of each quarter.



**Explanation:**

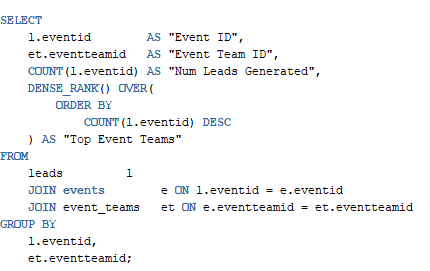
We use Dense\_Rank to rank all drivers with respect to the number of deliveries they have completed each month for the past six months. We join drivers and driver\_deliveries on driverID and left join orders on deliverID. We use left join because some orders are pick up orders.

**Output:**



# **[Query 7:](#_Table_of_Contents)**

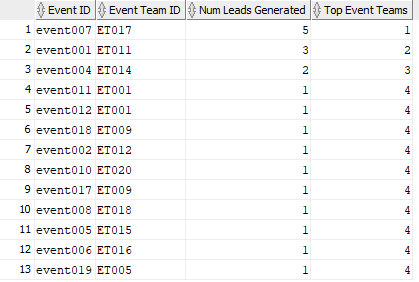
Grubhub generates customers, and corporate leads through events. Each event performance needs to be tracked by the number of leads generated. The top-ranked teams are rewarded. The following query gives a count of the leads generated, grouped by event ID and Event team ID.



**Explanation:**

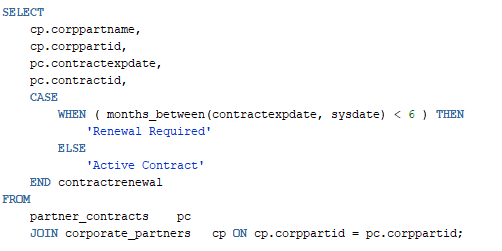
The query uses leads, events, and event\_teams tables and joins them using eventID and eventteamID. We count the number of leads generated by counting the eventID inside the leads table. The RANK function ranks in descending order with Rank 1 for the team that has generated the most leads.

**Output:**



# **[Query 8:](#_Table_of_Contents)**

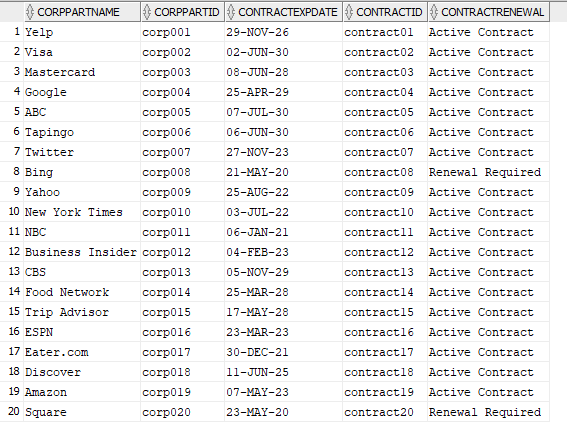
Grubhub has corporate tie-ups with multiple companies. Each corporate client is bound legally with a contract, and each contract has an expiration date. Grubhub would like to monitor contract status for each client, stating whether the agreement is Active or if there is a Renewal Required.



**Explanation:**

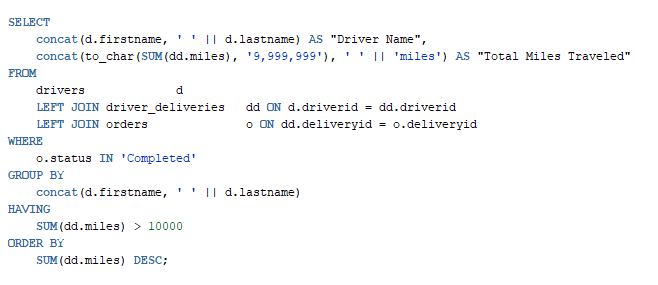
The query starts with a case statement, checking the validity of the contract with the current date (sysdate). If the contract expires within six months, then a renewal is required; otherwise, it is active. The query uses corporate\_partners and partner\_Contracts joining on corppartID.

**Output:**



# **[Query 9:](#_Table_of_Contents)**

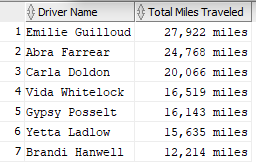
Grubhub tracks the number of miles traveled by each driver. Drivers are incentivized based on the number of miles traveled. Grubhub keeps track of the total miles traveled since the driver began making deliveries for Grubhub. Grubhub wishes to track drivers with a minimum of 10000 miles and excludes orders that are picked up by the customer.



**Explanation**:

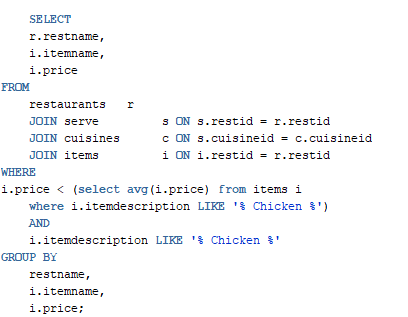
The query displays a list of drivers who have completed a minimum of 10,000 miles. Left Join is used with drivers because we do not include orders that are self-pickup. Additionally, we check that the delivery status is complete.

**Output:**



# **[Query 10:](#_Table_of_Contents)**

Grubhub periodically checks the average price of an item with certain ingredients compared with its average price across all restaurants in the United States. The below example takes Chicken as an ingredient and lists only those items that are priced below the average price across all restaurants in the United States.



**Explanation:**

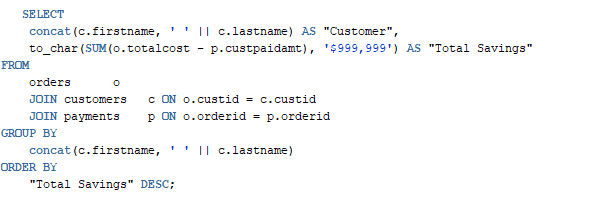
The query checks the price of an ingredient, say, Chicken, across item descriptions under items table, this table consists of menu across all chains, including its corresponding price. The WHERE clause checks for an average price of chickens across all items and compares it with each item price. The tables that we use are serve, cuisines, items, which are joined on restID, cuisineID, restID.

**Output:**



# **[Query 11:](#_Table_of_Contents)**

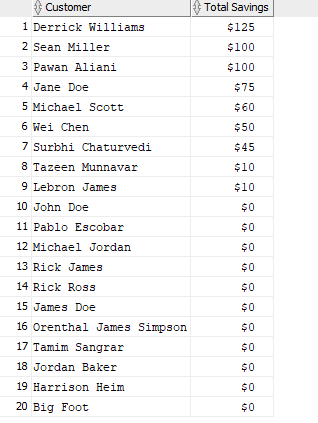
Grubhub wishes to track all the savings made by each customer. This is required by the company in order to maintain a record of their offerings and also inform customers as feedback after every month or quarter.



**Explanation:**

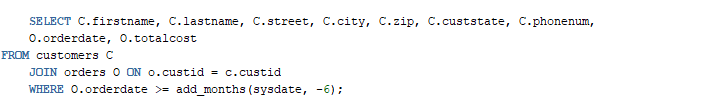
This query displays customer name and their respective cumulative savings in dollar amount for each order placed. The query specifically takes account of those orders where a promo code has been used. The query uses the orders, customers, and payment tables joined using custID and orderID and are displayed with the customer who has accrued the most savings.

**Output:**

****

# **[Query 12:](#_Table_of_Contents)**

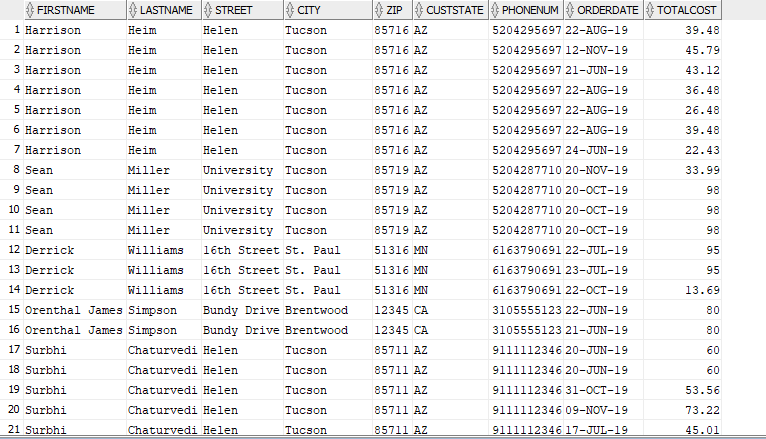
Grubhub wishes to monitor customers who have ordered every month continuously for six months.



**Explanation:**

The query displays customer demographics, along with their order cost and total cost. We have created a where clause checks if an order has been placed consecutively for six months by that customer.

