



CMPE 226 Database Systems - Project Report

Submitted by - Team Goat

Christopher Rehfeld

Haritha Peyyeti

Reshma Sudha

Somya Aggarwal

CMPE 226 (SPRING 2016)

Submitted to:

Prof. Ron Mak

Table of Contents

1. Application Introduction.....	4
2. Functional requirements	4
3. Non Functional Requirements	5
4. Application.....	5
5. Configuration.....	6
6. Data Sources	6
6.1 Product Data.....	6
6.2 User Data	7
6.3 Orders & Reviews	7
7. Data Synchronization - Operational to Analytical Tables.....	8
8. Tools Used	10
9. Why Tableau	10
10. Technologies Used.....	11
11. Data Models Overview	11
11.1 Operational Data Model	11
11.2 Analytical Data Model.....	12
11.2.1 Dimension Tables:.....	12
11.2.2 Fact Tables:.....	12
12. ER Diagram	13
13. Relational Schema.....	14
14. Star Schemas	15
15. Operational Queries:	15
15.1 Web Page Forms	15
15.1.1 Customer Registration:.....	16
15.1.2 Update Customer Information:.....	16
15.1.3 Order Lookup:.....	17
15.1.4 Product Lookup:	17
15.1.5 Submit Review:	18
16. Analytical Queries	19
16.1 Finding the most profitable brand.....	19
16.2 Product Popularity By Brand	20
16.3 Total Sales by Zip code	20
16.4 Sales By Calendar.....	21
16.5 Average rating for a product.....	22
17. OLAP Queries	23

17.1	Drill Up	23
17.2	Drill Down:	24
17.3	Slice:.....	24
17.4	Dice:	25
18.	Enhancements Planned	26

1. Application Introduction

Our online shopping system will manage data related to users, orders, products, shipments, reviews, and payments. It has the data model of a typical e-commerce website where you make an account, add products to a shopping cart, pay for it, and then we ship the goods.

Each user registers to create an account, which has 1 username, and password. All the basic details of a user like First Name, Last Name, Phone Number(s), Address, City, Zip and Email Address will be stored in the database. Each Account can have 1 or more addresses associated with it for contact, billing, and shipping purposes.

We store data about products we sell such as Name, Category, Price, Brand, Image, Model and Description. A user can use their account to place orders for products. An order contains 1 or more products, 1 payment, 1 shipping address, 1 billing address.

Also, the system is capable of maintaining the order details, where each order will be identified by an Order ID and a Payment ID which in turn is mapped to the unique Customer from which the order has been placed. We keep enough data about the order so that we could produce an invoice/receipt. Using that data, the receipt could be laid out like a typical retail store receipt, where it lists 1 product per line along with the product price, how many of that product, and then the extended price for that line which is the product price x qty. Also, the lines are ordered, with the first line corresponding to the first product that the user selected.

A user can easily track the status of his shipment as the order and shipment details are also stored and updated as progress is made on the order delivery. An order must be paid in full using only 1 payment method, and it must be shipped in full all at once – we wait until everything is in stock before we ship.

Our return policy is “No Return or Refunds”, for now. We only accept credit card payments. We don’t store credit card info, but we do store the payment gateway authorization code for verification & bookkeeping purposes.

We also support a rating and review system for the customers. A customer can rate any of our products, where they specify a numerical rating value, which product, and optionally some comments.

2. Functional requirements

All the functional requirements laid out for the database are met.

- Every customer needs to be identified using a unique id.
- Every customer needs to have a unique login credentials (user id & password). Password need be stored in hash encrypted format. It is an essential requirement for security.
- A customer can have multiple shipping/billing addresses. Database need to store all the addresses associated to a customer. Address database also should contain

customer's email id and any telephone numbers they provide, plus the customer's physical address.

- Every order should be uniquely identifiable. Each order should be attributed to a Customer.
- Order should have shipping details and associated with a unique payment id
- Every product should be identified with a unique id.
- Each product is hierarchically categorized, first by a subcategory, and then each subcategory is categorized by a regular category.
- Database needs to store product name and price.
- Payment must have a unique id and should be associated a billing address
- We need to store the payment amount.
- Database should store reviews optionally provided by customers.
- The review database should also store all attributes such as review comments, product id and a rating value

3. Non Functional Requirements

Some of the non-functional requirements are implemented. For others the implementation plan or proposal is mentioned below:

- Database should have all the security mechanism to avoid any malicious queries, e.g. SQL injection etc. We implemented this requirement by using prepared statements with bound parameters.
- Database should have an access control mechanism. It should have permissions at read/write/create level. Permissions should be clearly set on individual/group/public Level. We can have individual database tables for admins and users.
- Database should be scalable. With growing customers and products, it should be
- Scalable without any redesign. Our tables are created in such a way that the database is scalable.
- Data Archival - Older records (may be 5+ years) can be moved to another storage media or a data warehouse for analysis purposes.
- Database should have redundancy for load handling as well as backup.
- We can have a script running on the database server to take the backup periodically. Also, there can be replications to the existing database (master slave configurations etc.)
- Database should be consistent across different replicas. There should be a facility to fix any inconsistency manually.
- Security aspect is implemented by saving the passwords as hash values in the database instead of plain text.
- Performance and Load - It should support sharing with consistent hashing so that it can be scaled with time and able to perform efficiently with the increasing customer load.

4. Application

We used php, mysql, and apache to make a web app that drove many of our operational data interactions. The typical structure of one of our php pages includes an html form for

user input, and usually some html tables for displaying tabular output data, such as a search result list. Our various php pages used php's PDO module for database interaction, making exclusive use of prepared sql statements and bound parameters everywhere to avoid sql injection and ensure program correctness. Additionally, we used database transactions to make sure that our operations functioned atomically as a unit like we intended. This way if an unexpected error occurs in the middle of a series of sql statements, we don't leave our db in an inconsistent state.

5. Configuration

We have many php scripts that do database interaction. We created some reusable code to get a PDO database connection that was pre-configured to throw exceptions on errors. The connection username, password, database host etc... are sourced from an xml configuration file. This made it easy have multiple connection profiles to the various databases versions that we used during the development phase of the project. We only needed to update the profile name in one place and all pages would use the new connection configuration.

6. Data Sources

We had a difficult time finding a real dataset that gave us something suitable for use with our schema. We basically needed the records of an ecommerce website - products, customers, and their orders. But, considering most ecommerce sites that have real product data are commercial and unlikely to want to share, and also that real customer orders are considered private, we had to improvise and generate our own data.

6.1 Product Data

We found a free database from www.icecat.us that listed real products, along with a model number, category name, description, and some other fields.

