

Pizza Shop Analysis: A Report

Jason Ward

BUS601

Table of Contents

| | |
|--|-----------|
| Initial Report..... | 1 |
| Step One..... | 3 |
| - Oracle Installation and Database Creation..... | 3 |
| - Create User ID..... | 4 |
| - Grant Required Privileges and Roles..... | 4 |
| Step Two..... | 5 |
| - Collect Business Rules..... | 5 |
| - Develop an ER Diagram – Logical View..... | 6 |
| Step Three..... | 7 |
| - Create Tables..... | 7 |
| - Insert Data into Our Tables..... | 10 |
| Step Four..... | 14 |
| - Develop SQL Queries..... | 14 |
| Step Five..... | 17 |
| - Develop Queries for Analysis..... | 17 |
| Final Report..... | 27 |

Initial Report

Just Pizza! is a local pizza establishment with its roots embedded in the east side of the Greater Cleveland area. It is a staple restaurant in its respective area for all the population that resides there. It has proudly been in operation since its inception in the early years of 1944, where the current owners father started it as a family business. While time will allow for brand loyalty to develop, it can also spell disaster from a multitude of sources.

Just Pizza! has seemed to beat the odds for decades, even battling through a significant recession. Lately though, it has seen a significant decrease in orders and people walking through the door. The owner has noticed some of his most loyal customers disappear with what seems like no reason. This has put the restaurant and its staff into a state of panic trying to discover what are the root causes. This is where our agency takes over to do a comprehensive review and offer consultation on how to possibly improve upon any complications the restaurant is facing.

Before any analysis is completed, the agency decided to review multiple avenues of the restaurant to try to focus in on specific issues. First, a survey was mailed out to past customers to get their input; the results were telling. Many selected options that showed the pizza shop has increased in price significantly over the years. As well, quality has seemed to drop in more recent times. A large amount also said there were never any offers or deals available, seemingly a result of the old school pizza shop mentality. Although, many participants seemed willing to pay the higher prices to support their favorite pizza shop if decent deals were given out semi-often. After speaking to the owner, he described how the shop had no choice but to increase prices due to rising supplier pricing.

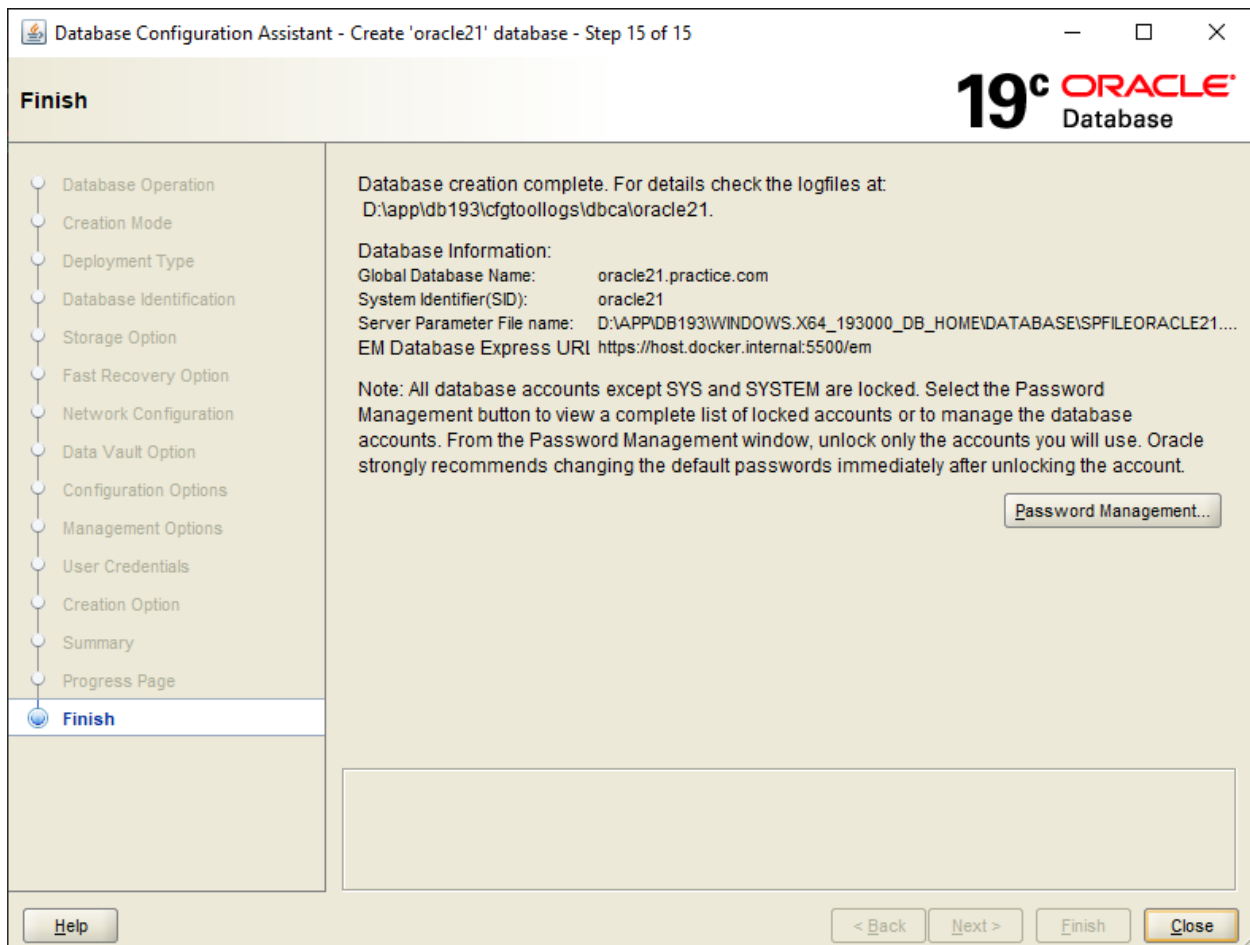
Secondly, our agency decided to go forwards in having discussions with the staff on board. Many hesitantly discussed how they believed the prices were a tad high; even outweighing any extra cost from supplier expenses. We also discovered that many of the staff are unsatisfied with their role and pay. Possibly examining some more information around the staff to a closer degree may help us achieve a positive outcome with our goal at hand. This seems to possibly be an answer as to why quality has diminished in recent times. This leaves us with enough information to start analyzing Just Pizza! and see if there is a way to increase sales.

The whole process our agency will be going through is comprehensive. Our report will show every detail of analysis, from the database creation to the final report. We will use the tables within the database to derive any possible insights to help shore up any issues. This involves querying Relational Databases with the language SQL. With full cooperation of all parties involved and excellent effort from our agency, we hope to help Just Pizza! remain a staple for everyone in the local area.