**Output:**



# **[Complex Triggers and Procedures](#_Table_of_Contents)**

# **[Payments Trigger](#_Table_of_Contents)**

When a customer places an order, a tuple with the order details is stored in the “Orders” relation. There are two types of orders:

● Delivery Order

● Pickup Order

If the order is a Delivery Order, a delivery ID is assigned to it (deliverid column of figure 1). The customers are eligible for a discount if they apply a valid perk code (shown in the perkid column of Figure 1).

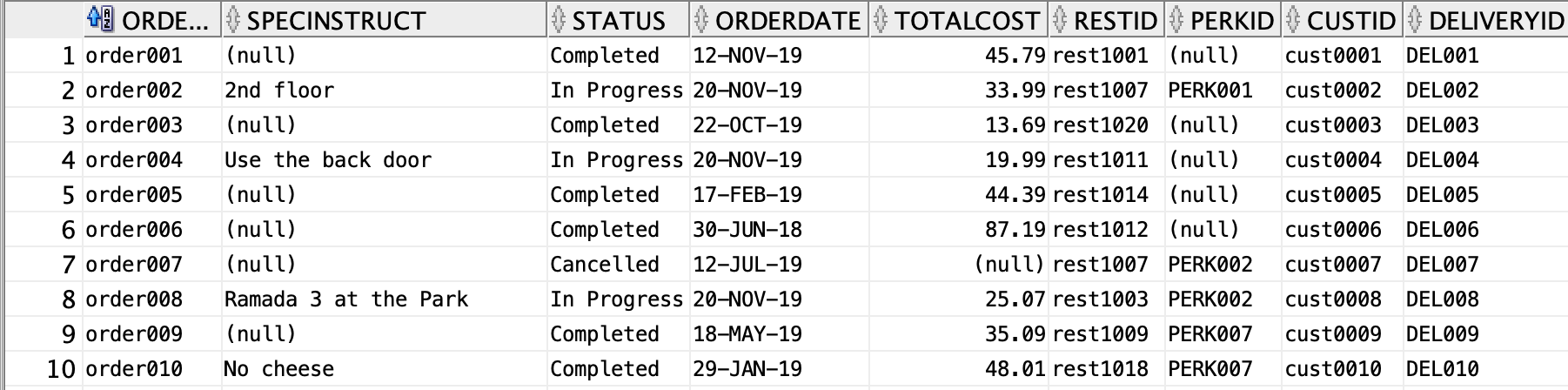


Figure 1

A trigger is fired after an insert into the orders table, such that the customer paid amount can be divided between the restaurant, driver as well as Grubhub. This data is stored in the payments table (details in Figure 2). Our business model is designed such that a proportion of the amount paid by the customer for their order goes to:

● Grubhub as profit (10%)

● The restaurant (75%)

● The driver (15%)

If the order is a pickup order, then the restaurant would receive 90% of the total amount, and the driver field would be null.

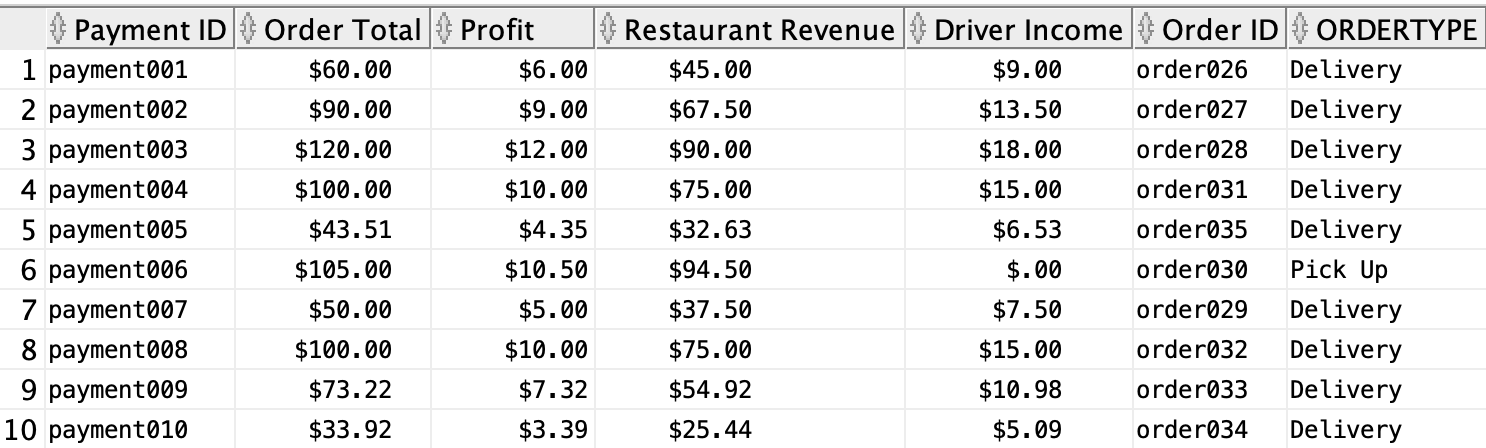


Figure 2

A sequence is created on the Payments table such that the minimum value is 001, and it increments by 1. If a customer has applied a valid perk to their order, using the perkid (from Figure 1), a discount is given to them on their order total. The validity of the perk is verified for that order by checking if the order date lies between the perk start and end date from the perks table (Figure 3).

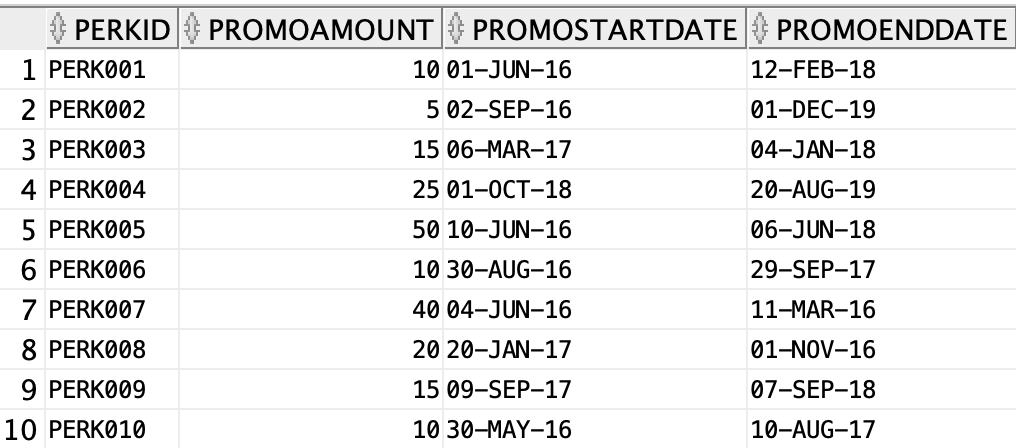


Figure 3

From Figure 2,

1. Example 1: Delivery Order

PaymentID = payment008

Order Total = $100.00

* Profit earned by Grubhub = 10% \* 100 = $10.00
* Restaurant revenue = 75% \* 100 = $75.00
* Driver Income = 15% \* 100 = $15.00

1. Example 1: Pickup Order

PaymentID = payment006

Order Total = $105.00

* Profit earned by Grubhub = 10% \* 100 = $10.50
* Restaurant revenue = 90% \* 100 = $94.50
* Driver Income = $0.00

**Code:**

Step 1:

A Payment ID sequence and a trigger after insert on orders is created with all the necessary variables declared in the DECLARE section of the code. The required data is selected and stored in the variables using SELECT.. INTO.. after the BEGIN block of the code (Refer to Figure 4).



Figure 4

Step 2:

The conditions checked for are:

● Perk Applied

○ Order date should be within perk start date and perk end date

● Delivery Order

● Pickup Order

If the order status is “In Progress” or “Completed,” then the Payments table is automatically populated based on the conditions satisfied by the order. If the order status is “Cancelled,” then there is no entry in the Payments table (Refer to Figure 5).