Unfortunately it didn't give any suggested prices, so we devised an algorithm to guess a price. We basically generate a random price between 0 to \$10,000 but we used random weighting to favor prices that are close to multiple of 10, 100, and 1000, as well as ending in .99, .00, or .60. This way we got semi realistic looking prices such as \$99.99, \$499.00, \$3.99 etc...

It also didn't provide any product images, and our db needed at least some image file names, so we generated fake image urls by concatenating the part number with ".png".

Another shortcoming was that there was only 1 level of category for the product, and our db wants to levels. For example, our schema envisioned a category named "Electronics", and then a subcategory named "Cell Phones". Since this data set only gave us 1 category, we just improvised and made all products have the same super category - "Main".

This product data came as a large tab separated gzipped file. Each column had a header name, and was always separated by exactly 3 tab characters. We wrote a php

script to parse this file row by row, preprocessing the data to fit our needs, and then importing into our database. The file had millions of entries but we only used 500,000 to keep the data size easy to work with. The name of the script, `import-products.php` makes calls to a class named `ProductDataImporter`, which encapsulated the logic needed to parse the file, generate any missing pieces of data, and then process the data into suitable records for our db.

6.2 User Data

For user data, we found an awesome web service, www.mockaroo.com, that helps you generate fake data. We were able to specify what fields we needed: `FirstName`, `LastName`, `Email`, `StreetAddress`, `Zipcode`, `City`, `State`, and `PhoneNumber` - and they literally generated it for us! It produced nice random but realistic looking data that fit the expected domain of each field - eg, a phone number looks like a phone number, a first name like a first name etc... The only field we had to generate on our own was the user's password hash.

They let us pick the output format, for which we chose csv. To import this data, we wrote a script named `import-users.php` which calls a class named `UserDataImporter`, which parses the records, processes them mildly, generates a fake password hash, and then inserts them into our db.

6.3 Orders & Reviews

We decided we needed to generate some fake orders, simulating a user purchasing a few products, paying for it, and having it shipped to them. Our simulated user data from mockaroo was placed into an ancillary table named `simulateduserdata` so that our real operational tables for user data, such as the `customer` and `addressablecontact` tables, only contained data for customers who had placed an order.

We wrote a script named `generate-orders.php` which calls a class named `SimulatedOrderGenerator` to create these fake orders for us. It also generates a fake review for each user. The script allows you specify some inputs such as how many orders to make, and a data range to use when picking a random date for each order. When you run the script it picks N random users from the `simulateduserdata` table, and then makes an order for each.

When making the order, it will randomly pick 1 to 5 products, and randomly choose a quantity between 1 to 3 for each. Once the total is computed, a payment record is created for the order as well as any other records that are needed. Just about every table in the db is involved here, either reading or writing, in order to create an order.

Additionally, we generate 1 fake product review at the same time. We pick one of the products the user purchased, and we pick a random rating value, and create a random text comment. To make the comment we found some "lorem ipsor dolor" style sentences in someone's github project. We pick a few of these sentences at random and shuffle their order.

7. Data Synchronization - Operational to Analytical Tables

We wrote a php script which migrates and synchronizes our data from our operational to analytical tables. Most of the work is done by carefully crafted sql statements that select data from various operational tables and insert/update the analytical tables using that data. The script named `migrate-operational-to-analytical.php` calls a class named `OperationalToAnalyticalDataProcessor` which executes the sql statements. The sql is located in a file named `move-operational-to-analytical.sql`, which the php script reads and executes.

The strategy is to run the script periodically, say once a day or week via a cron job, to keep the analytical db in sync with the operational db. Given the nature of our business, being an ecommerce site, we don't see a large need to delete records. Customers may come back a decade later and still want to see their previous orders, so we don't want to delete any data, or at least none of the values in our current set of tables. A real ecommerce site would have more tables and surely some of them wouldn't be desirable to keep forever.

This simplifies the synchronization task. We need to insert new records, update existing ones, and recompute aggregations. We were able to sync every analytic table by using an insert statement that obeys carefully selected unique constraints on certain columns. We make a lot of use of "insert ignore ..." and "insert ... on duplicate key update". Since we don't delete data from our operational tables, each time we sync we will be trying to copy the same existing records into the analytic tables again. However, our unique constraints will allow us to easily detect these duplicates, and either ignore the insert, or in some other cases it will update the record by using the new values.

Additionally, since this update process issues many update and insert statements, we use a transaction to wrap them all so that if there were to be a problem, we can rollback and cancel the entire sync

for the day, giving an admin time to investigate the issue. We wouldn't want to update some tables without updating others - this would leave the database in an inconsistent state and give us bad data.

Below we will detail the specific strategies we used for synchronizing each of our analytical tables:

Calendar - For our date table, we wrote a stored procedure that generates a row for each day between the start and end date. Each time we run the sync script, we use the current date as the upper bound. This ensures that our calendar table always has enough dates to cover our records.

```
create procedure fillDates(dateStart DATE, dateEnd DATE)
begin
```



```

while dateStart <= dateEnd do
  insert ignore into cmpe226a6.CALENDAR (CalendarKey, Date, Month, Quarter,
DayOfWeek, Year)
    values (null, dateStart, month(dateStart), quarter(dateStart),
dayofweek(dateStart), year(dateStart));
  set dateStart = date_add(dateStart, interval 1 day);
end while;
end;

...

call fillDates('2010-01-01', '2016-12-31');

```

Product - We joined our operational product and brand tables and then insert a subset of the columns it into the analytical product table. If duplicate keys are encountered we update the rows.

Customer - We joined our operational customer, addressablecontact, and zipcode tables and then insert a subset of the columns into the analytical customer table. If duplicate keys are encountered we update the rows.

Customer_Location - We joined our operational customer, addressablecontact, and zipcode tables and then insert a subset of the columns into the analytical customer_location table. If duplicate keys are encountered we update the rows.

Order - We joined our operational order and payment tables and then insert a subset of the columns into the analytical order table. If duplicate keys are encountered we update the rows.

Location - We joined our operational city, state, and zipcode tables and then insert a subset of the columns into the analytical location table. If duplicate keys are encountered we update the rows.

Sales - We joined our operational order, orderline, addressablecontact, zipcode, city, and state tables and joined those to our analytical product, customer_location, order, location, calendar, and product and then insert a subset of the columns into the analytical sales table. If duplicate keys are encountered we update the rows.

Product_Popularity - We joined our operational order and orderline tables and joined those to our analytical product and calendar tables, grouping by the date and summing the quantity, and then insert a subset of the columns into the analytical product_popularity table. If duplicate keys are encountered we update the rows.

Order_History - We joined our operational order, and orderline tables and joined those to our analytical product, order, and customer tables and then insert a subset of the columns into the analytical order_history table. If duplicate keys are encountered we update the rows.

Review - We selected from our operational review table and then insert a subset of the columns into the analytical review table. If duplicate keys are encountered we update the rows.