Step One

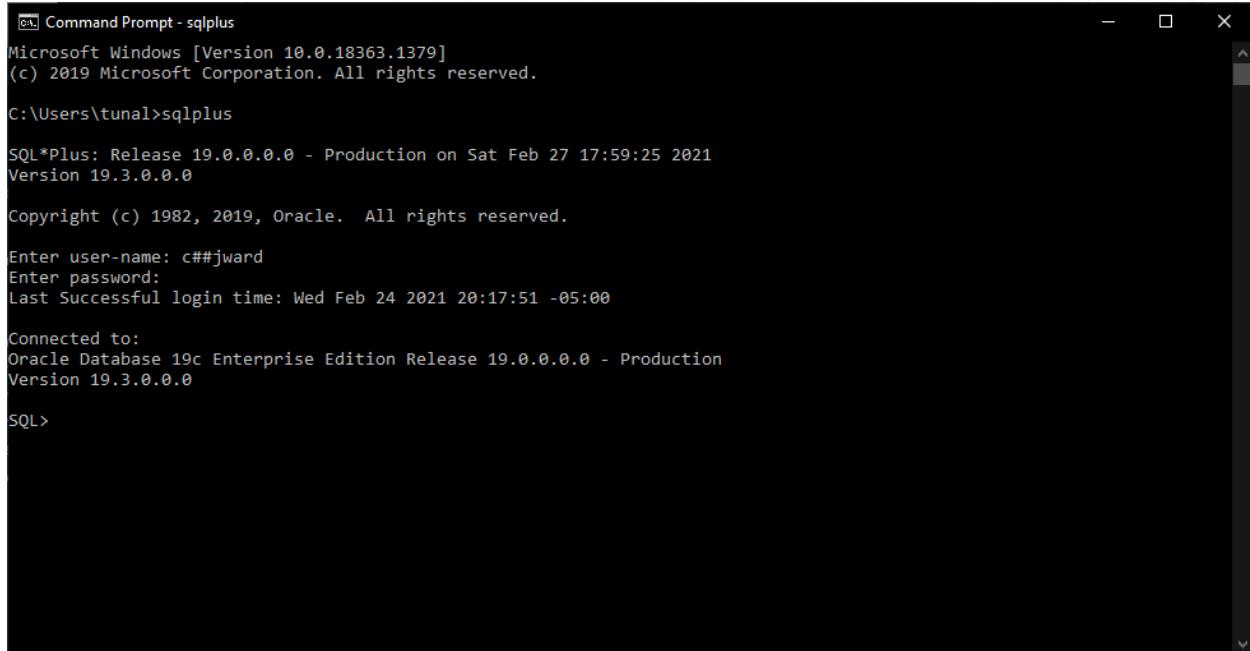
Oracle Installation and Database Creation

The first step in our process is to install our RDBMS which is “Oracle.” Afterwards, we take some time to create our database so we can get prepared for the next step.



Create User ID

Here we are creating our user ID and password so we can have access to our own personal account. To keep it simple, we make our username to only include our first initial concatenated with our last name. Sometimes, it requires c## before the other info to register correctly.



```
Command Prompt - sqlplus
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\tunal>sqlplus

SQL*Plus: Release 19.0.0.0.0 - Production on Sat Feb 27 17:59:25 2021
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle. All rights reserved.

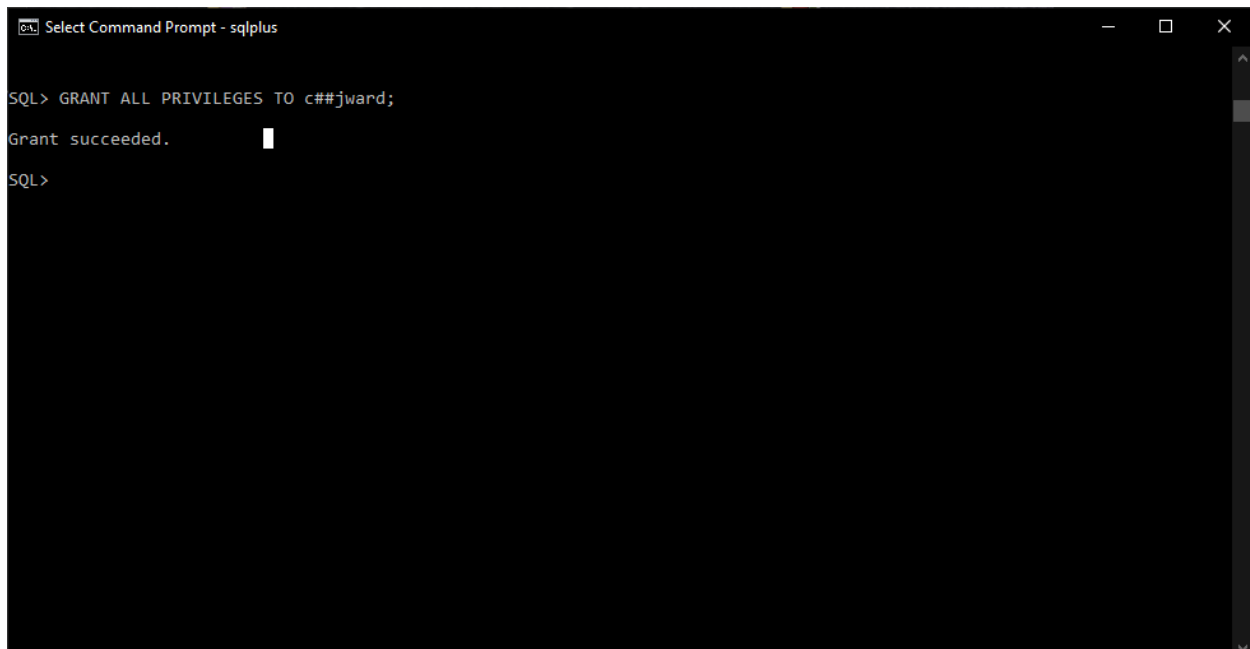
Enter user-name: c##jward
Enter password:
Last Successful login time: Wed Feb 24 2021 20:17:51 -05:00

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0

SQL>
```

Grant Required Privileges and Roles

Under the general access sys account, we connect and grant our newly made account the appropriate roles and privileges to manage our database.



```
Select Command Prompt - sqlplus

SQL> GRANT ALL PRIVILEGES TO c##jward;

Grant succeeded.

SQL>
```

Step Two

Collect Business Rules

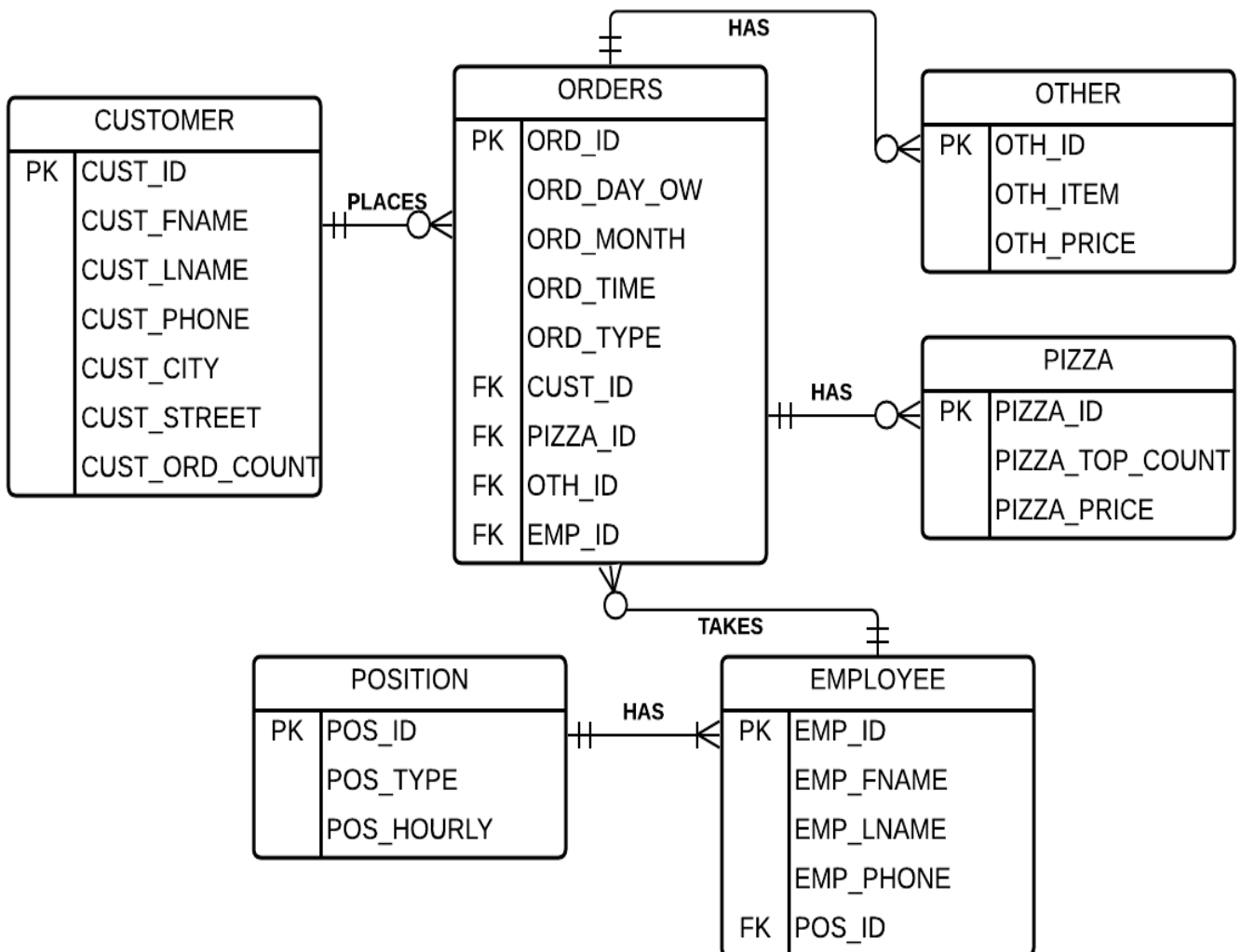
Next, we create our and distinguish or business rules to help us get a better understanding of how the data will work together. This is a crucial step for things to proceed appropriately.