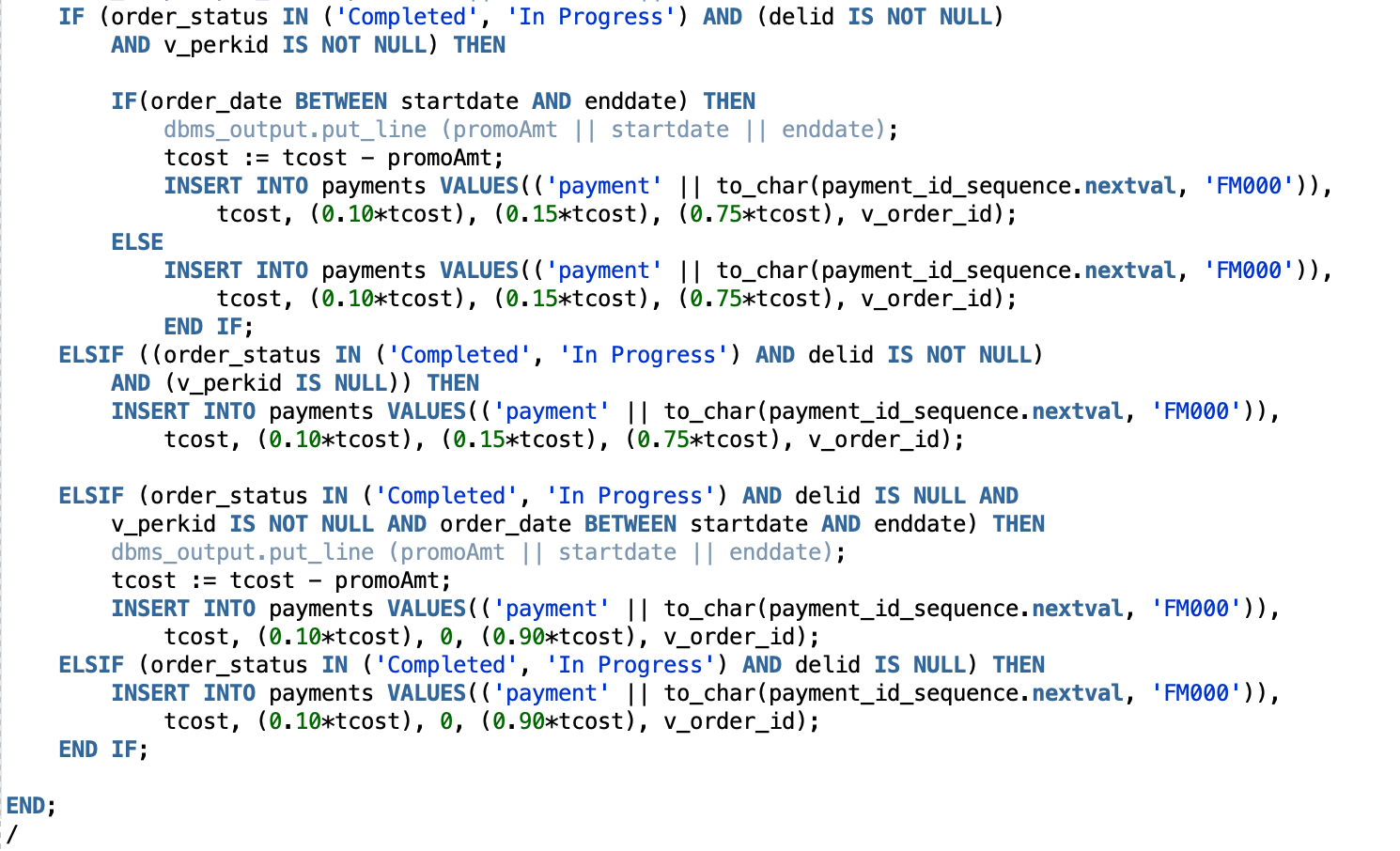
****

Figure 5

# **[Customer Favorites Procedure](#_Table_of_Contents)**

When a customer orders the same item multiple times, the item is stored in the customer favorites table against the customer’s ID (custid) as well as the restaurant ID (restid). By doing this, the customer can easily access all of their preferred items.

**Code:**

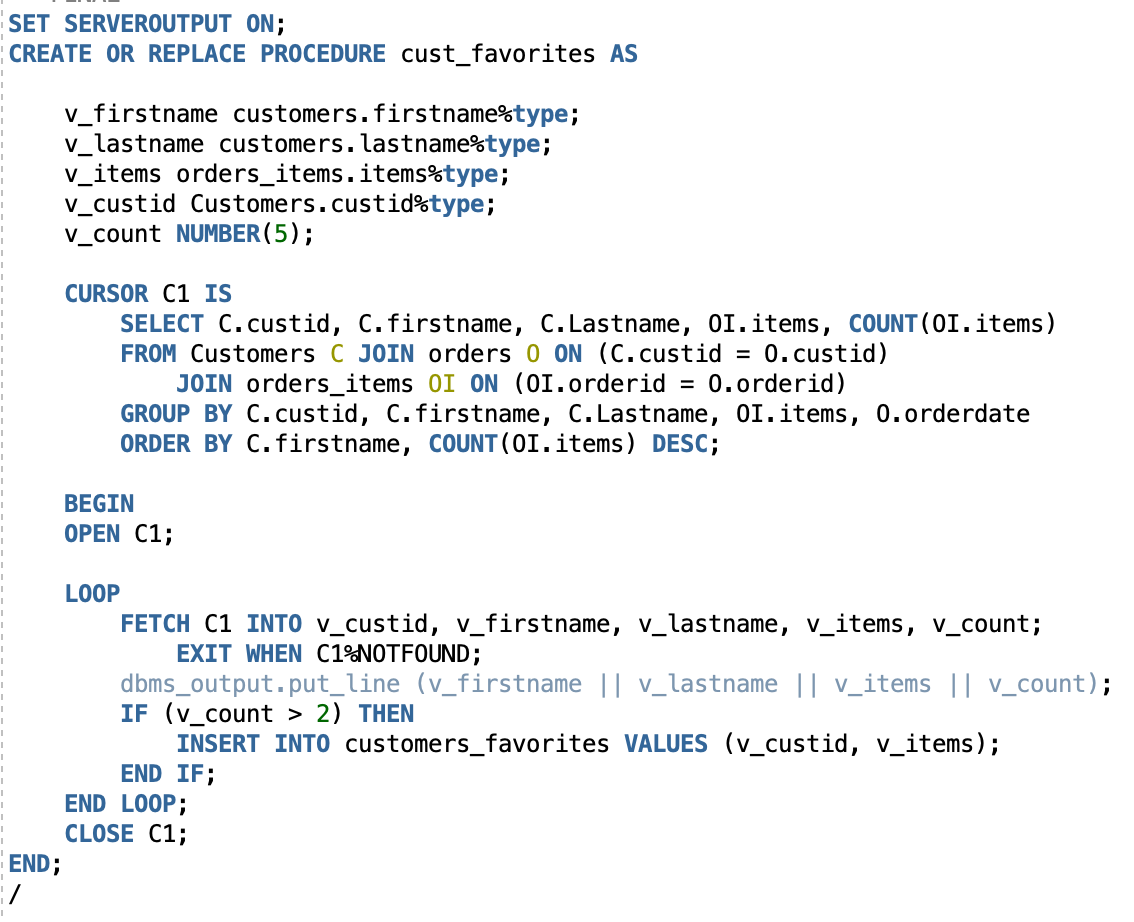


Figure 6

A cursor is created to fetch the customer details along with the items that they have ordered and the number of times they have ordered the same item. The number of items ordered is derived using the COUNT() function, and the items are taken from the orders\_items table, as shown in Figure 7 below.

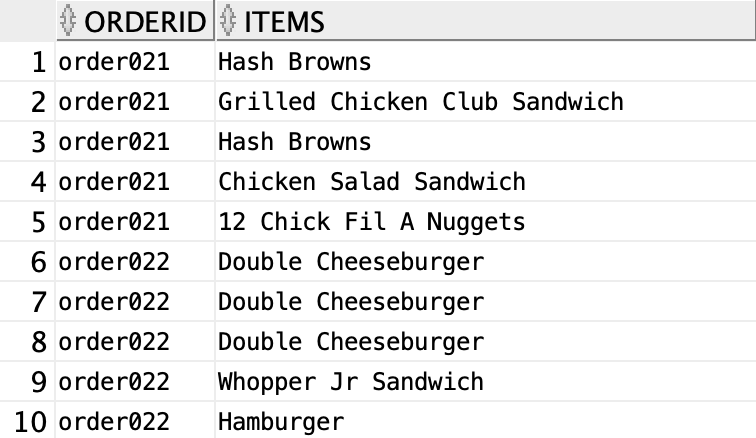


Figure 7

If the count of the items from the orders\_items table (shown in Figure 7) is greater than 2, then a new row is inserted into the customers\_favorites table. The result is displayed in Figure 8.

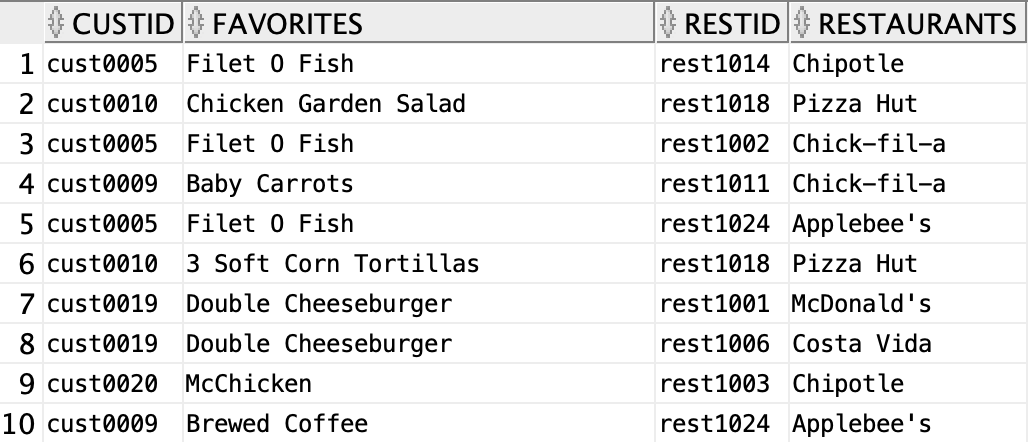


Figure 8

# **[Password Hash Trigger](#_Table_of_Contents)**

To ensure a secure login, passwords are hashed and stored in the database. The table portal\_user\_info (shown in Figure 9) contains the usernames and passwords of our users.