Feedback - We joined our operational review table with our analytical product, review, calendar, and customer tables and then insert a subset of the columns into the analytical order_history table. If duplicate keys are encountered we update the rows.

AvgRating - We joined our operational review table with our analytical product table, grouping by the ProductKey and averaging the rating, and then insert a subset of the columns into the analytical avgrating table. If duplicate keys are encountered we update the rows.

To save space, we will show only one of the above queries here as an example:

```
insert
  into cmpe226a6.avgrating
    (ProductKey, AverageRating)
select t.ProductKey, t.ar
  from (
    select pp.ProductKey, AVG(r.Rating) ar
      from cmpe226operational.review r
    join cmpe226a6.PRODUCT pp
      on r.ProductId = pp.ProductId
    group
      by pp.ProductKey
    ) t

on duplicate key update
  AverageRating = t.ar
;
```

8. Tools Used

- XAMPP
- MYSQL Server for creating the database and tables
- ERD Plus for creating the ER diagram, relational schema and star schema.
- Tableau for the analytical queries.

9. Why Tableau

- Ease of connection with local MYSQL server.
- Easy import of existing databases
- Flexibility of User Input

- No Hard coding in SQL Queries
- Multiple filters can be applied
- Data Visualization

10. Technologies Used

- HTML
- CSS
- PHP
- SQL
- Data Warehousing
- XML
- CSV

11. Data Models Overview

11.1 Operational Data Model

Below are the tables in 3NF in the Operational data model:

- Customer - Customer Id, User name and password are stored in this table.
- AddressableContact - This table contains all the customer related details like customer's first name and last name, email id, street address and zipcode.
- ContactPhone - This table contains all the phone numbers pertaining to a customer mapped with addressablecontact key.
- Zipcode, City, State are included as separate tables.
- Payment - This table contains payment amount, authorization code and payment id linked to customer id.
- Order - This table give order details like order id, status and the date of order linked to payment id and customer id.
- OrderLine - It will store each line from the order (description of one item in the order).
- Review- This table contains the rating, review description,date of review for each product and the customer id of the reviewer.
- Product - Product details like id, name, model, price, imageurl.
- Brand - Brand details are stored in this table. E.g. Apple, Samsung etc.
- Category - Product category id and details are stored in this table. E.g. Electronics, Clothing etc.
- Subcategory - Product subcategory id and details are stored in this table. E.g. Phones, Laptops etc.

11.2 Analytical Data Model

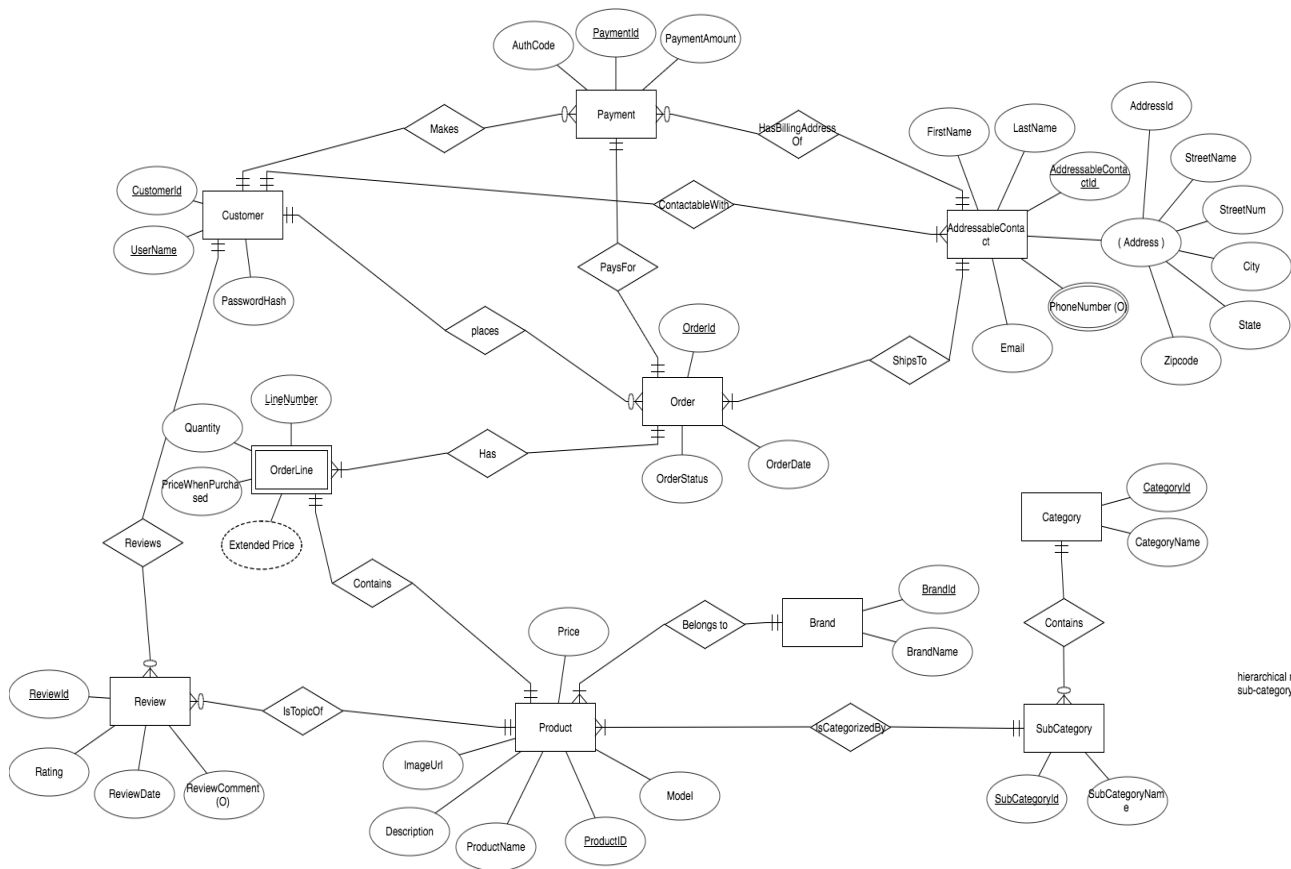
11.2.1 Dimension Tables:

- **Customer** : It will have the customer details pulled up from the customer operational table
- **Order** : It will have the order details populated from the orders operational table.
- **Location** : It will have the location attributes generated from the location, city, state and zipcode operational tables.
- **Product** : It will have the attributes from the product and brand operational tables.
- **Calendar**: Calendar fields like month, day, date, year and quarter will be a part of this dimension table.
- **Review** : It will contain the review operational table attributes - like rating, comment etc.