Description of operations

- A customer can place many orders over the phone.
- A single order can only be placed by one customer.
- A customer does not need an order to exist.
- An order requires a customer.
- A single order can only be taken by one employee.
- One employee can take many orders.
- An employee does not need an order to exist.
- An order requires an employee.
- Every employee can only have one position.
- A single position can have many employees.
- An employee must have a position assigned.
- A position needs to be assigned to an employee.
- An order can have many “other” food items added.
- One “other” food item can only have one order.
- An order does not require an “other” food item.
- An “other” food item requires an order.
- An order can have many pizzas added.
- A single pizza can only have one order.
- An order does not require a pizza on it.
- A pizza requires an order.

Develop an ER Diagram – Logical View

This next step's creation is called an "Entity Relationship Diagram." This another crucial piece in the general concept of the creation of the database. This helps plan out every detail of every table within the database including their keys, relationships, and constraints. This is the ERD that has been created in the way that was best envisioned for an efficient database.



Step Three

Create Tables

This is the beginning of the actual creation of the database through code. We start this process by creating the actual tables individually. Within each SQL query, we set them up with multiple components entailing things such as creating the relationships and developing the constraints. Having the table created be as accurate as possible is crucial for precise analysis in the later stages. Below, the six tables created are titled “OTHER”, “PIZZA”, “POSITION”, “EMPLOYEE”, “CUSTOMER”, and “ORDERS.”

```
SQL> CREATE TABLE OTHER (  
  2  OTH_ID INTEGER NOT NULL,  
  3  OTH_ITEM VARCHAR(10),  
  4  OTH_PRICE VARCHAR(255),  
  5  PRIMARY KEY (OTH_ID));
```

Table created.

```
SQL> CREATE TABLE PIZZA (  
  2  PIZZA_ID INTEGER NOT NULL,  
  3  PIZZA_TOP_COUNT VARCHAR(10),  
  4  PIZZA_PRICE VARCHAR(255),  
  5  PRIMARY KEY (PIZZA_ID));
```

Table created.

```
SQL> CREATE TABLE POSITION (  
  2  POS_ID INTEGER NOT NULL,  
  3  POS_TYPE VARCHAR(20) NOT NULL,  
  4  POS_HOURLY VARCHAR(20) NOT NULL,  
  5  PRIMARY KEY (POS_ID));
```

Table created.

```
SQL> CREATE TABLE EMPLOYEE (  
  2  EMP_ID INTEGER NOT NULL,  
  3  EMP_FNAME VARCHAR(20) NOT NULL,  
  4  EMP_LNAME VARCHAR(20) NOT NULL,  
  5  EMP_PHONE VARCHAR(13) NOT NULL,  
  6  POS_ID INTEGER NOT NULL,  
  7  PRIMARY KEY (EMP_ID),  
  8  FOREIGN KEY (POS_ID) REFERENCES POSITION);
```

Table created.

```
SQL> CREATE TABLE CUSTOMER (  
  2  CUST_ID INTEGER NOT NULL,  
  3  CUST_FNAME VARCHAR(20) NOT NULL,  
  4  CUST_LNAME VARCHAR(20),  
  5  CUST_PHONE VARCHAR(13) NOT NULL,  
  6  CUST_CITY VARCHAR(15),  
  7  CUST_STREET VARCHAR(30),  
  8  CUST_ORD_COUNT INTEGER,  
  9  PRIMARY KEY (CUST_ID));
```

Table created.

```
SQL> CREATE TABLE ORDERS (  
  2  ORD_ID INTEGER NOT NULL,  
  3  ORD_DAY_OW VARCHAR(9) NOT NULL,  
  4  ORD_MONTH VARCHAR(9) NOT NULL,  
  5  ORD_TIME VARCHAR(5) NOT NULL,  
  6  ORD_TYPE VARCHAR(8) NOT NULL,  
  7  CUST_ID INTEGER NOT NULL,  
  8  PIZZA_ID INTEGER NOT NULL,  
  9  OTH_ID INTEGER NOT NULL,  
10  EMP_ID INTEGER NOT NULL,  
11  PRIMARY KEY (ORD_ID),  
12  FOREIGN KEY (CUST_ID) REFERENCES CUSTOMER,  
13  FOREIGN KEY (PIZZA_ID) REFERENCES PIZZA,  
14  FOREIGN KEY (OTH_ID) REFERENCES OTHER,  
15  FOREIGN KEY (EMP_ID) REFERENCES EMPLOYEE);
```

Table created.

Insert Data into Our Tables

This step is the most important of our whole setup of the database itself, data insertion! Through queries, we manually insert all the data we have available in the order the was determined by the business rules, ERD, and constraints. With most databases, there are tables with gigabytes of information. Within our tables, we have about twenty entries each, which is significantly smaller. This is just a sample size of the past year's data from Just Pizza!, but it should be enough to help us derive insights and deliver a proper analysis.

IN

```
SQL> SELECT * FROM PIZZA;
```

| PIZZA_ID | PIZZA_TOP_ | PIZZA_PRICE |
|----------|------------|-------------|
| 100 | 5 | 26 |
| 419 | 3 | 20 |
| 14 | 1 | 10 |
| 235 | 1 | 10 |
| 106 | 1 | 10 |
| 545 | 3 | 20 |
| 6 | 2 | 16 |
| 38 | 2 | 16 |
| 210 | 3 | 20 |
| 155 | 1 | 10 |
| 311 | 3 | 20 |

| PIZZA_ID | PIZZA_TOP_ | PIZZA_PRICE |
|----------|------------|-------------|
| 76 | 5 | 26 |
| 95 | 4 | 23 |
| 188 | 1 | 26 |
| 421 | 4 | 23 |
| 10 | 1 | 10 |
| 236 | | |
| 69 | | |
| 192 | | |
| 364 | | |

```
20 rows selected.
```

```
SQL> SELECT * FROM OTHER;
```