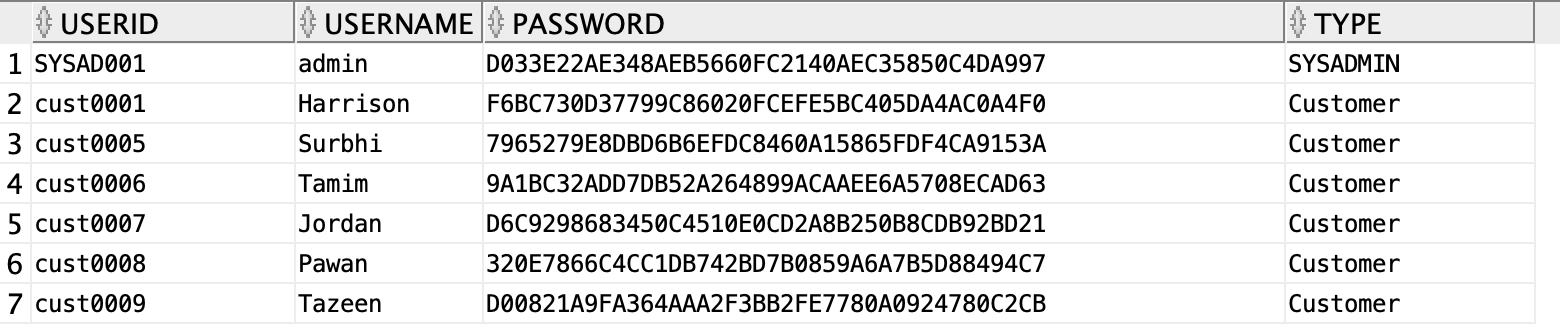


Figure 9

**Code:**

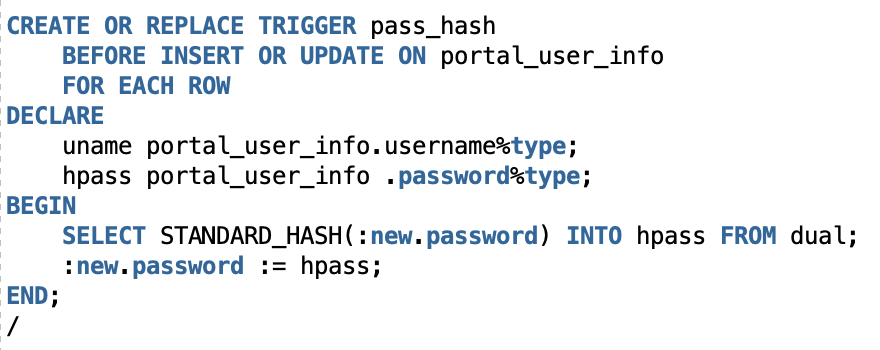
****

Figure 10

pass\_hash is a before insert on the portal\_user\_info table (shown in Figure 9). Before the userid, username, password, and the type of user is saved into the portal\_user\_info table, this trigger is fired and hashes the password. STANDARD\_HASH(:new.password) is used to hash the password, assigned to :new.password, and then saved into the table. This way, the password will not be visible to anyone who has access to the underlying tables in the database.

# **[Password Check Procedure](#_Table_of_Contents)**

The password check procedure checks if the customer has entered their correct password or not. Two parameters received from the front end - the username TextBox and the password TextBox are passed into this procedure.

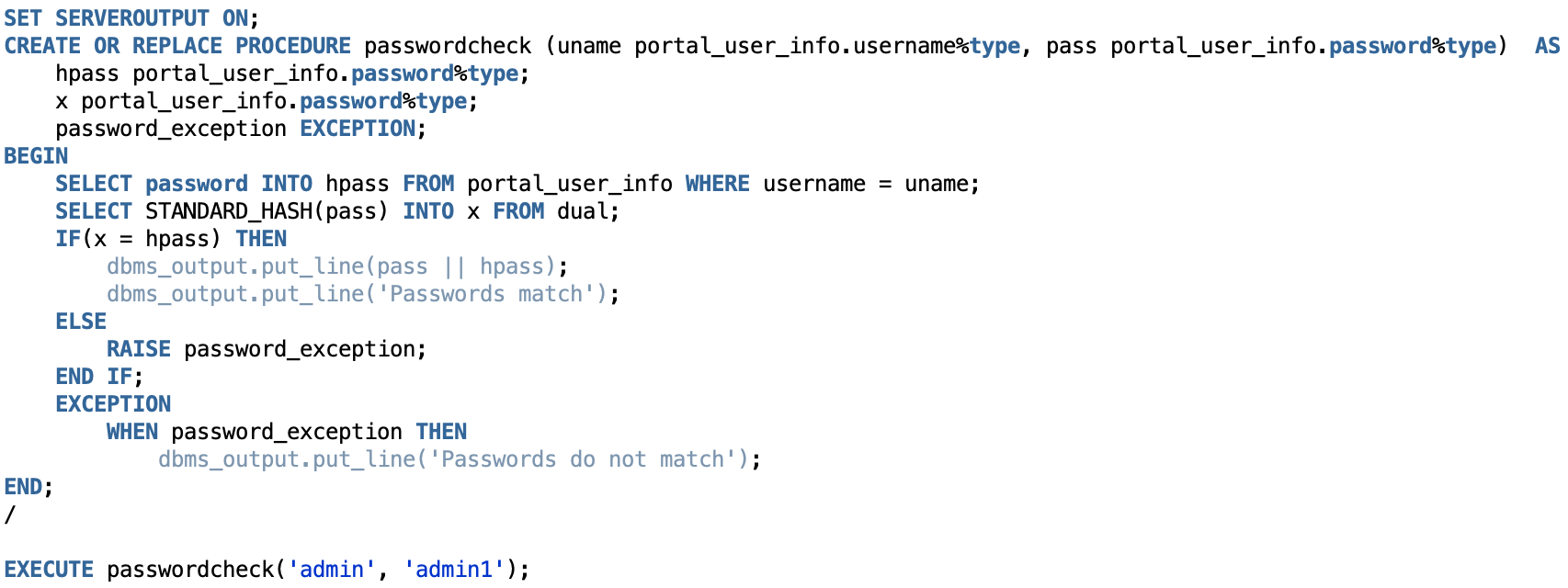


Figure 11

**Code:**

Step 1:

Retrieve the password from the portal\_user\_info table (shown in Figure 9) and save it into hpass, which is a variable that is declared.

Step 2:

The password retrieved from the frontend is passed into the procedure as a parameter, is saved as a variable - x and declared.

Step 3:

The two variables (from Step 1 and Step 2) are compared. If hpass (Step 1) is equal to x (Step 2), then the passwords match, and the user is taken to the home page of Grubhub. If the passwords do not match, then an Exception is raised that passwords do not match.

Note: The STANDARD\_HASH() function is used to hash the password for security purposes. DBMS\_CRYPTO() could not be used because of insufficient privileges.

# **[User Interface](#_Table_of_Contents)**

The technology stack used to develop the cloud-based web interface is HTML, CSS, JavaScript for the front end, and using ASP.net connected to the Oracle SQL Server for the database. We deployed the interface on Amazon Web Services.

The cloud-web interface showcases the Grubhub database as the System Administrator. To log in to the dashboard, you will need to enter the following credentials: Username = admin1 (case sensitive) Password = admin (case sensitive).

# **[Demo](#_Table_of_Contents)**

Once logged in, please refer to the video attached to proceed with the walkthrough of the interface's functionalities.

# **[Conclusion](#_Table_of_Contents)**

Through this semester-long project of designing and creating a full database for a company like Grubhub, we encountered many different and unexpected challenges. The three biggest takeaways for us have been: do not underestimate the complexity, plan for user-based issues and errors, and never assume that you have found a complete solution.

Many of us assumed that we were familiar with the ins and outs of how Grubhub handles transactions and fulfills orders. What we didn’t account for initially was how intricate the data collection process was and the entities that we had to consider. For example, we did not foresee the collection of Grubhub employees who were not drivers, yet that is an essential part of Grubhub’s business operations. Another instance we did not expect to model was the job application process and processing applicants. While it may seem trivial to collect information for individuals who may not receive a job offer, it is still important to handle that data. When it comes to processing payments, that sequence has many moving parts that require consideration and attention, such as applying discount codes and making sure that the right parties get paid for their work.