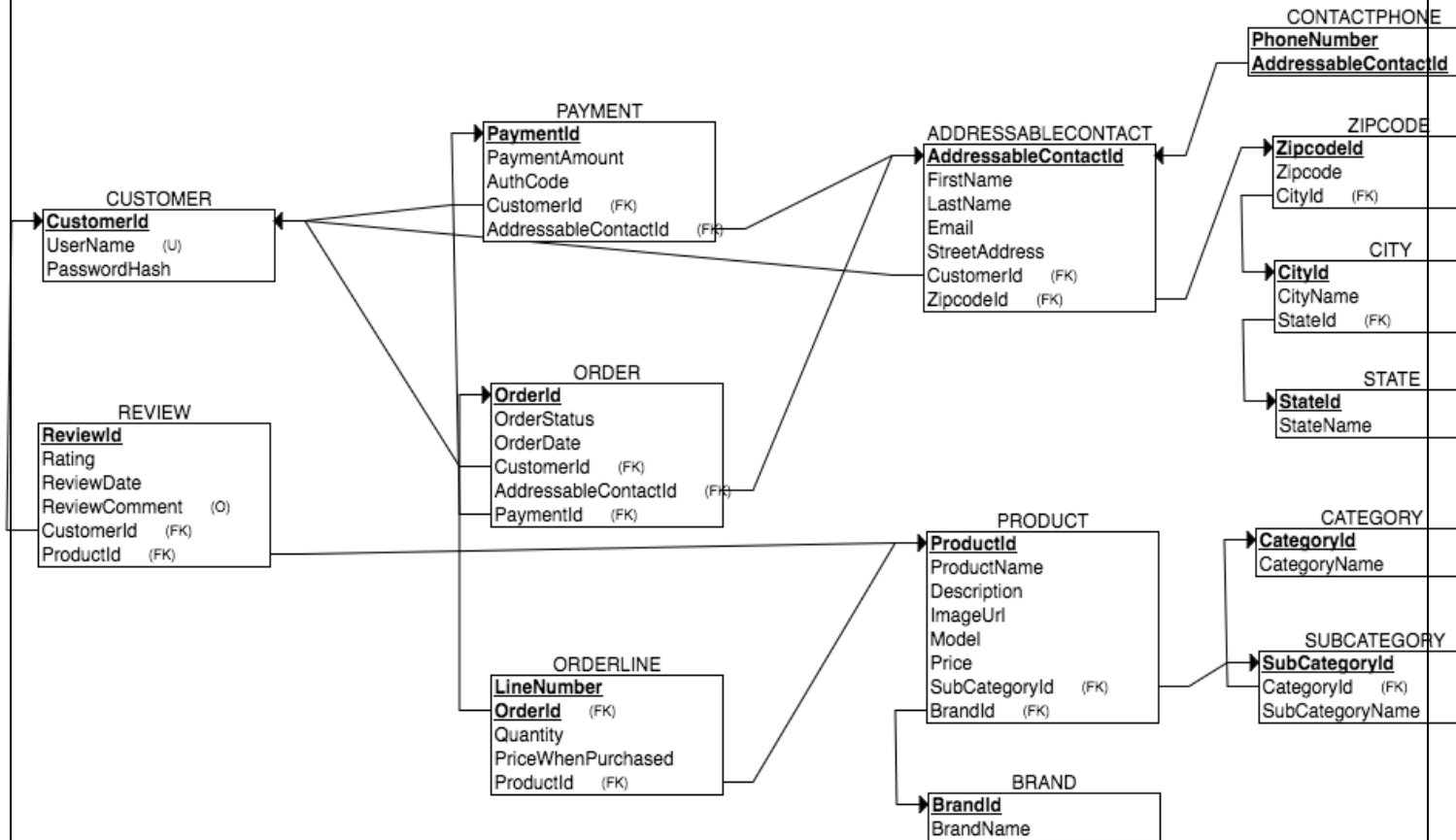
11.2.2 Fact Tables:

- **Sales**: This fact table will be used to store the total sales and total number of units sold. Linked with customer, order, product, location and calendar dimension tables.
- **Product Popularity**: It will contain the quantities of a product sold. Linked with product and calendar dimension tables.
- **Feedback**: It will contain the rating and review comments. Linked with customer, product, calendar and review.
- **Order History**: It is linked with customer, product and order fact table. It can be used for recommending the user based on his order history.
- **Average Rating**: It is linked to product and review tables. It will be used to get the average rating for a product.