| OTH_ID | OTH_ITEM | OTH_PRICE |
|--------|----------|-----------|
| 106 | WINGS | 8 |
| 364 | BREAD | 5 |
| 235 | WINGS | 8 |
| 210 | WINGS | 8 |
| 545 | BREAD | 5 |
| 155 | BREAD | 5 |
| 14 | COOKIE | 6 |
| 100 | COOKIE | 6 |
| 6 | BREAD | 5 |
| 76 | BREAD | 5 |
| 421 | | |
| 69 | BREAD | 5 |
| 311 | | |
| 236 | WINGS | 8 |
| 95 | | |
| 419 | | |
| 38 | | |
| 188 | | |
| 10 | | |
| 192 | COOKIE | 6 |

```
20 rows selected.
```

```
SQL> SELECT * FROM POSITION;
```

| POS_ID | POS_TYPE | POS_HOURLY |
|--------|-----------------|------------|
| 1 | GENERAL MANAGER | 25 |
| 2 | MANAGER | 16 |
| 3 | MANAGER | 14 |
| 4 | MANAGER | 14 |
| 5 | INSIDER | 11 |
| 6 | INSIDER | 10 |
| 7 | INSIDER | 10 |
| 8 | INSIDER | 10 |
| 9 | DRIVER | 5 |
| 10 | DRIVER | 5 |
| 11 | DRIVER | 5 |
| 12 | DRIVER | 5 |
| 13 | DRIVER | 5 |
| 14 | DRIVER | 5 |
| 15 | DISHWASHER | 6 |

```
15 rows selected.
```

```
SQL> SELECT * FROM EMPLOYEE;
```

| EMP_ID | EMP_FNAME | EMP_LNAME | EMP_PHONE | POS_ID |
|--------|-----------|-------------|--------------|--------|
| 1 | SAFWAN | WHITTINGTON | 440-665-4895 | 1 |
| 2 | JULIA | MATTHEWS | 216-326-5489 | 2 |
| 3 | AMAN | BERRY | 440-512-3165 | 3 |
| 4 | MAIZIE | TIMMS | 440-598-1532 | 4 |
| 5 | JACK | LENNON | 216-321-5487 | 5 |
| 6 | NEV | BEIL | 440-656-8221 | 6 |
| 7 | GEORGE | HUTCHINSON | 440-542-1598 | 7 |
| 8 | MACAULAY | GREENAWAY | 440-548-3256 | 8 |
| 9 | MIGUEL | GIBSON | 440-568-4526 | 9 |
| 10 | ARCHIBALD | STARK | 330-654-5468 | 10 |
| 11 | MYLA | GREAVES | 216-512-4865 | 11 |
| 12 | JENSON | MURRAY | 440-516-5689 | 12 |
| 13 | JANNAT | PHAN | 440-548-9623 | 13 |
| 14 | GISELLE | JOHNS | 216-696-3216 | 14 |
| 15 | DONNIE | BUCHANAN | 440-598-4123 | 15 |

```
15 rows selected.
```

```
SQL> SELECT * FROM ORDERS;
```

| ORD_ID | ORD_DAY_O | ORD_MONTH | ORD_T | ORD_TYPE | CUST_ID | PIZZA_ID | OTH_ID | EMP_ID |
|--------|-----------|-----------|-------|----------|---------|----------|--------|--------|
| 1 | FRIDAY | JANUARY | 10:00 | DELIVERY | 1 | 545 | 545 | 5 |
| 2 | SATURDAY | JUNE | 17:00 | DELIVERY | 2 | 192 | 192 | 7 |
| 3 | MONDAY | MAY | 18:00 | DELIVERY | 3 | 76 | 76 | 6 |
| 4 | THURSDAY | FEBRUARY | 15:00 | DELIVERY | 4 | 14 | 14 | 5 |
| 5 | FRIDAY | APRIL | 11:00 | DELIVERY | 5 | 236 | 236 | 2 |
| 6 | SATURDAY | MAY | 17:00 | DELIVERY | 6 | 106 | 106 | 1 |
| 7 | SUNDAY | JANUARY | 19:00 | DELIVERY | 7 | 210 | 210 | 5 |
| 8 | WEDNESDAY | OCTOBER | 18:00 | DELIVERY | 8 | 6 | 6 | 7 |
| 9 | SATURDAY | NOVEMBER | 10:00 | DELIVERY | 9 | 421 | 421 | 15 |
| 10 | WEDNESDAY | OCTOBER | 17:00 | DELIVERY | 10 | 311 | 311 | 9 |
| 11 | SUNDAY | SEPTEMBER | 19:00 | DELIVERY | 11 | 69 | 69 | 6 |
| | | | | | | | | |
| ORD_ID | ORD_DAY_O | ORD_MONTH | ORD_T | ORD_TYPE | CUST_ID | PIZZA_ID | OTH_ID | EMP_ID |
| 12 | THURSDAY | MARCH | 18:00 | DELIVERY | 12 | 10 | 10 | 3 |
| 13 | FRIDAY | JUNE | 12:00 | DELIVERY | 13 | 188 | 188 | 14 |
| 14 | FRIDAY | DECEMBER | 18:00 | PICK UP | 14 | 235 | 235 | 2 |
| 15 | FRIDAY | FEBRUARY | 12:00 | PICK UP | 15 | 155 | 155 | 12 |
| 16 | WEDNESDAY | DECEMBER | 16:00 | PICK UP | 16 | 38 | 38 | 4 |
| 17 | SATURDAY | NOVEMBER | 11:00 | PICK UP | 17 | 419 | 419 | 14 |
| 18 | SATURDAY | JULY | 20:00 | PICK UP | 18 | 100 | 100 | 4 |
| 19 | FRIDAY | DECEMBER | 12:00 | PICK UP | 19 | 95 | 95 | 8 |
| 20 | TUESDAY | NOVEMBER | 14:00 | PICK UP | 20 | 364 | 364 | 8 |

20 rows selected.

```
SQL> SELECT * FROM CUSTOMER;
```

| CUST_ID | CUST_FNAME | CUST_LNAME | CUST_PHONE | CUST_CITY | CUST_STREET | CUST_ORD_COUNT |
|---------|------------|------------|--------------|------------|-----------------|----------------|
| 1 | ZAINA | ALDRED | 440-654-3216 | MENTOR | MATTHEWS STREET | 2 |
| 2 | DARRELL | BENNETT | 216-654-6548 | WILLOUGHBY | EXMOUTH LAWN | 10 |
| 3 | FERN | TURNER | 440-562-2135 | MENTOR | HERNE STREET | |
| 4 | NOEL | HOPPER | 440-986-5123 | WILLOUGHBY | SEAFORD DRIVE | 4 |
| 5 | SOFIE | WELLS | 440-541-3216 | EASTLAKE | ORGAN WAY | 4 |
| 6 | HARVIR | STONE | 330-320-1562 | WILLOUGHBY | WARWICK ROAD | 2 |
| 7 | OLIVIA | BENTLEY | 440-896-2135 | WILLOUGHBY | SEAFIELD AVENUE | 1 |
| 8 | MILLIE | BASSETT | 440-235-1845 | EASTLAKE | BELMONT ROAD | |
| 9 | HECTOR | FLOYD | 440-842-1525 | MENTOR | JACKSON STREET | 3 |
| 10 | ZAKK | SPEARS | 440-695-2351 | WILLOUGHBY | BEAUFORT AVENUE | 7 |
| 11 | ONUR | KINNEY | 216-512-2013 | WICKLIFFE | CROSS STREET | 4 |
| | | | | | | |
| CUST_ID | CUST_FNAME | CUST_LNAME | CUST_PHONE | CUST_CITY | CUST_STREET | CUST_ORD_COUNT |
| 12 | ISIS | BARRY | 216-215-8421 | MENTOR | BIRCH AVENUE | 5 |
| 13 | EMERSON | MCKAY | 440-548-1235 | WILLOUGHBY | ROWAN AVENUE | |
| 14 | NINA | WARE | 440-666-1523 | WICKLIFFE | ADLER LANE | 3 |
| 15 | FATEMA | PINEDA | 440-352-1542 | WILLOUGHBY | MANOR LANE | 1 |
| 16 | GRIFF | EASTON | 440-842-3269 | MENTOR | FAIRFIELD ROAD | 2 |
| 17 | LILLIE | CAIN | 216-654-8452 | | | 5 |
| 18 | RONNY | KIRBY | 440-215-1263 | | | 1 |
| 19 | AMEEN | PROSSER | 440-985-4252 | | | |
| 20 | LEONARDO | RAMSAY | 216-685-1253 | | | 4 |

20 rows selected.