Another lesson learned through this semester has been that there will always be a customer request that you cannot account for, and you must be able to find a solution. For example, customers may want to edit an order while it is happening because they may have placed an order, and the default address is their home, but they want it delivered elsewhere. This was an issue that had not occurred to us until the presentation in class. Another customer scenario that we appeared with was partial refunds; for example, what if a customer cancels part of the order? Both of these scenarios require complicated triggers and procedures that we were unable to address at the time of submission properly. This leads to our final lesson learned.

One of the biggest lessons that we will take away is the resounding objective from this project has been that, no matter how much you plan and research, at each step, you are going to encounter issues that were not expected, and you must adapt. Many parts of our ER Diagram could have been expanded and gone into a deeper level of granularity. The more data records we entered, the more opportunities we saw for improvement. Even after our in-class presentation, we found ways to change and improve our design. If allowed, this project would have continued for many months, and even then, we would not have felt that the product would be ready for customer deployment. By having a hard deadline and getting feedback from our peers, we saw that this isn’t a finished product and never will be; it is a constant iterative process.

For this project to continue to grow and scale-up, we have collected and presented a series of economic projects that we believe illustrate what it would take to develop and maintain a digital food delivery company.

**Financial Projections**

1. Amazon Web Services (AWS)
   1. Virtual Machine: 1 $ 1,345.24
   2. OS: Windows
   3. Type: SQL Server
   4. Tier: Standard
   5. License: SQL Enterprise
   6. Instance: 4 cores, 16 GB RAM, 100 GB Storage
   7. Managed Disks: $ 163.84
      1. Tier: Standard HDD
      2. Disk Size: 4096 GB
   8. Storage Transactions: 1000000 units $ 500

1. Personnel



# **[Supplemental Information](#_Table_of_Contents)**

# **[SQL Statements to create tables, define constraints, triggers, procedures and sequences](#_Table_of_Contents)**

CREATE TABLE APPLICANTS

("APPLICANTID" VARCHAR2(10),

"HIGHESTDEGREE" VARCHAR2(50),

"FIRSTNAME" VARCHAR2(50),

"LASTNAME" VARCHAR2(50),

"DOB" DATE,

"AGE" NUMBER (3,0),

PRIMARY KEY ("APPLICANTID")

);

CREATE TABLE APPLICATIONS

("APPLICATIONID" VARCHAR2(10),

"APP\_RESUME" BLOB,

"APPLICANTID" VARCHAR2(50),

PRIMARY KEY ("APPLICATIONID"),

FOREIGN KEY ("APPLICANTID") REFERENCES APPLICANTS ("APPLICANTID") ON DELETE CASCADE

);

CREATE TABLE APPLY\_TO

("APPLICATIONID" VARCHAR2(10),

"JOBPOSTINGID" VARCHAR2(10),

FOREIGN KEY ("APPLICATIONID") REFERENCES APPLICATIONS ("APPLICATIONID") ON DELETE CASCADE,

FOREIGN KEY ("JOBPOSTINGID") REFERENCES JOB\_POSTINGS ("JOBPOSTINGID") ON DELETE CASCADE

);

CREATE TABLE CAMPAIGNS

("CAMPAIGNID" VARCHAR2(10),

"CAMPAIGNNAME" VARCHAR2(50),

PRIMARY KEY ("CAMPAIGNID")

);

CREATE TABLE CATERING\_RESTAURANT

("RESTID" VARCHAR2(10),

"PRICEPERHEAD" NUMBER(5,0) NOT NULL ENABLE,

"MAXPARTYSIZE" NUMBER(5,0),

"MINPARTYSIZE" NUMBER(5,0),

"REQUIREDLEADTIME" NUMBER(5,0),

"DOWNPAYMENTAMT" NUMBER(5,0),

FOREIGN KEY ("RESTID") REFERENCES RESTAURANTS ("RESTID") ON DELETE CASCADE

);

CREATE TABLE CHAINS

("CHAINID" VARCHAR2(10),

"WEBSITE" VARCHAR2(50),

"CHAINNAME" VARCHAR2(50) NOT NULL ENABLE,

PRIMARY KEY ("CHAINID")

);

CREATE SEQUENCE chainID\_seq INCREMENT BY 1 START WITH 101 MAXVALUE 999;

CREATE OR REPLACE TRIGGER triggerChainID

BEFORE INSERT ON CHAINS

FOR EACH ROW

DECLARE

temp\_chainID chains.chainID%type;

BEGIN

SELECT LPAD(to\_char(chainID\_seq.nextval),8,'chain') INTO temp\_chainID

FROM dual;

:new.chainID:= temp\_chainID;

END;

/

CREATE TABLE COMPRISE\_HUMAN\_RESOURCES

("EVENTTEAMID" VARCHAR2(10),

"EMPID" VARCHAR2(10),

FOREIGN KEY ("EVENTTEAMID") REFERENCES EVENT\_TEAMS ("EVENTTEAMID") ON DELETE CASCADE,

FOREIGN KEY ("EMPID") REFERENCES HUMAN\_RESOURCES\_EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE COMPRISE\_MARKETING

("EVENTTEAMID" VARCHAR2(10),

"EMPID" VARCHAR2(10),

FOREIGN KEY ("EVENTTEAMID") REFERENCES EVENT\_TEAMS ("EVENTTEAMID") ON DELETE CASCADE,

FOREIGN KEY ("EMPID") REFERENCES MARKETING\_EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE COMPRISE\_SALES

("EVENTTEAMID" VARCHAR2(10),

"EMPID" VARCHAR2(10),

FOREIGN KEY ("EVENTTEAMID") REFERENCES EVENT\_TEAMS ("EVENTTEAMID") ON DELETE CASCADE,

FOREIGN KEY ("EMPID") REFERENCES SALES\_EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE CORPORATE\_PARTNERS

("CORPPARTID" VARCHAR2(10),

"CORPPARTNAME" VARCHAR2(50) NOT NULL ENABLE,

"SERVICEPROVIDED" VARCHAR2(50) NOT NULL ENABLE,

"EMPID" VARCHAR2(10),

PRIMARY KEY ("CORPPARTID"),

FOREIGN KEY ("EMPID") REFERENCES SALES\_EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE CREATES

("EMPID" VARCHAR2(10),

"CAMPAIGNID" VARCHAR2(10),

FOREIGN KEY ("CAMPAIGNID") REFERENCES CAMPAIGNS ("CAMPAIGNID") ON DELETE CASCADE,

FOREIGN KEY ("EMPID") REFERENCES MARKETING\_EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE CUISINES

("CUISINEID" VARCHAR2(10),

"CUISINETYPE" VARCHAR2(50),

"REGION" VARCHAR2(50),

PRIMARY KEY ("CUISINEID")

);

CREATE TABLE CUSTOMER\_CATEGORIES

("CUST\_CATEGORY" CHAR(5),

"CATEGORY\_NAME" VARCHAR2(20),

"POINTS\_REQUIRED" NUMBER(4,0),

"SERVICE\_DISCOUNT" NUMBER(4,3),

CONSTRAINT "SERVICE\_DISCOUNT\_CHECK" CHECK (Service\_Discount >= 0 and service\_discount < 1.0),

CONSTRAINT "CUST\_CATEGORY\_PK" PRIMARY KEY ("CUST\_CATEGORY")

);

CREATE TABLE CUSTOMER\_SERVICE\_EMPLOYEES

("EMPID" VARCHAR2(10),

"CUSTREVIEWRATING" VARCHAR2(5),

PRIMARY KEY ("EMPID"),

FOREIGN KEY ("EMPID") REFERENCES EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE CUSTOMERS

("CUSTID" VARCHAR2(10),

"FIRSTNAME" VARCHAR2(50) NOT NULL ENABLE,

"LASTNAME" VARCHAR2(50) NOT NULL ENABLE,

"STREET" VARCHAR2(50) NOT NULL ENABLE,

"CITY" VARCHAR2(50) NOT NULL ENABLE,

"ZIP" VARCHAR2(5) NOT NULL ENABLE,

"CUSTSTATE" VARCHAR2(2) NOT NULL ENABLE,

"PHONENUM" NUMBER(10,0) NOT NULL ENABLE,

"CUSTEMAIL" VARCHAR2(100),

PRIMARY KEY ("CUSTID"),

CONSTRAINT "CHK\_CUSTPHONE" CHECK (phoneNum not like '%[^0-9]%'),

CONSTRAINT "CUSTPHONELEN" CHECK (LENGTH(phoneNum) = 10)

);

CREATE SEQUENCE custID\_seq INCREMENT BY 1 START WITH 1001 MAXVALUE 9999;

CREATE OR REPLACE TRIGGER triggerCustID

BEFORE INSERT ON CUSTOMERS

FOR EACH ROW

DECLARE

temp\_custID customers.custID%type;