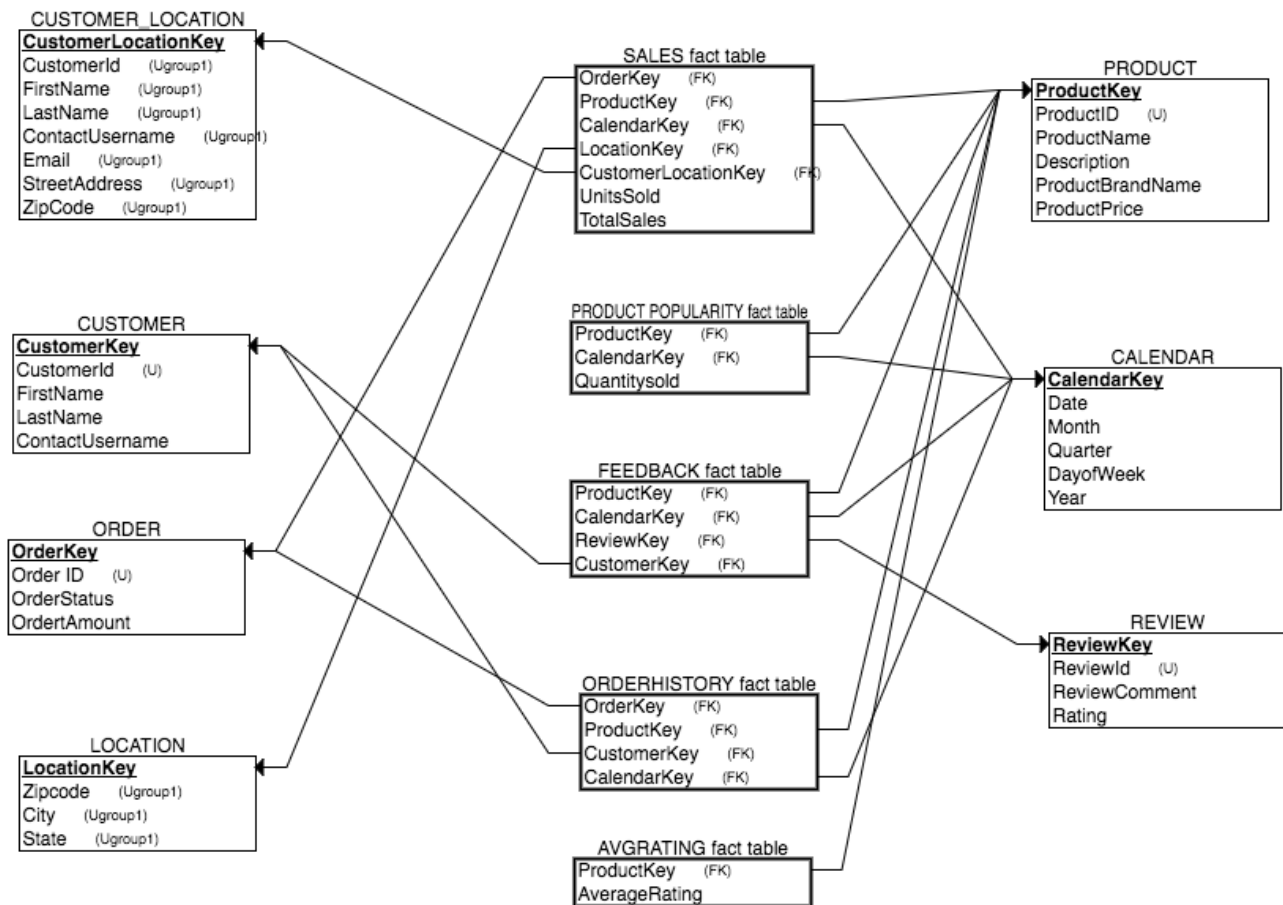
12. ER Diagram



13. Relational Schema



14. Star Schemas



15. Operational Queries:

We have performed various OLTP operations such as SELECT, INSERT, and UPDATE with our highly normalized tables. Most of these operations are driven by our html forms in our php powered web pages, which we'll detail next.

15.1 Web Page Forms

Customers use a login web page to access to our system. When the user submits the login form, we query the customer table to retrieve their record, so we can verify that their password matches the hashed value we have stored in our database. If so, we store their username in a server side session so that we can access it on other pages. Note that we store their password as a hash in the db - this is more secure.

Login

User Name	<input type="text"/>
Password	<input type="password"/>
<input type="button" value="Submit"/>	

[New User? Please sign-up here.](#)

15.1.1 Customer Registration:

New Customers can **Sign Up** by entering all the customer details, which gets inserted into different customer related tables customer, addressablecontact, contactphone, zipcode, city, and state. So when the form is submitted a new row gets inserted into most tables unless there's already an equivalent row - for example, maybe a row for the city of San Jose already exists, in which case we just reuse the existing id that corresponds to the San Jose row..

Customer Registration

First Name	<input type="text"/>
Last Name	<input type="text"/>
User Name	<input type="text"/>
Password	<input type="password"/>
Email Address	<input type="text"/>
Phone Number	<input type="text"/>
Street Address	<input type="text"/>
Zip Code	<input type="text"/>
City	<input type="text"/>
State	<input type="text"/>
<input type="button" value="Submit"/>	

15.1.2 Update Customer Information:

Once the update form is submitted we perform the update query to update the customer details for all the form data entered by a particular customer id. All the corresponding fields to the particular customer id gets updated in all the corresponding tables. The tables involved are customer, addressablecontact, zipcode, city, state, and contactphone.

Customer Information Update

First Name	<input type="text"/>
Last Name	<input type="text"/>
Password	<input type="text"/>
Email Address	<input type="text"/>
Phone Number	<input type="text"/>
Street Address	<input type="text"/>
Zip Code	<input type="text"/>
City	<input type="text"/>
State	<input type="text"/>
<input type="button" value="Submit"/>	

15.1.3 Order Lookup:

When the order search is performed the username stored in the session is used as one of the parameters to perform a query to retrieve the customer id. With this id we retrieve all the order details from both the `order` and `payment` tables. All orders are listed in the results:

Search Index Page	
Order ID	Order Lookup
51	Product Lookup
Back	Update Contact Info

15.1.4 Product Lookup:

Once the customer performs a product search a select query is performed to get details like the brandname, product name, description, model, price from a join between the `brand` and `product` tables using the customer provided “sub -category name”.

Enter the type of product (like Computers / Cell phones etc.):

notebooks

Search

[Back](#)

Brand Name	Product Name	Model	Description	Price	Select
Toshiba	Satellite 3000-400	PS300E-03YNL-DU	Toshiba Satellite 3000-400	9.60	<input type="radio"/>
Acer	TravelMate 740LF	91.42R01.32H	Acer TravelMate 740LF	6999.99	<input type="radio"/>
Toshiba	Portégé 7220CT / NW2	PP722E-HE390W-NL	Toshiba Portégé 7220CT / NW2	5.99	<input type="radio"/>
Acer	TravelMate 740LF	91.42R01.30U/KIT	Acer TravelMate 740LF	2.99	<input type="radio"/>
Toshiba	Satellite 2800-500	PS285E-00014-DU	Toshiba Satellite 2800-500	599.99	<input type="radio"/>
Toshiba	Satellite 1800-514	PS183E-004TV-DU	Toshiba Satellite 1800-514	4.99	<input type="radio"/>
Toshiba	Satellite Pro 4600 PIII800	PS460E-0CXX7-DU	Toshiba Satellite Pro 4600 PIII800	70.99	<input type="radio"/>
Toshiba	Portege 7220CT PIII650	PP722E-4390Q-NL	Toshiba Portege 7220CT PIII650	2999.00	<input type="radio"/>
Toshiba	Satellite Pro 4600 PIII900/128/20/14.1TFT DVD	PS460E-94NVT-DU	Toshiba Satellite Pro 4600 PIII900/128/20/14.1TFT DVD	8.99	<input type="radio"/>
Toshiba	Satellite Pro 4600 PIII900	PS460E-94NVX-NL	Toshiba Satellite Pro 4600 PIII900	40.99	<input type="radio"/>
Toshiba	Tecra 8200 PIII750	PT820E-05LVT-NL	Toshiba Tecra 8200 PIII750	8999.00	<input type="radio"/>
Toshiba	Tecra 8200-41EQP	PT820E-01EQP-NL	Toshiba Tecra 8200-41EQP	2.99	<input type="radio"/>
Toshiba	Satellite Pro 4600 PIII800	PS460E-018SF-DU	Toshiba Satellite Pro 4600 PIII800	9.99	<input type="radio"/>
Toshiba	Satellite Pro 4600 PIII800	PS460E-0CXXD-NL	Toshiba Satellite Pro 4600 PIII800	60.00	<input type="radio"/>
IBM	ThinkPad A22e	TA25ANU	IBM ThinkPad A22e	8.60	<input type="radio"/>
Toshiba	Satellite 1800-400	PS181E-00D59-DU	Toshiba Satellite 1800-400	20.99	<input type="radio"/>

15.1.5 Submit Review:

A customer can submit a **rating** and **review comment** for a particular product.