Step Four

Develop SQL Queries

Now that we have the most tedious part finished, we can move on to some query testing. Let's start this by doing some random joins to see if there is any information that can be explored or extracted. This first one that will be performed is a *NATURAL JOIN* of the “CUSTOMER” and “ORDERS” tables.

```
SQL> SELECT * FROM CUSTOMER NATURAL JOIN ORDERS;
truncating (as requested) before column PIZZA_ID
truncating (as requested) before column OTH_ID
truncating (as requested) before column EMP_ID
```

| CUST_ID | CUST_FNAME | CUST_LNAME | CUST_PHONE | CUST_CITY | CUST_STREET | CUST_ORD_COUNT | ORD_ID | ORD_DAY_O | ORD_MONTH | ORD_T | ORD_TYPE |
|---------|------------|------------|--------------|------------|-----------------|----------------|--------|-----------|-----------|-------|----------|
| 1 | ZAINA | ALDRED | 440-654-3216 | MENTOR | MATTHEWS STREET | 2 | 1 | FRIDAY | JANUARY | 10:00 | DELIVERY |
| 2 | DARRELL | BENNETT | 216-654-6548 | WILLOUGHBY | EXMOUTH LAWN | 10 | 2 | SATURDAY | JUNE | 17:00 | DELIVERY |
| 3 | FERN | TURNER | 440-562-2135 | MENTOR | HERNE STREET | 4 | 3 | MONDAY | MAY | 18:00 | DELIVERY |
| 4 | NOEL | HOPPER | 440-986-5123 | WILLOUGHBY | SEAFORD DRIVE | 4 | 4 | THURSDAY | FEBRUARY | 15:00 | DELIVERY |
| 5 | SOFIE | WELLS | 440-541-3216 | EASTLAKE | ORGAN WAY | 4 | 5 | FRIDAY | APRIL | 11:00 | DELIVERY |
| 6 | HARVIR | STONE | 330-320-1562 | WILLOUGHBY | WARWICK ROAD | 2 | 6 | SATURDAY | MAY | 17:00 | DELIVERY |
| 7 | OLIVIA | BENTLEY | 440-896-2135 | WILLOUGHBY | SEAFIELD AVENUE | 1 | 7 | SUNDAY | JANUARY | 19:00 | DELIVERY |
| 8 | LILLIE | BASSETT | 440-235-1845 | EASTLAKE | BELMONT ROAD | 8 | 8 | WEDNESDAY | OCTOBER | 19:00 | DELIVERY |
| 9 | HECTOR | FLOYD | 440-842-1525 | MENTOR | JACKSON STREET | 3 | 9 | SATURDAY | NOVEMBER | 10:00 | DELIVERY |
| 10 | ZAKK | SPEARS | 440-695-2351 | WILLOUGHBY | BEAUFORT AVENUE | 7 | 10 | WEDNESDAY | OCTOBER | 17:00 | DELIVERY |
| 11 | ONUR | KINNEY | 216-512-2013 | WICKLIFFE | CROSS STREET | 4 | 11 | SUNDAY | SEPTEMBER | 19:00 | DELIVERY |
| 12 | ISIS | BARRY | 216-215-8421 | MENTOR | BIRCH AVENUE | 5 | 12 | THURSDAY | MARCH | 18:00 | DELIVERY |
| 13 | EMERSON | MCKAY | 440-548-1235 | WILLOUGHBY | ROWAN AVENUE | 3 | 13 | FRIDAY | JUNE | 12:00 | DELIVERY |
| 14 | NINA | WARE | 440-666-1523 | WICKLIFFE | ADLER LANE | 1 | 14 | FRIDAY | DECEMBER | 18:00 | PICK UP |
| 15 | FATEMA | PINEDA | 440-352-1542 | WILLOUGHBY | MANOR LANE | 1 | 15 | FRIDAY | FEBRUARY | 12:00 | PICK UP |
| 16 | GRIFF | EASTON | 440-842-3269 | MENTOR | FAIRFIELD ROAD | 2 | 16 | WEDNESDAY | DECEMBER | 16:00 | PICK UP |
| 17 | LILLIE | CAIN | 216-654-8452 | | | 5 | 17 | SATURDAY | NOVEMBER | 11:00 | PICK UP |
| 18 | RONNY | KIRBY | 440-215-1263 | | | 1 | 18 | SATURDAY | JULY | 20:00 | PICK UP |
| 19 | AMEEN | PROSSER | 440-985-4252 | | | 1 | 19 | FRIDAY | DECEMBER | 12:00 | PICK UP |
| 20 | LEONARDO | RAMSAY | 216-685-1253 | | | 4 | 20 | TUESDAY | NOVEMBER | 14:00 | PICK UP |

20 rows selected.

We will follow that up with a *LEFT OUTER JOIN* of the “EMPLOYEE” and “ORDERS” tables.

```
SQL> SELECT *
2 FROM EMPLOYEE LEFT JOIN ORDERS
3 ON EMPLOYEE.EMP_ID = ORDERS.EMP_ID;
```

| EMP_ID | EMP_FNAME | EMP_LNAME | EMP_PHONE | POS_ID | ORD_ID | ORD_DAY_O | ORD_MONTH | ORD_T | ORD_TYPE | CUST_ID | PIZZA_ID | OTH_ID | EMP_ID |
|--------|-----------|-------------|--------------|--------|--------|-----------|-----------|-------|----------|---------|----------|--------|--------|
| 1 | SAFWAN | WHITTINGTON | 440-665-4895 | 1 | 6 | SATURDAY | MAY | 17:00 | DELIVERY | 6 | 106 | 106 | 1 |
| 2 | JULIA | MATTHEWS | 216-326-5489 | 2 | 14 | FRIDAY | DECEMBER | 18:00 | PICK UP | 14 | 235 | 235 | 2 |
| 3 | JULIA | MATTHEWS | 216-326-5489 | 2 | 5 | FRIDAY | APRIL | 11:00 | DELIVERY | 5 | 236 | 236 | 2 |
| 3 | AMAN | BERRY | 440-512-3165 | 3 | 12 | THURSDAY | MARCH | 18:00 | DELIVERY | 12 | 10 | 10 | 3 |
| 4 | MAIZIE | TIMMS | 440-598-1532 | 4 | 18 | SATURDAY | JULY | 20:00 | PICK UP | 18 | 100 | 100 | 4 |
| 4 | MAIZIE | TIMMS | 440-598-1532 | 4 | 16 | WEDNESDAY | DECEMBER | 16:00 | PICK UP | 16 | 38 | 38 | 4 |
| 5 | JACK | LENNON | 216-321-5487 | 5 | 4 | THURSDAY | FEBRUARY | 15:00 | DELIVERY | 4 | 14 | 14 | 5 |
| 5 | JACK | LENNON | 216-321-5487 | 5 | 1 | FRIDAY | JANUARY | 10:00 | DELIVERY | 1 | 545 | 545 | 5 |
| 5 | JACK | LENNON | 216-321-5487 | 5 | 7 | SUNDAY | JANUARY | 19:00 | DELIVERY | 7 | 210 | 210 | 5 |
| 6 | NEV | BEIL | 440-656-8221 | 6 | 3 | MONDAY | MAY | 18:00 | DELIVERY | 3 | 76 | 76 | 6 |
| 6 | NEV | BEIL | 440-656-8221 | 6 | 11 | SUNDAY | SEPTEMBER | 19:00 | DELIVERY | 11 | 69 | 69 | 6 |
| 7 | GEORGE | HUTCHINSON | 440-542-1598 | 7 | 2 | SATURDAY | JUNE | 17:00 | DELIVERY | 2 | 192 | 192 | 7 |
| 7 | GEORGE | HUTCHINSON | 440-542-1598 | 7 | 8 | WEDNESDAY | OCTOBER | 18:00 | DELIVERY | 8 | 6 | 6 | 7 |
| 8 | MACAULAY | GREENAWAY | 440-548-3256 | 8 | 19 | FRIDAY | DECEMBER | 12:00 | PICK UP | 19 | 95 | 95 | 8 |
| 8 | MACAULAY | GREENAWAY | 440-548-3256 | 8 | 20 | TUESDAY | NOVEMBER | 14:00 | PICK UP | 20 | 364 | 364 | 8 |
| 9 | MIGUEL | GIBSON | 440-568-4526 | 9 | 10 | WEDNESDAY | OCTOBER | 17:00 | DELIVERY | 10 | 311 | 311 | 9 |
| 10 | ARCHIBALD | STARK | 330-654-5468 | 10 | | | | | | | | | |
| 11 | MYLA | GREAVES | 216-512-4865 | 11 | | | | | | | | | |
| 12 | JENSON | MURRAY | 440-516-5689 | 12 | 15 | FRIDAY | FEBRUARY | 12:00 | PICK UP | 15 | 155 | 155 | 12 |
| 13 | JANNAT | PHAN | 440-548-9623 | 13 | | | | | | | | | |
| 14 | GISELLE | JOHNS | 216-696-3216 | 14 | 17 | SATURDAY | NOVEMBER | 11:00 | PICK UP | 17 | 419 | 419 | 14 |
| 14 | GISELLE | JOHNS | 216-696-3216 | 14 | 13 | FRIDAY | JUNE | 12:00 | DELIVERY | 13 | 188 | 188 | 14 |
| 15 | DONNIE | BUCHANAN | 440-598-4123 | 15 | 9 | SATURDAY | NOVEMBER | 10:00 | DELIVERY | 9 | 421 | 421 | 15 |