BEGIN

SELECT LPAD(to\_char(custID\_seq.nextval),8,'cust') INTO temp\_custID

FROM dual;

:new.custID := temp\_custID;

END;

/

CREATE TABLE CUSTOMERS\_FAVORITES

("CUSTID" VARCHAR2(10),

"FAVORITES" VARCHAR2(50),

FOREIGN KEY ("CUSTID") REFERENCES CUSTOMERS ("CUSTID") ON DELETE CASCADE

);

CREATE TABLE CUSTOMERS\_PAYMETHOD

("CUSTID" VARCHAR2(10),

"PAYMETHOD" VARCHAR2(50),

FOREIGN KEY ("CUSTID") REFERENCES CUSTOMERS ("CUSTID") ON DELETE CASCADE

);

CREATE TABLE DELIVERY\_RESTAURANTS

("RESTID" VARCHAR2(10),

"DELCOST" NUMBER(10,2),

"MINORDERPRICE" NUMBER(10,0),

FOREIGN KEY ("RESTID") REFERENCES RESTAURANTS ("RESTID") ON DELETE CASCADE

);

CREATE TABLE DEVELOP

("EMPID" VARCHAR2(10),

"PERKID" VARCHAR2(10),

FOREIGN KEY ("EMPID") REFERENCES MARKETING\_EMPLOYEES ("EMPID") ON DELETE CASCADE,

FOREIGN KEY ("PERKID") REFERENCES PERKS ("PERKID") ON DELETE CASCADE

);

CREATE TABLE DRIVER\_DELIVERIES

("DELIVERYID" VARCHAR2(50),

"CLOCKINDATE" DATE,

"CLOCKINTIME" VARCHAR2(10),

"CLOCKOUTDATE" DATE,

"CLOCKOUTTIME" VARCHAR2(10),

"MILES" NUMBER(10,0),

"NUMDELIVERY" NUMBER(10,0),

"DRIVERID" VARCHAR2(10),

PRIMARY KEY ("DELIVERYID"),

CONSTRAINT "DELDELIVERIES"

FOREIGN KEY ("DRIVERID") REFERENCES DRIVERS ("DRIVERID") ON DELETE CASCADE

);

CREATE TABLE DRIVER\_RATINGS

("RATINGID" VARCHAR2(10),

"NUMSTARS" NUMBER(1,0),

FOREIGN KEY ("RATINGID") REFERENCES RATINGS ("RATINGID") ON DELETE CASCADE

);

CREATE TABLE DRIVERS

("DRIVERID" VARCHAR2(10),

"CITY" VARCHAR2(50),

"DRIVERSTATE" VARCHAR2(2),

"STREET" VARCHAR2(50),

"ZIP" VARCHAR2(5) NOT NULL,

"BANKACCNUM" NUMBER(10,0) NOT NULL,

"DOB" DATE NOT NULL,

"DRIVERREWARDS" VARCHAR2(50),

"EMAIL" VARCHAR2(50),

"FIRSTNAME" VARCHAR2(50) NOT NULL,

"LASTNAME" VARCHAR2(50) NOT NULL,

"SSN" VARCHAR2(11) NOT NULL,

"MAXRANGE" NUMBER(35,0),

"PHONENUM" VARCHAR2(12) NOT NULL,

"INSURANCEPNUM" VARCHAR2(20) NOT NULL,

"LICNUM" VARCHAR2(10) NOT NULL,

CONSTRAINT "CHK\_DRIVEREMAIL" CHECK (email like '%@%.%'),

CONSTRAINT "CHK\_DRIVERPHONE" CHECK (phoneNum not like '%[^0-9]%'),

CONSTRAINT "PHONELEN" CHECK (LENGTH(phoneNum) = 10),

PRIMARY KEY ("DRIVERID"),

UNIQUE ("SSN")

);

CREATE SEQUENCE driverID\_seq INCREMENT BY 1 START WITH 101 MAXVALUE 999;

CREATE OR REPLACE TRIGGER triggerDriverID

BEFORE INSERT ON DRIVERS

FOR EACH ROW

DECLARE

temp\_driverID drivers.driverID%type;

BEGIN

SELECT LPAD(to\_char(driverID\_seq.nextval),4,'D') INTO temp\_driverID

FROM dual;

:new.driverID := temp\_driverID;

END;

/

CREATE TABLE EMPLOYEES

("EMPID" VARCHAR2(10),

"EMPFIRSTNAME" VARCHAR2(50) NOT NULL,

"PHONENUM" NUMBER(10,0) NOT NULL,

"SSN" VARCHAR2(11 BYTE) NOT NULL,

"STREET" VARCHAR2(50),

"CITY" VARCHAR2(50),

"EMPSTATE" VARCHAR2(50),

"ZIP" VARCHAR2(5) NOT NULL,

"EMAIL" VARCHAR2(50) NOT NULL,

"JOINDATE" DATE NOT NULL,

"DOB" DATE NOT NULL,

"AGE" NUMBER(3,0),

"EMPCOMP" NUMBER(10,0) NOT NULL,

"WAGETYPE" VARCHAR2(10),

"TENURE" VARCHAR2(50),

"DEPARTMENTNAME" VARCHAR2(50),

"EMPLASTNAME" VARCHAR2(50),

CONSTRAINT "CHK\_PHONE" CHECK (phoneNum not like '%[^0-9]%'),

CHECK (age >= 18),

PRIMARY KEY ("EMPID"),

UNIQUE ("SSN"),

CONSTRAINT "CHK\_EMPEMAIL" CHECK (email LIKE '%@%.%'),

CONSTRAINT "EMPLEN" CHECK (LENGTH(phoneNum)=10)

);

CREATE TABLE EVENT\_TEAMS

("EVENTTEAMID" VARCHAR2(10),

"EVENTTEAMNAME" VARCHAR2(50) NOT NULL,

"CREATIONDATE" DATE,

PRIMARY KEY ("EVENTTEAMID")

);

CREATE TABLE EVENTS

("EVENTID" VARCHAR2(10),

"STREET" VARCHAR2(50),

"STATE" VARCHAR2(2),

"CITY" VARCHAR2(50),

"ZIP" VARCHAR2(5) NOT NULL,

"NUMATTEND" NUMBER(10,0),

"EVENTTEAMID" VARCHAR2(10),

PRIMARY KEY ("EVENTID"),

FOREIGN KEY ("EVENTTEAMID") REFERENCES EVENT\_TEAMS ("EVENTTEAMID") ON DELETE CASCADE

);

CREATE TABLE EXTERNAL\_CAMPAIGN

("CAMPAIGNID" VARCHAR2(10),

"EC\_TYPE" VARCHAR2(50),

"EC\_MEDIUM" VARCHAR2(50),

PRIMARY KEY ("CAMPAIGNID"),

FOREIGN KEY ("CAMPAIGNID") REFERENCES CAMPAIGNS ("CAMPAIGNID") ON DELETE CASCADE

);

CREATE TABLE FILLED\_JOBS

("JOBID" VARCHAR2(10),

"FILLDATE" DATE NOT NULL,

"EMPID" VARCHAR2(10),

FOREIGN KEY ("JOBID") REFERENCES JOBS ("JOBID") ON DELETE CASCADE,

FOREIGN KEY ("EMPID") REFERENCES EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE FINANCE\_EMPLOYEES

("EMPID" VARCHAR2(10),

"CFACERTIFIED" CHAR(1),

PRIMARY KEY ("EMPID"),

FOREIGN KEY ("EMPID") REFERENCES EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE FOOD\_RATINGS

("RATINGID" VARCHAR2(10),

"NUMFOODSTARS" VARCHAR2(1),

FOREIGN KEY ("RATINGID") REFERENCES RATINGS ("RATINGID") ON DELETE CASCADE

);

CREATE TABLE HUMAN\_RESOURCES\_EMPLOYEES

("EMPID" VARCHAR2(10),

"COMMUNICATION\_SKILLS" VARCHAR2(50),

PRIMARY KEY ("EMPID"),

FOREIGN KEY ("EMPID") REFERENCES EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE INTERNAL\_CAMPAIGN

("CAMPAIGNID" VARCHAR2(10),

"ROOMNUM" NUMBER(10,0),

"OFFICELOC" VARCHAR2(50),

PRIMARY KEY ("CAMPAIGNID"),

FOREIGN KEY ("CAMPAIGNID") REFERENCES CAMPAIGNS ("CAMPAIGNID") ON DELETE CASCADE

);

CREATE TABLE ITEMS

("ITEMID" VARCHAR2(10),

"PRICE" NUMBER(5,2),

"ITEMDESCRIPTION" VARCHAR2(500),

"DIETARYRESTRICTIONS" VARCHAR2(50),

"RESTID" VARCHAR2(10),

"CHAINID" VARCHAR2(10),

"ITEMNAME" VARCHAR2(100),

PRIMARY KEY ("ITEMID"),

CONSTRAINT "ITEMSRESTFK",

FOREIGN KEY ("RESTID") REFERENCES RESTAURANTS ("RESTID") ON DELETE CASCADE,

CONSTRAINT "CHAINPERKFK"