When the review form is submitted, we use the the username stored in the user's session to perform a query and get their customer id from the customer table. Then we insert a new row into the review table, using the customerid, and the other review fields submitted with the form post, such as the product id, rating(a number between 1 to 5), and their comment text.

Fujitsu	LIFEBOOK C-1020 CEL-1.8G	LKN:NDL-126100-037	Fujitsu LIFEBOOK C-1020 CEL-1.8G	30.99	<input type="radio"/>
Fujitsu	LIFEBOOK C-1020 CEL-1.8G	LKN:NDL-126100-040	Fujitsu LIFEBOOK C-1020 CEL-1.8G	9.99	<input type="radio"/>
Fujitsu	LIFEBOOK C-1020 CEL-1.8G	LKN:NDL-126100-036	Fujitsu LIFEBOOK C-1020 CEL-1.8G	3.98	<input type="radio"/>
Fujitsu	LIFEBOOK C-1020 CEL-1.8G	LKN:NDL-126100-039	Fujitsu LIFEBOOK C-1020 CEL-1.8G	4.99	<input type="radio"/>
Fujitsu	LIFEBOOK E7010 P4-1.7G	LKN:NDL-121150-003	Fujitsu LIFEBOOK E7010 P4-1.7G	1.99	<input type="radio"/>
Fujitsu	LIFEBOOK E7010 P4-1.7G	LKN:NDL-121150-006	Fujitsu LIFEBOOK E7010 P4-1.7G	8.99	<input type="radio"/>
Fujitsu	LIFEBOOK E7010 P4-1.7G	LKN:NDL-121150-010	Fujitsu LIFEBOOK E7010 P4-1.7G	7.99	<input type="radio"/>
Fujitsu	LIFEBOOK E7010 P4-1.7G	LKN:NDL-121150-013	Fujitsu LIFEBOOK E7010 P4-1.7G	4.00	<input type="radio"/>
Fujitsu	LIFEBOOK E7010 P4-1.7G	LKN:NDL-121150-017	Fujitsu LIFEBOOK E7010 P4-1.7G	10.99	<input type="radio"/>
Fujitsu	LIFEBOOK E7010 P4-1.7G	LKN:NDL-121150-020	Fujitsu LIFEBOOK E7010 P4-1.7G	799.99	<input type="radio"/>
Fujitsu	LIFEBOOK E7010 P4-1.7G	LKN:NDL-121150-004	Fujitsu LIFEBOOK E7010 P4-1.7G	9.99	<input type="radio"/>
Fujitsu	LIFEBOOK E7010 P4-1.7G	LKN:NDL-121150-011	Fujitsu LIFEBOOK E7010 P4-1.7G	4.60	<input type="radio"/>
Fujitsu	LIFEBOOK E7110 P4-1.7G	LKN:NDL-121170-005	Fujitsu LIFEBOOK E7110 P4-1.7G	10.99	<input type="radio"/>

Submit a review

[Back](#)

© 2016eKart.com . All rights reserved.

Review

Submit Review

[Back](#)

© 2016eKart.com . All rights reserved.

16. Analytical Queries

We are using a tool name Tableau to visualize the analytical results. Tableau will be connected to our Native MySQL server and will be pulling the dimension, analytical tables and their data from the SQL server.

Tableau gave us great flexibility to view the results without hard coding a specific column field. Thus, many queries can be merged together and analyzed.

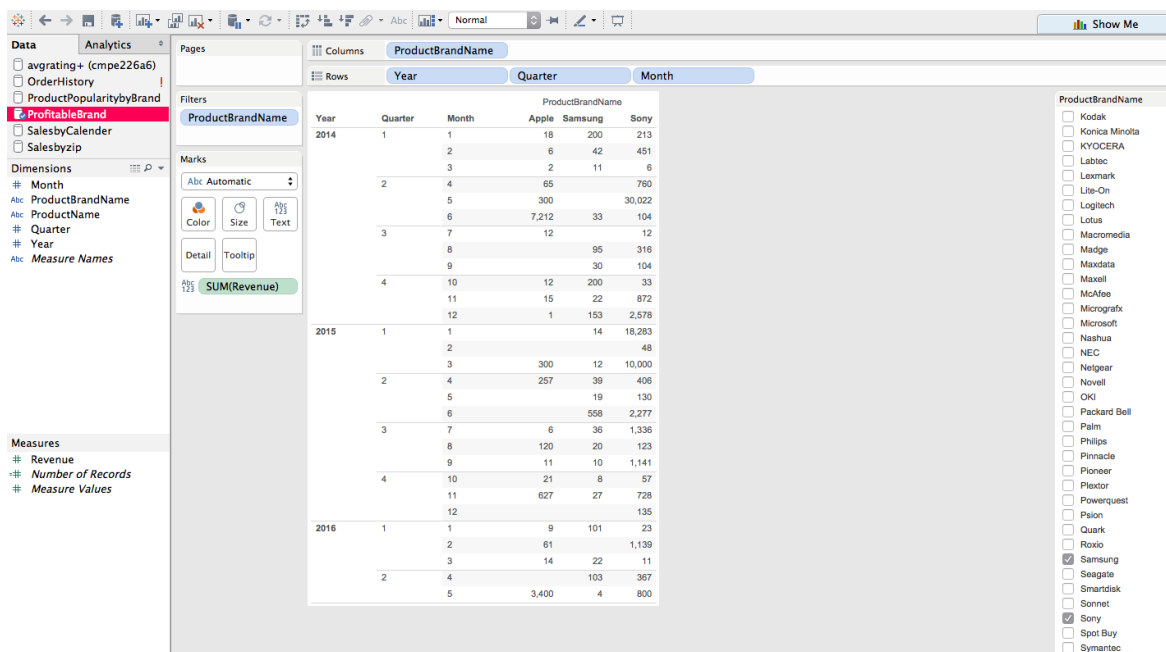
16.1 Finding the most profitable brand.

This query can help in analyzing the total revenue generated by a brand (which is the sum of its total sales) over a period of time. We can analyze this for some particular year (may be 2010 or 2011) or over some month/quarter (could be a festive month/quarter).

The output is displayed for all the brands. Total revenue for one brand (say apple) can also be seen by applying a Tableau filter on the product brand name, which essentially just adds an extra condition to the where clause for us.