23 rows selected.

Next will involve joining more than two tables together. This will be performed with an *INNER JOIN* on the “PIZZA”, “ORDERS”, and “OTHER” tables

```
SQL> SELECT *
  2 FROM ORDERS
  3 INNER JOIN PIZZA
  4 ON ORDERS.PIZZA_ID = PIZZA.PIZZA_ID
  5 INNER JOIN OTHER
  6 ON ORDERS.OTH_ID = OTHER.OTH_ID;
rows will be truncated
rows will be truncated
rows will be truncated
```

| ORD_ID | ORD_DAY_O | ORD_MONTH | ORD_T | ORD_TYPE | CUST_ID | PIZZA_ID | OTH_ID | EMP_ID | PIZZA_ID | PIZZA_TOP_ | PIZZA_PRICE |
|--------|-----------|-----------|-------|----------|---------|----------|--------|--------|----------|------------|-------------|
| 18 | SATURDAY | JULY | 20:00 | PICK UP | 18 | 100 | 100 | 4 | 100 | 5 | 26 |
| 17 | SATURDAY | NOVEMBER | 11:00 | PICK UP | 17 | 419 | 419 | 14 | 419 | 3 | 20 |
| 4 | THURSDAY | FEBRUARY | 15:00 | DELIVERY | 4 | 14 | 14 | 5 | 14 | 1 | 10 |
| 14 | FRIDAY | DECEMBER | 18:00 | PICK UP | 14 | 235 | 235 | 2 | 235 | 1 | 10 |
| 6 | SATURDAY | MAY | 17:00 | DELIVERY | 6 | 106 | 106 | 1 | 106 | 1 | 10 |
| 1 | FRIDAY | JANUARY | 10:00 | DELIVERY | 1 | 545 | 545 | 5 | 545 | 3 | 20 |
| 8 | WEDNESDAY | OCTOBER | 18:00 | DELIVERY | 8 | 6 | 6 | 7 | 6 | 2 | 16 |
| 16 | WEDNESDAY | DECEMBER | 16:00 | PICK UP | 16 | 38 | 38 | 4 | 38 | 2 | 16 |
| 7 | SUNDAY | JANUARY | 19:00 | DELIVERY | 7 | 210 | 210 | 5 | 210 | 3 | 20 |
| 15 | FRIDAY | FEBRUARY | 12:00 | PICK UP | 15 | 155 | 155 | 12 | 155 | 1 | 10 |
| 10 | WEDNESDAY | OCTOBER | 17:00 | DELIVERY | 10 | 311 | 311 | 9 | 311 | 3 | 20 |

| ORD_ID | ORD_DAY_O | ORD_MONTH | ORD_T | ORD_TYPE | CUST_ID | PIZZA_ID | OTH_ID | EMP_ID | PIZZA_ID | PIZZA_TOP_ | PIZZA_PRICE |
|--------|-----------|-----------|-------|----------|---------|----------|--------|--------|----------|------------|-------------|
| 3 | MONDAY | MAY | 18:00 | DELIVERY | 3 | 76 | 76 | 6 | 76 | 5 | 26 |
| 19 | FRIDAY | DECEMBER | 12:00 | PICK UP | 19 | 95 | 95 | 8 | 95 | 4 | 23 |
| 13 | FRIDAY | JUNE | 12:00 | DELIVERY | 13 | 188 | 188 | 14 | 188 | 1 | 26 |
| 9 | SATURDAY | NOVEMBER | 10:00 | DELIVERY | 9 | 421 | 421 | 15 | 421 | 4 | 23 |
| 12 | THURSDAY | MARCH | 18:00 | DELIVERY | 12 | 10 | 10 | 3 | 10 | 1 | 10 |
| 5 | FRIDAY | APRIL | 11:00 | DELIVERY | 5 | 236 | 236 | 2 | 236 | | |
| 11 | SUNDAY | SEPTEMBER | 19:00 | DELIVERY | 11 | 69 | 69 | 6 | 69 | | |
| 2 | SATURDAY | JUNE | 17:00 | DELIVERY | 2 | 192 | 192 | 7 | 192 | | |
| 20 | TUESDAY | NOVEMBER | 14:00 | PICK UP | 20 | 364 | 364 | 8 | 364 | | |

20 rows selected.

Finally, a *VIEW* will be created to include the “EMP_ID”, “EMP_FNAME”, and “EMP_LNAME” from the “EMPLOYEE” table. This helps us get a quick view of the employees on board with their respective ID’s.

```
SQL> CREATE VIEW EMP_FULL_NAME AS
  2  SELECT EMP_ID, EMP_FNAME, EMP_LNAME
  3  FROM EMPLOYEE;
```

View created.

```
SQL> SELECT * FROM EMP_FULL_NAME;
```

| EMP_ID | EMP_FNAME | EMP_LNAME |
|--------|-----------|-------------|
| 1 | SAFWAN | WHITTINGTON |
| 2 | JULIA | MATTHEWS |
| 3 | AMAN | BERRY |
| 4 | MAIZIE | TIMMS |
| 5 | JACK | LENNON |
| 6 | NEV | BEIL |
| 7 | GEORGE | HUTCHINSON |
| 8 | MACAULAY | GREENAWAY |
| 9 | MIGUEL | GIBSON |
| 10 | ARCHIBALD | STARK |
| 11 | MYLA | GREAVES |
| EMP_ID | EMP_FNAME | EMP_LNAME |
| 12 | JENSON | MURRAY |
| 13 | JANNAT | PHAN |
| 14 | GISELLE | JOHNS |
| 15 | DONNIE | BUCHANAN |

15 rows selected.