FOREIGN KEY ("CHAINID") REFERENCES CHAINS ("CHAINID") ON DELETE CASCADE

);

CREATE TABLE JOB\_POSTINGS

("JOBPOSTINGID" VARCHAR2(10),

"JOBPOSTDATE" DATE,

"JOBDESCRIPTION" VARCHAR2(50) NOT NULL,

"EMPID" VARCHAR2(10),

PRIMARY KEY ("JOBPOSTINGID"),

FOREIGN KEY ("EMPID") REFERENCES HUMAN\_RESOURCES\_EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE JOBS

("JOBID" VARCHAR2(10),

"JOBPOSITION" VARCHAR2(50) NOT NULL,

"JOBSTATUS" VARCHAR2(50),

"JOBPOSTINGID" VARCHAR2(10),

PRIMARY KEY ("JOBID"),

FOREIGN KEY ("JOBPOSTINGID") REFERENCES JOB\_POSTINGS ("JOBPOSTINGID") ON DELETE CASCADE

);

CREATE TABLE LEADS

("LEADID" VARCHAR2(10),

"LEADNAME" VARCHAR2(50) NOT NULL,

"LEADCONTACTFNAME" VARCHAR2(50) NOT NULL,

"LEADCONTACTLNAME" VARCHAR2(50) NOT NULL,

"LEADCONTACTTITLE" VARCHAR2(50),

"LEADCONTACTPHONE" NUMBER(10,0) NOT NULL,

"EVENTID" VARCHAR2(10),

"EMPID" VARCHAR2(10),

CONSTRAINT "CHK\_LEADPHONE" CHECK (leadContactPhone not like '%[^0-9]%'),

CONSTRAINT "LEADPHONELEN" CHECK (LENGTH(leadContactPhone) = 10),

PRIMARY KEY ("LEADID"),

FOREIGN KEY ("EVENTID") REFERENCES EVENTS ("EVENTID")ON DELETE CASCADE,

FOREIGN KEY ("EMPID") REFERENCES SALES\_EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE MARKETING\_EMPLOYEES

("EMPID" VARCHAR2(10),

"YEARSEXP" NUMBER(2,0),

PRIMARY KEY ("EMPID"),

FOREIGN KEY ("EMPID") REFERENCES EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE OFFERS

("PERKID" VARCHAR2(10),

"RESTID" VARCHAR2(10),

CONSTRAINT "OFFERSRESTFK"

FOREIGN KEY ("RESTID") REFERENCES RESTAURANTS ("RESTID") ON DELETE CASCADE,

CONSTRAINT "OFFERSPERKFK"

FOREIGN KEY ("PERKID") REFERENCES PERKS ("PERKID") ON DELETE CASCADE

);

CREATE TABLE OPEN\_JOBS

("JOBID" VARCHAR2(10),

"CLOSEDATE" DATE NOT NULL,

"JOBPOSTINGID" VARCHAR2(10),

FOREIGN KEY ("JOBID") REFERENCES JOBS ("JOBID") ON DELETE CASCADE,

FOREIGN KEY ("JOBPOSTINGID") REFERENCES JOB\_POSTINGS ("JOBPOSTINGID") ON DELETE CASCADE

);

CREATE TABLE ORDERS

("ORDERID" VARCHAR2(10),

"SPECINSTRUCT" VARCHAR2(50),

"STATUS" VARCHAR2(50),

"ORDERDATE" DATE,

"TOTALCOST" NUMBER(5,2),

"RESTID" VARCHAR2(10),

"PERKID" VARCHAR2(10),

"CUSTID" VARCHAR2(10),

"DELIVERYID" VARCHAR2(10),

PRIMARY KEY ("ORDERID"),

CONSTRAINT "DELDELIVERYORDERS"

FOREIGN KEY ("DELIVERYID") REFERENCES DRIVER\_DELIVERIES ("DELIVERYID") ON DELETE CASCADE,

CONSTRAINT "RESTORDERSFK"

FOREIGN KEY ("RESTID") REFERENCES RESTAURANTS ("RESTID") ON DELETE CASCADE,

CONSTRAINT "PERKORDERSFK"

FOREIGN KEY ("PERKID") REFERENCES PERKS ("PERKID") ON DELETE CASCADE,

CONSTRAINT "CUSTOMERSORDERSFK"

FOREIGN KEY ("CUSTID") REFERENCES CUSTOMERS ("CUSTID") ON DELETE CASCADE

);

CREATE OR REPLACE TRIGGER ORDER\_SEQUENCE\_TRIGGER

BEFORE INSERT OR UPDATE ON orders

FOR EACH ROW

BEGIN

:new.orderid := 'order'||to\_char(order\_id\_sequence.nextval,'FM0000');

END

;

/

CREATE OR REPLACE TRIGGER PAYMENTS\_TRIGGER

AFTER INSERT ON ORDERS

FOR EACH ROW

DECLARE

order\_status orders.status%type;

pid payments.paymentid%type;

tcost orders.totalcost%type;

v\_order\_id orders.orderid%type;

delid orders.deliveryid%type;

order\_date orders.orderdate%type;

promoAmt perks.promoamount%type;

startdate perks.promostartdate%type;

enddate perks.promoenddate%type;

v\_perkid perks.perkid%type;

newtcost orders.totalcost%type;

BEGIN

SELECT :new.status, :new.totalcost, :new.orderid, :new.deliveryid INTO order\_status, tcost, v\_order\_id, delid FROM dual;

v\_order\_id := :new.orderid;

v\_perkid := :new.perkid;

order\_date := :new.orderdate;

IF (v\_perkid IS NOT NULL) THEN

SELECT promoamount, promostartdate, promoenddate INTO promoAmt, startdate, enddate FROM perks WHERE perkid = v\_perkid;

END IF;

dbms\_output.put\_line (promoAmt || startdate || enddate);

IF (order\_status IN ('Completed', 'In Progress') AND (delid IS NOT NULL)

AND v\_perkid IS NOT NULL) THEN

IF(order\_date BETWEEN startdate AND enddate) THEN

dbms\_output.put\_line (promoAmt || startdate || enddate);

tcost := tcost - promoAmt;

INSERT INTO payments VALUES(('payment' || to\_char(payment\_id\_sequence.nextval, 'FM000')),

tcost, (0.10\*tcost), (0.15\*tcost), (0.75\*tcost), v\_order\_id);

ELSE

INSERT INTO payments VALUES(('payment' || to\_char(payment\_id\_sequence.nextval, 'FM000')),

tcost, (0.10\*tcost), (0.15\*tcost), (0.75\*tcost), v\_order\_id);

END IF;

ELSIF ((order\_status IN ('Completed', 'In Progress') AND delid IS NOT NULL)

AND (v\_perkid IS NULL)) THEN

INSERT INTO payments VALUES(('payment' || to\_char(payment\_id\_sequence.nextval, 'FM000')),

tcost, (0.10\*tcost), (0.15\*tcost), (0.75\*tcost), v\_order\_id);

ELSIF (order\_status IN ('Completed', 'In Progress') AND delid IS NULL AND

v\_perkid IS NOT NULL AND order\_date BETWEEN startdate AND enddate) THEN

dbms\_output.put\_line (promoAmt || startdate || enddate);

tcost := tcost - promoAmt;

INSERT INTO payments VALUES(('payment' || to\_char(payment\_id\_sequence.nextval, 'FM000')),

tcost, (0.10\*tcost), 0, (0.90\*tcost), v\_order\_id);

ELSIF (order\_status IN ('Completed', 'In Progress') AND delid IS NULL) THEN

INSERT INTO payments VALUES(('payment' || to\_char(payment\_id\_sequence.nextval, 'FM000')),

tcost, (0.10\*tcost), 0, (0.90\*tcost), v\_order\_id);

END IF;

END;

/

CREATE TABLE ORDERS\_ITEMS

("ORDERID" VARCHAR2(10),

"ITEMS" VARCHAR2(50),

CONSTRAINT "OIRESTFK"

FOREIGN KEY ("ORDERID")REFERENCES ORDERS ("ORDERID") ON DELETE CASCADE

);

CREATE TABLE PARTNER\_CONTRACTS

("CONTRACTID" VARCHAR2(10),

"CONTRACTEXPDATE" DATE NOT NULL,

"CONTRACTSTARTDATE" DATE NOT NULL,

"STATUS" VARCHAR2(20),

"CORPPARTID" VARCHAR2(10),

PRIMARY KEY ("CONTRACTID"),

FOREIGN KEY ("CORPPARTID") REFERENCES CORPORATE\_PARTNERS ("CORPPARTID") ON DELETE CASCADE

);