SQL Query:

```
select ProductName, ProductBrandName, Year, Quarter,Month, SUM(TotalSales) as
Revenue
from PRODUCT A, SALES B, CALENDAR C
WHERE A.ProductKey = B.ProductKey and B.CalendarKey = C.CalendarKey
group by 1,2,3,4,5
ORDER BY 6 DESC
```



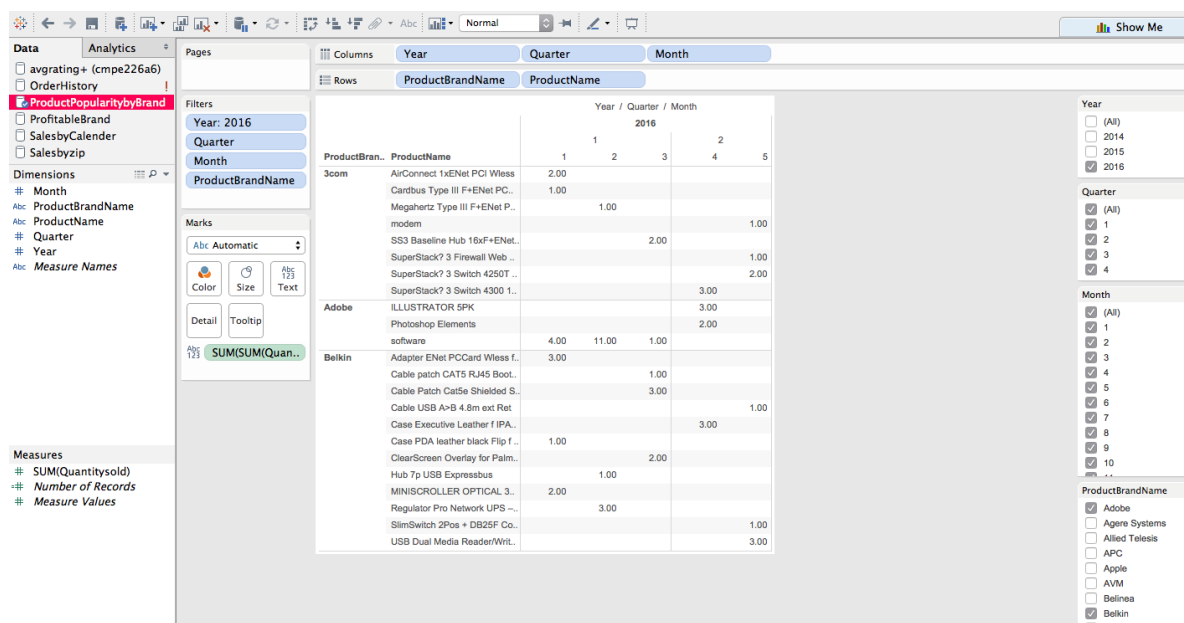
16.2 Product Popularity By Brand

This query is used to visualize which product is most popular product within a brand (computed by total units sold for a product) over a period of time or during some time frame. We can analyze this for some particular year (may be 2010 or 2011) or over some month/quarter (could be a festive month/quarter).

Hence, brands can find which product (Say, Apple iphone) is most popular or least popular (say, 11" tablet by Samsung) and the brand can apply some marketing strategy to improve or maintain the current numbers!

SQL Query:

```
select ProductName, ProductBrandName, Year, Quarter, Month, SUM(Quantitysold)
from PRODUCT A, PRODUCT_POPULARITY B, CALENDAR C
WHERE A.ProductKey = B.ProductKey and B.CalendarKey =C.CalendarKey
group by 1,2,3,4,5
```



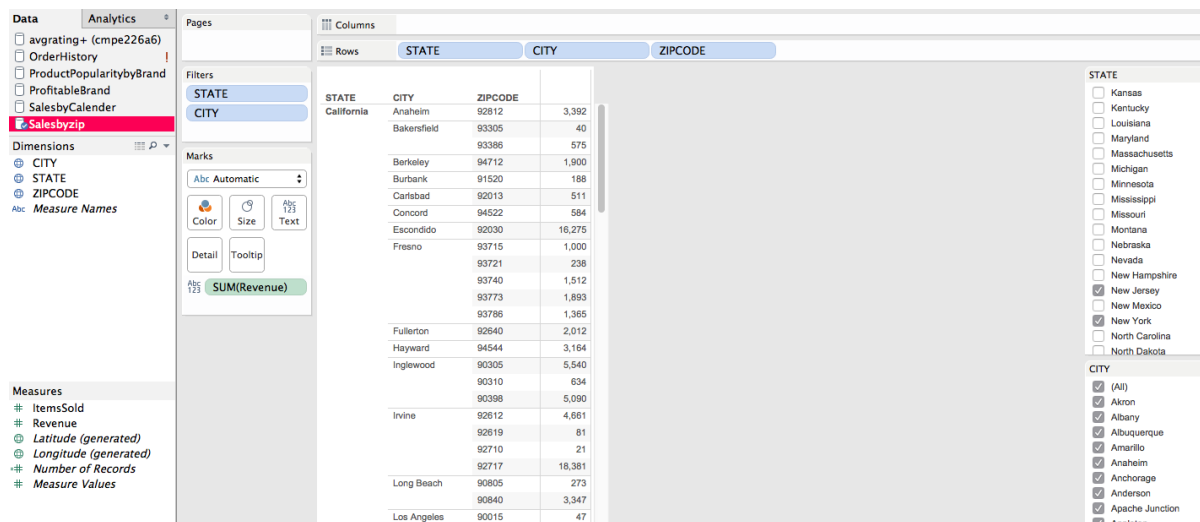
16.3 Total Sales by Zip code

This query will display the total sales and items delivered to the zip code for the city. It displays the data for all the cities including all the zip codes falling in that city. Filters can be applied to see the city receiving maximum number of orders or the zip code within a city from where least number of orders is coming.

It can be used by the ecommerce website to roll out some offers (may be free home delivery etc.) in the areas from where not much orders are coming. It can also be used to roll out some promotions in the city with maximum number of orders and revenue.

SQL Query:

```
select STATE,CITY,ZIPCODE, SUM(TotalSales) as Revenue, SUM(UnitsSold) as ItemsSold
from SALES, LOCATION
WHERE LOCATION.LocationKey = SALES.LocationKey
group by 1,2,3
```



16.4 Sales By Calendar

This query will display the sales made over some period of time. Time period can be selected as a year, month or quarter by applying the filters (where clause in SQL). This can be helpful in analyzing the most profitable time where sales were very high (say during a new product launch or during holidays).

SQL Query:

```
SELECT YEAR,QUARTER,MONTH,SUM(UNITSSOLD) AS UNITS_SOLD,SUM(TOTALSALES) AS REVENUE
FROM SALES A, CALENDAR B
WHERE A.CalendarKey = B.CalendarKey
GROUP BY 1,2,3
```

Data | Analytics

avgrating+ (cmpe226a6)
OrderHistory
ProductPopularitybyBrand
ProfitableBrand
SalesbyCalender
Salesbyzip

Dimensions

- MONTH
- QUARTER
- YEAR
- Measure Names

Measures

- REVENUE
- UNITS_SOLD
- Number of Records
- Measure Values

Columns | YEAR

Rows | QUARTER | MONTH

QUARTER	MONTH	YEAR		
		2014	2015	2016
1	1	72,181	116,227	85,779
	2	60,423	63,681	52,895
	3	104,571	81,923	66,965
2	4	107,961	227,851	100,114
	5	137,976	177,725	84,871
	6	44,668	103,217	2,398
3	7	70,942	141,230	
	8	90,999	139,238	
	9	143,513	94,804	
4	10	99,443	166,999	
	11	116,662	78,079	
	12	124,651	118,826	

16.5 Average rating for a product

This query will display the average rating for a product based on the ratings received till current date. This can be very useful to a customer to make choice by comparing different products while shopping.