Step Five

Develop Queries for Analysis

The first queries we will write will involve trying to obtain when sales are occurring primarily for Just Pizza! The first of these will be finding when on what day of the week the most sales are. This query gave us the result that *Friday* is the day in which the highest volume of orders occurs.

```
SQL> SELECT ORD_DAY_OW, COUNT(*) AS "NUM"  
2 FROM ORDERS  
3 GROUP BY ORD_DAY_OW;
```

| ORD_DAY_O | NUM |
|-----------|-----|
| SATURDAY | 5 |
| SUNDAY | 2 |
| MONDAY | 1 |
| FRIDAY | 6 |
| THURSDAY | 2 |
| WEDNESDAY | 3 |
| TUESDAY | 1 |

```
7 rows selected.
```

The second query will involve discover what time of day customers tend to order at the most. The result gave us the most orders occurring at “18:00.” Right behind were “17:00” and “12:00” having the same amount.

```
SQL> SELECT ORD_TIME, COUNT(*) AS "NUM"  
2 FROM ORDERS  
3 GROUP BY ORD_TIME;
```

| ORD_T | NUM |
|-------|-----|
| 10:00 | 2 |
| 12:00 | 3 |
| 20:00 | 1 |
| 17:00 | 3 |
| 19:00 | 2 |
| 16:00 | 1 |
| 14:00 | 1 |
| 18:00 | 4 |
| 15:00 | 1 |
| 11:00 | 2 |

```
10 rows selected.
```

The third and final query in the set looks at what months and time of the year sales occur the most. Upon observation it's clear to see *November* and *December* are the busiest months. As well, the time of the year where Just Pizza! receives the most business is all *Winter* and *Late Fall*.

```
SQL> SELECT ORD_MONTH, COUNT(*) AS "NUM"  
2 FROM ORDERS  
3 GROUP BY ORD_MONTH;
```

| ORD_MONTH | NUM |
|-----------|-----|
| OCTOBER | 2 |
| FEBRUARY | 2 |
| JANUARY | 2 |
| DECEMBER | 3 |
| SEPTEMBER | 1 |
| APRIL | 1 |
| MARCH | 1 |
| NOVEMBER | 3 |
| JUNE | 2 |
| MAY | 2 |
| JULY | 1 |

```
11 rows selected.
```

The next queries involve on understanding which employee is the most productive. Also, it will involve understanding who the best at their own position is. This will give perspective on the people who most deserve a possible raise and/or promotion.

```
SQL> SELECT EMP_FNAME, EMP_LNAME, POS_TYPE, COUNT(EMP_FNAME) AS "NUM"
2  FROM ORDERS, EMPLOYEE, POSITION
3  WHERE EMPLOYEE.POS_ID = POSITION.POS_ID
4  AND ORDERS.EMP_ID = EMPLOYEE.EMP_ID
5  GROUP BY EMP_FNAME, EMP_LNAME, POS_TYPE
6  ORDER BY POS_TYPE DESC;
```

| EMP_FNAME | EMP_LNAME | POS_TYPE | NUM |
|-----------|-------------|-----------------|-----|
| AMAN | BERRY | MANAGER | 1 |
| JULIA | MATTHEWS | MANAGER | 2 |
| MAIZIE | TIMMS | MANAGER | 2 |
| NEV | BEIL | INSIDER | 2 |
| MACAULAY | GREENAWAY | INSIDER | 2 |
| GEORGE | HUTCHINSON | INSIDER | 2 |
| JACK | LENNON | INSIDER | 3 |
| SAFWAN | WHITTINGTON | GENERAL MANAGER | 1 |
| MIGUEL | GIBSON | DRIVER | 1 |
| GISELLE | JOHNS | DRIVER | 2 |
| JENSON | MURRAY | DRIVER | 1 |
| EMP_FNAME | EMP_LNAME | POS_TYPE | NUM |
| DONNIE | BUCHANAN | DISHWASHER | 1 |

12 rows selected.

The results show us that the overall most efficient employee is *Jack Lennon*. He should be immediately considered for possible promotion to management. Following this, it shows the *Julia Matthews* and *Maizie Timms* as managers were equally as efficient. Finally, the most productive driver is *Giselle Johns*.

On this next query, we compiled information to let us see what each employee is receiving hourly in wages. By observing our top performers, we can tell make an interesting observation. While *Jack Lennon* was the top performer this past year, he is also making the highest hourly rate in comparison to the other insiders. This shows that there is consistency in his working effort. This allows us to confidently state that he should receive a promotion to manager. For current managers *Julia Matthews* is already earning a higher wage than the other two managers. This gives us the opportunity to reward *Maizie Timms* with a raise for her matched effort to the pizza shops most efficient manager. Finally, *Giselle Johns* should be considered for a bump in pay as a driver.

```
SQL> SELECT EMP_FNAME, EMP_LNAME, POS_TYPE, POS_HOURLY
2  FROM EMPLOYEE, ORDERS, POSITION
3  WHERE ORDERS.EMP_ID = EMPLOYEE.EMP_ID
4  AND EMPLOYEE.POS_ID = POSITION.POS_ID
5  GROUP BY EMP_FNAME, EMP_LNAME, POS_TYPE, POS_HOURLY
6  ORDER BY POS_TYPE DESC;
```

| EMP_FNAME | EMP_LNAME | POS_TYPE | POS_HOURLY |
|-----------|-------------|-----------------|------------|
| AMAN | BERRY | MANAGER | 14 |
| JULIA | MATTHEWS | MANAGER | 16 |
| MAIZIE | TIMMS | MANAGER | 14 |
| NEV | BEIL | INSIDER | 10 |
| MACAULAY | GREENAWAY | INSIDER | 10 |
| GEORGE | HUTCHINSON | INSIDER | 10 |
| JACK | LENNON | INSIDER | 11 |
| SAFWAN | WHITTINGTON | GENERAL MANAGER | 25 |
| MIGUEL | GIBSON | DRIVER | 5 |
| GISELLE | JOHNS | DRIVER | 5 |
| JENSON | MURRAY | DRIVER | 5 |
| EMP_FNAME | EMP_LNAME | POS_TYPE | POS_HOURLY |
| DONNIE | BUCHANAN | DISHWASHER | 6 |

12 rows selected.

The following query set inquires on what the average amount of toppings are per pizza and what is the most ordered total number of toppings on a pizza. To answer the first observation, the average amount of toppings ordered are 2.5. This number doesn't help us too much. For the second observation, the information becomes clear. The most ordered number of toppings on pizza is the usual "*one*" topping. Behind that is a "*three*" topping.

```
SQL> SELECT AVG(PIZZA_TOP_COUNT) AS "AVERAGE PIZZA TOPPING AMOUNT"  
2 FROM PIZZA;
```

```
AVERAGE PIZZA TOPPING AMOUNT  
-----  
2.5
```

```
SQL> SELECT PIZZA_TOP_COUNT AS "TOP AMOUNT", COUNT(*) AS NUM  
2 FROM ORDERS JOIN PIZZA  
3 ON ORDERS.PIZZA_ID = PIZZA.PIZZA_ID  
4 GROUP BY PIZZA_TOP_COUNT  
5 ORDER BY NUM DESC;
```

```
TOP AMOUNT      NUM  
-----  
1              6  
3              4  
              4  
4              2  
5              2  
2              2  
  
6 rows selected.
```


The upcoming query looks at what is the most common type of item “*other*” than pizza is ordered. When we run the query, we instantly notice an error. There is a typo for two entries in the “*BREAD*” category. This requires a quick query to clean up and fix this.

```
SQL> SELECT OTH_ITEM, COUNT(*) AS "NUM"  
2 FROM OTHER JOIN ORDERS  
3 ON OTHER.OTH_ID = ORDERS.OTH_ID  
4 GROUP BY OTH_ITEM  
5 ORDER BY NUM;
```

| OTH_ITEM | NUM |
|----------|-----|
| BREAD | 2 |
| COOKIE | 3 |
| BREAD | 4 |
| WINGS | 4 |
| | 7 |

We will write a query using the *Update* function on the two respective rows containing the typos.

```
SQL> UPDATE OTHER  
2 SET OTH_ITEM = 'BREAD'  
3 WHERE OTH_ID = '545';
```

1 row updated.

```
SQL> UPDATE OTHER  
2 SET OTH_ITEM = 'BREAD'  
3 WHERE OTH_ID = '155';
```

1 row updated.