CREATE TABLE PAYMENTS

("PAYMENTID" VARCHAR2(10),

"CUSTPAIDAMT" NUMBER(10,2),

"PROFITEARNED" NUMBER(10,2),

"DRIVERRCVDAMT" NUMBER(10,2),

"RESTRCVDAMT" NUMBER(10,2),

"ORDERID" VARCHAR2(10),

PRIMARY KEY ("PAYMENTID"),

CONSTRAINT "PAYMENTORDERFK"

FOREIGN KEY ("ORDERID") REFERENCES ORDERS ("ORDERID") ON DELETE CASCADE

);

CREATE TABLE PERKS

("PERKID" VARCHAR2(10),

"PROMOAMOUNT" NUMBER(2,0),

"PROMOCODE" VARCHAR2(10),

"PERKNAME" VARCHAR2(50),

"PROMOENDDATE" DATE,

"PROMOSTARTDATE" DATE,

PRIMARY KEY ("PERKID")

);

CREATE TABLE PICKUP\_RESTAURANT

("RESTID" VARCHAR2(10),

"PICKUPTIME" VARCHAR2(10),

CONSTRAINT "RESTPICKUPFK"

FOREIGN KEY ("RESTID") REFERENCES RESTAURANTS ("RESTID") ON DELETE CASCADE

);

CREATE TABLE PROBLEM\_TICKET

("TICKETID" VARCHAR2(10),

"STATUS" VARCHAR2(15),

"DATERESOLVED" DATE NOT NULL,

"DATECREATED" DATE NOT NULL,

"PROBID" VARCHAR2(10),

"CUSTID" VARCHAR2(10),

"EMPID" VARCHAR2(10),

PRIMARY KEY ("TICKETID"),

CONSTRAINT "PROBTICKETCUSTFK"

FOREIGN KEY ("CUSTID") REFERENCES CUSTOMERS ("CUSTID") ON DELETE CASCADE,

CONSTRAINT "PROBTICKETPROBTFK"

FOREIGN KEY ("PROBID") REFERENCES PROBLEM\_TYPES ("PROBID") ON DELETE CASCADE,

CONSTRAINT "CUSTSERVICEFK"

FOREIGN KEY ("EMPID") REFERENCES CUSTOMER\_SERVICE\_EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE PROBLEM\_TYPES

("PROBID" VARCHAR2(10),

"PROBRESOLUTION" VARCHAR2(255),

"PROBDESCRIPTION" VARCHAR2(50) NOT NULL,

PRIMARY KEY ("PROBID")

);

CREATE TABLE PROBLEMS\_CATEGORY

("PROBID" VARCHAR2(10),

"PROBCATEGORY" VARCHAR2(50),

FOREIGN KEY ("PROBID") REFERENCES PROBLEM\_TYPES ("PROBID") ON DELETE CASCADE

);

CREATE TABLE PROMOTE

("EVENTID" VARCHAR2(10),

"CAMPAIGNID" VARCHAR2(10),

FOREIGN KEY ("EVENTID") REFERENCES EVENTS ("EVENTID") ON DELETE CASCADE,

FOREIGN KEY ("CAMPAIGNID") REFERENCES CAMPAIGNS ("CAMPAIGNID") ON DELETE CASCADE

);

CREATE TABLE RATINGS

("RATINGID" VARCHAR2(10),

"CUSTCOMMENTS" VARCHAR2(255),

"ORDERID" VARCHAR2(10),

PRIMARY KEY ("RATINGID"),

CONSTRAINT "RATINGSORDERFK"

FOREIGN KEY ("ORDERID") REFERENCES ORDERS ("ORDERID") ON DELETE CASCADE

);

CREATE TABLE RESTAURANT\_ACCOUNTS

("ACCTID" VARCHAR2(10),

"ACC\_USERNAME" VARCHAR2(20) NOT NULL,

"ACC\_PASSWORD" VARCHAR2(20) NOT NULL,

"BANKACCTNUM" VARCHAR2(10),

"EMPID" VARCHAR2(10),

"RESTID" VARCHAR2(10),

PRIMARY KEY ("ACCTID"),

FOREIGN KEY ("EMPID") REFERENCES FINANCE\_EMPLOYEES ("EMPID") ON DELETE CASCADE,

CONSTRAINT "RESTACCRESTFK"

FOREIGN KEY ("RESTID")REFERENCES RESTAURANTS ("RESTID") ON DELETE CASCADE

);

CREATE TABLE RESTAURANT\_HOURS

("RESTID" VARCHAR2(10),

"WEEKDAY" VARCHAR2(10),

"OPENTIME" VARCHAR2(10),

"CLOSETIME" VARCHAR2(10),

CONSTRAINT "RESTHRSRESTFK"

FOREIGN KEY ("RESTID") REFERENCES RESTAURANTS ("RESTID") ON DELETE CASCADE

);

CREATE TABLE RESTAURANTS

("RESTID" VARCHAR2(10),

"RESTNAME" VARCHAR2(50) NOT NULL,

"REST\_CATEGORY" VARCHAR2(50),

"PHONENUM" NUMBER(12,0),

"STREET" VARCHAR2(50),

"CITY" VARCHAR2(50),

"RESTSTATE" VARCHAR2(2),

"ZIP" VARCHAR2(5) NOT NULL,

"CHAINID" VARCHAR2(10),

CONSTRAINT "CHK\_RESTPHONE" CHECK (phoneNum not like '%[^0-9]%'),

CONSTRAINT "RESTPHONELEN" CHECK (LENGTH(phoneNum) = 10),

PRIMARY KEY ("RESTID"),

FOREIGN KEY ("CHAINID") REFERENCES CHAINS ("CHAINID") ON DELETE CASCADE

);

CREATE TABLE REVIEW\_BY

("APPLICATIONID" VARCHAR2(10),

"EMPID" VARCHAR2(10),

FOREIGN KEY ("APPLICATIONID") REFERENCES APPLICATIONS ("APPLICATIONID") ON DELETE CASCADE,

FOREIGN KEY ("EMPID") REFERENCES HUMAN\_RESOURCES\_EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE SALES\_EMPLOYEES

("EMPID" VARCHAR2(10),

"SALESTARGET" NUMBER (10,0),

PRIMARY KEY ("EMPID"),

FOREIGN KEY ("EMPID") REFERENCES EMPLOYEES ("EMPID") ON DELETE CASCADE

);

CREATE TABLE SCHEDULE

("SCHEDULEID" VARCHAR2(10),

"SCHEDULE\_DATE" DATE,

"STARTTIME" VARCHAR2(10),

"ENDTIME" VARCHAR2(10),

"DRIVERID" VARCHAR2(10),

PRIMARY KEY ("SCHEDULEID"),

CONSTRAINT "DELSCHEDULES"

FOREIGN KEY ("DRIVERID") REFERENCES DRIVERS ("DRIVERID") ON DELETE CASCADE

);

CREATE TABLE SERVE

("CUISINEID" VARCHAR2(10),

"RESTID" VARCHAR2(10),

FOREIGN KEY ("CUISINEID") REFERENCES CUISINES ("CUISINEID") ON DELETE CASCADE,

CONSTRAINT "SERVERESTFK"

FOREIGN KEY ("RESTID") REFERENCES RESTAURANTS ("RESTID") ON DELETE CASCADE

);

CREATE TABLE SOFTWARE\_DEV\_EMPLOYEES

("EMPID" VARCHAR2(10),

"SPECIALITY" VARCHAR2(20),

FOREIGN KEY ("EMPID") REFERENCES EMPLOYEES ("EMPID") ON DELETE CASCADE

);

# **[Works Citied](#_Table_of_Contents)**

All images used in the website and presentation are from the following links:

<https://www.google.com/search?rlz=1C1CHBD_enUS876US876&tbm=isch&sxsrf=ACYBGNT2r57eVGNDBrbTrHZX3GWlhY9Yeg:1575412717134&q=grubhub+white+logo&chips=q:seamless+white,g_1:background:JMSSU0pAi-Y%3D,online_chips:grub,online_chips:transparent&usg=AI4_-kRgFelciOJDhJSnz7PWISR6fUVH-A&sa=X&ved=0ahUKEwiclITbxZrmAhVgCTQIHSoOBBEQ4lYILigA&biw=1242&bih=553&dpr=1.1#imgrc=gWFcFDWMThKn6M:>

<https://www.google.com/search?rlz=1C1CHBD_enUS876US876&tbm=isch&sxsrf=ACYBGNT2r57eVGNDBrbTrHZX3GWlhY9Yeg:1575412717134&q=grubhub+white+logo&chips=q:seamless+white,g_1:background:JMSSU0pAi-Y%3D,online_chips:grub,online_chips:transparent&usg=AI4_-kRgFelciOJDhJSnz7PWISR6fUVH-A&sa=X&ved=0ahUKEwiclITbxZrmAhVgCTQIHSoOBBEQ4lYILigA&biw=1242&bih=553&dpr=1.1#imgrc=pVA3FP0e64KF2M:>