SQL Query:

```
SELECT A.PRODUCTKEY,B.PRODUCTID,B.PRODUCTNAME, B.PRODUCTBRANDNAME, A.AVERAGERATING
FROM AVGRATING A, PRODUCT B
WHERE A.PRODUCTKEY = B.PRODUCTKEY
```

The screenshot shows the Tableau Desktop interface. On the left, the 'Data' pane shows a table named 'avgRating' (cmpe226a6) with dimensions: PRODUCTBRANDNAME, PRODUCTID, and PRODUCTNAME. The 'Columns' shelf contains these three dimensions. The main view displays a table of products with columns: PRODUCTBRANDNAME, PRODUCTID, and PRODUCTNAME. The table lists various products from brands like Acer, Apple, and Samsung. On the right, the 'Filters' pane shows a list of product names with checkboxes for selection.

17. OLAP Queries

17.1 Drill Up

Total Sales and Items sold can be analyzed at different levels of aggregation from zip code to city or state level.

City level aggregation:

SQL Query:

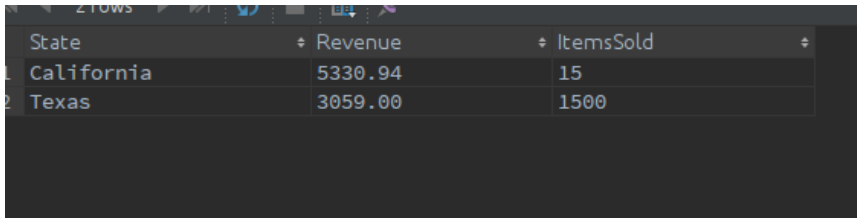
```
select State, City, SUM(TotalSales) as Revenue, SUM(UnitsSold) as ItemsSold
from SALES, LOCATION
WHERE LOCATION.LocationKey = SALES.LocationKey
group by 1, 2
ORDER BY 3 DESC,4 DESC;
```

	State	City	Revenue	ItemsSold
1	California	San Carlos	2099.97	5
2	California	San Jose	1899.97	3
3	Texas	Houston	1689.00	350
4	California	Sunnyvale	1220.00	6
5	Texas	San Antonio	650.00	500
6	Texas	Dallas	500.00	100
7	Texas	Austin	220.00	550
8	California	Menlo Park	111.00	1

State Level aggregation:

SQL Query:

```
select State, SUM(TotalSales) as Revenue, SUM(UnitsSold) as ItemsSold
from SALES, LOCATION
WHERE LOCATION.LocationKey = SALES.LocationKey
group by 1
ORDER BY 2 DESC,3 DESC;
```

A screenshot of a database query result showing state-level aggregation. The table has three columns: State, Revenue, and ItemsSold. The results are ordered by Revenue in descending order.

State	Revenue	ItemsSold
California	5330.94	15
Texas	3059.00	1500

17.2 Drill Down:

Revenue generated and items sold can be drilled down to view at a detailed level like zipcode.

SQL Query:

```
select State, City, Zipcode, SUM(TotalSales) as Revenue, SUM(UnitsSold) as
ItemsSold
from SALES, LOCATION
WHERE LOCATION.LocationKey = SALES.LocationKey
group by 1,2,3
ORDER BY 3 DESC,4 DESC;
```

Zipcode	city	state	Revenue	ItemsSold
94070	San Carlos	California	2099.97	5
95101	San Jose	California	1899.97	3
94080	sunnyvale	California	900.00	3
94081	Sunnyvale	California	200.00	2
94086	Sunnyvale	California	120.00	1
94025	Menlo Park	California	111.00	1

17.3 Slice:

A deep-dive analysis can be performed on a particular product or brand or city level. Also, we can slice the data on the time frame of analysis - like displaying sales for a given year.

SQL Query:

```
select ProductName, ProductBrandName, Year, Quarter, SUM(Quantitysold)
from PRODUCT A, PRODUCT_POPULARITY B, CALENDAR C
WHERE A.ProductKey = B.ProductKey and B.CalendarKey = C.CalendarKey
group by 1,2,3,4
ORDER BY 5 DESC;
```

ProductName	ProductBrandName	Year	Quarter	SUM(Quantitysold)
civic	Honda	2011	1	10
Ipad	Apple	2010	1	7
Ipod	Apple	2010	1	7
accord	Honda	2011	1	5
Crossover	Ford	2016	1	4
Crossover	Ford	2011	2	4
s2000	Honda	2016	1	3
s2000	Honda	2011	2	3
iPhone 6s 32gb	Apple	2016	1	2
iPhone 6s 32gb	Apple	2011	2	2
galaxy s7	Samsung	2016	1	1
galaxy s7	Samsung	2011	2	1

SQL Query for products sold in 2010:

```
select ProductName, ProductBrandName, SUM(Quantitysold)
from PRODUCT A, PRODUCT_POPULARITY B, CALENDAR C
WHERE A.ProductKey = B.ProductKey and B.CalendarKey = C.CalendarKey and
YEAR = 2010 group by 1,2
ORDER BY 3 DESC;
```

ProductName	ProductBrandName	SUM(Quantitysold)
Ipad	Apple	7
Ipod	Apple	7

17.4 Dice:

Data can be diced by applying multiple filters for analytical purposes. For e.g below query displays items sold by Honda in first quarter of 2011.

SQL Query:

```
select ProductName, ProductBrandName, SUM(Quantitysold)
from PRODUCT A, PRODUCT_POPULARITY B, CALENDAR C
WHERE A.ProductKey = B.ProductKey and B.CalendarKey = C.CalendarKey and
```

```
YEAR = 2011 and Quarter = 1 and ProductBrandName like '%honda%' group by 1,2  
ORDER BY 3 DESC;
```

ProductName	ProductBrandName	SUM(Quantitysold)
civic	Honda	10
accord	Honda	5

18. Enhancements Planned

- ✓ Backup of the data by writing a script.
- ✓ Data archival for the records which are more than 5 years old into some storage media or data warehouse. It can be used for data analysis purpose.
- ✓ We can store data related to the customer search history and then can do behavioral analysis based on that.
- ✓ Review comments can be used to do the sentimental analysis.