After the Update function was used, the results concatenate into their proper places leaving us with a nicer looking outcome. With this corrected output, it finally gives us the answer that “*BREAD*” is indeed the highest selling item that is not pizza.

```
SQL> SELECT OTH_ITEM, COUNT(*) AS "NUM"  
2   FROM OTHER JOIN ORDERS  
3   ON OTHER.OTH_ID = ORDERS.OTH_ID  
4   GROUP BY OTH_ITEM  
5   ORDER BY NUM;
```

| OTH_ITEM | NUM |
|----------|-----|
| COOKIE | 3 |
| WINGS | 4 |
| BREAD | 6 |
| | 7 |

The next query involves understanding where most of our customers reside. This information could be used to mail out coupons for promotional reasons to entice the lower attending city customers to come eat at the pizza shop. The result has shown us that *Willoughby* receives the most customers, which makes sense since the store is in the same city. The lowest attending cities are *Wickliffe* and *Eastlake*. These should be our targets.

```
SQL> SELECT CUST_CITY, COUNT(*) AS "NUM"  
2 FROM CUSTOMER JOIN ORDERS  
3 ON CUSTOMER.CUST_ID = ORDERS.CUST_ID  
4 GROUP BY CUST_CITY  
5 ORDER BY NUM DESC;
```

| CUST_CITY | NUM |
|------------|-----|
| WILLOUGHBY | 7 |
| MENTOR | 5 |
| | 4 |
| WICKLIFFE | 2 |
| EASTLAKE | 2 |

The final query we will be using is to find the most common pairing of topping amount and items that are other than pizza. With this, it shows that the standard “*one-topping pizza*” and “*wings*” are the most purchased combination. This could be a good idea for a future promotion.

```
SQL> SELECT OTH_ITEM, PIZZA_TOP_COUNT, COUNT(ORD_ID) AS "NUM"
  2  FROM ORDERS, PIZZA, OTHER
  3  WHERE ORDERS.PIZZA_ID = PIZZA.PIZZA_ID
  4  AND ORDERS.OTH_ID = OTHER.OTH_ID
  5  GROUP BY OTH_ITEM, PIZZA_TOP_COUNT
  6  ORDER BY NUM DESC;
```

| OTH_ITEM | PIZZA_TOP_ | NUM |
|----------|------------|-----|
| | 3 | 2 |
| WINGS | 1 | 2 |
| | 1 | 2 |
| BREAD | | 2 |
| | 4 | 2 |
| COOKIE | 5 | 1 |
| WINGS | | 1 |
| BREAD | 3 | 1 |
| BREAD | 2 | 1 |
| WINGS | 3 | 1 |
| COOKIE | | 1 |

| OTH_ITEM | PIZZA_TOP_ | NUM |
|----------|------------|-----|
| | | |
| COOKIE | 1 | 1 |
| | 2 | 1 |
| BREAD | 1 | 1 |
| BREAD | 5 | 1 |

15 rows selected.

Final Report

The time has come to give out results to Just Pizza! after much speculation and query analysis. We will parse out the ideas in this report in a systematic fashion. There have been many discoveries within our findings that can help push the pizza shop into an upward trajectory. With this report, we hope that it leaves the impression as it is intended.

The first analyses through queries that were completed had the goal in mind of understanding when sales occurred the most. This boiled down to indicators such as *Day of Week*, *Time of Day*, and *Month/Season*. The results gave seemingly standard answers for any person that is involved in the pizza business. The busiest day is *Friday*, with *Saturday* right behind it emphasizing that the weekend is the store's most engaged time of the week. As well, *18:00* was the busiest time of the day with *17:00* and *12:00* right behind it. Finally, the busiest months of the year are *November* and *December*. This helps bolster the busiest time of the year is *Winter* and *Late Fall*.

With this information, special promotions can be employed to encourage more business in the busy months as well as the idle months. It's preferable to run more significant campaigns in the idle months to help increase customer turnout; especially if the store is operating on a limited budget. Also, the store would want to look at promoting more common offers during slow times in the day and days of the week with the lowest sales. Finally, the store could run the antithesis of the other campaign ideas and have an extremely targeted approach towards its busiest times for all three categories. This would involve running a super promotion in *Late Fall / Winter* at *18:00* on *Friday / Saturday*. This would lead to heavy anticipation for customers and possibly lead to more exposure through word of mouth.

The second analyses were homed in on the employees, specifically because there seems to be some forms of unrest and unhappiness within the store. Running these queries helps gain more insight on how to approach this issue. More specifically in mind, it was used to understand whom the most productive of the employees are and who may be up for promotion or a raise. This would do a lot in helping ease some of this dissatisfaction within the store.

Within these findings, things were made clear on where and how to approach these issues. The most efficient employee in the store was shown to be *Jack Lennon* as an insider. He

also is currently receiving the highest hourly rate in comparison to the other insiders. This bolsters the idea that *Jack Lennon* should immediately be considered for promotion to management. Otherwise, the most efficient employees based on their position were *Julia Matthews* and *Maizie Timms* at manager and *Giselle Johns* at driver. *Giselle Johns* should be considered for a raise based on her performance instantly. The manager situation is a little more complicated, though. This is the case because both top-performing managers were efficient at the same rate. Upon further insight, a significant observation can be made. *Julia Matthews* is already receiving a higher hourly rate compared to the other two managers. This means that *Maizie Timms* should receive a raise for her matching efforts.

The third and final analyses gives us insight on the items that are being sold and at what rate. Also, as to understanding what cities the customers live in. This will be another effort in understanding what promotions can be ran based on the hidden details behind the sales of these items. This can be a crucial component in reviving the customer base of Just Pizza!

The first query made was understanding the average pizza topping amount which gave us an answer “2.5.” This was insignificant so switched the focus on the most common amount of toppings per pizza, which was *one*. Next, we looked to see what was the most sold item that wasn’t pizza. After some corrections in our table, it gave us the result that *Bread* takes the crown. Furthermore, our next query focuses on understanding where the customer base primarily resides as well as where it doesn’t. Finding the primary city gave us the expected result of *Willloughby*, specifically because this is where the pizza shop is located. The two lowest ordering cities were *Wickliffe* and *Eastlake*. The final query made was to understand which two items sold the most in combination with one another. These two items ended up being a “*One-Topping Pizza*” alongside “*Wings*.”

The final portion of this analysis leads us here. With this final set of information, Just Pizza! should run its most aggressive promotional campaign. This would entail targeting the lower purchasing areas with deals of the most paired items. It should be a significant discount, specifically for the loyal customers who became jaded over the years. A *one-topping pizza* paired with *wings* mailed out to patrons in the cities of *Wickliffe* and *Eastlake* should significantly boost business.

With this analytical report ending, our agency is confident that Just Pizza! has been handed information at highest extent of quality. We believe they can improve their sales situation by listening to the advice given. Whether it be promotional campaigns that are time sensitive and selective, possible raises and promotions for current employees, or offers targeted at a specific subsection of the local area, confidence is bestowed that their circumstances will be restored. Hopefully Just Pizza! listens and is able to remain lovable restaurant and staple of the Greater Cleveland Area.