

Database Project Report

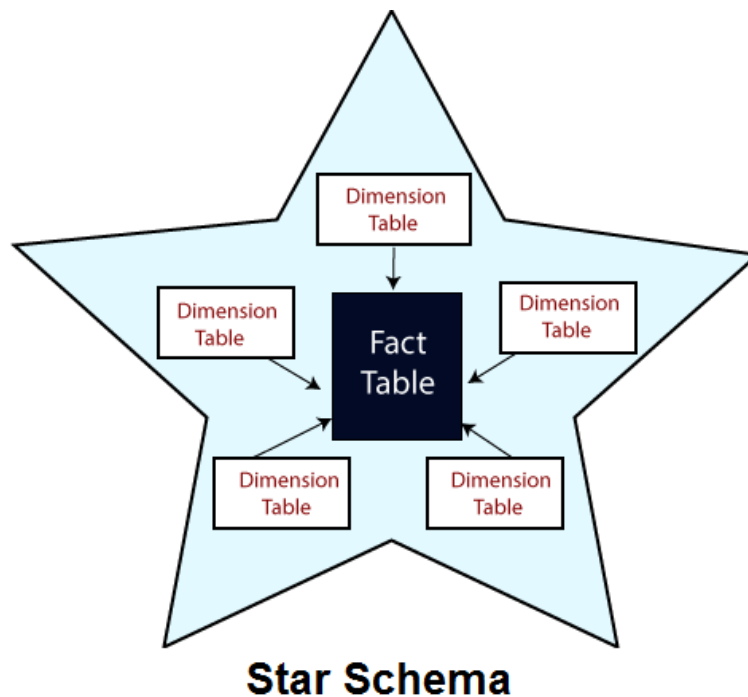
Phan Manh Tung, Louis Choules

Topic: OLAP Data warehouse using star schema for reporting and analyzing purposes.

I. Introduction

OLAP, or Online Analytical Processing, is a methodology that enables multidimensional analysis of data to provide powerful insights and support business decisions. OLAP is commonly used in data warehousing projects, where large amounts of data are collected from various sources and stored in a central repository for analysis.

One of the key components of an OLAP system is the Star Schema database model. In contrast to traditional, normalized database models, the Star Schema is designed specifically for OLAP analysis. It organizes data into a central fact table surrounded by dimension tables, creating a star-like structure. This allows for easy querying and analysis of data using OLAP tools.



One important aspect of the Star Schema is the denormalized structure. Compared to a normal, normalized database model, the Star Schema offers several advantages. Firstly, it is optimized for OLAP analysis and provides faster query performance. Secondly, it simplifies data modeling, making it easier to understand and maintain. Thirdly, it allows for flexible and dynamic analysis of data, enabling users to slice and dice data in various ways to gain more insights.

In the context of e-commerce, the Star Schema is particularly useful for analyzing sales data, customer behavior, and product performance. With an OLAP Data Warehouse project for e-commerce, businesses can gain a competitive edge by quickly identifying trends and opportunities in the market, and making informed decisions based on data-driven insights. For this project, we intend to build a OLAP Data Warehouse specifically for an e-commerce business, analysing all transactions and customers.

II. OLAP Principles

As previously mentioned, the Star Schema has a denormalized structure, which is different from normal transactional database. Therefore, we will list below some key bullet principles to follow when designing a OLAP Data Warehouse, mainly on Fact tables and Dimensions tables:

- Fact tables should contain numeric data, also known as measures, such as sales revenue, quantity sold, or profit margin.
- Fact tables should have a foreign key that links to one or more dimension tables, creating a relationship between the measures and the descriptive attributes.
- Fact table should have a "grain", which refers to the level of detail or granularity at which the fact data is captured and stored in the table. The grain of a fact table is determined by the intersection of its associated dimensions, and represents the lowest level of detail at which the measures in the fact table are relevant.
- Fact tables should not contain any descriptive attributes, but rather contain numeric data that can be aggregated by the attributes in dimension tables.
- Fact tables should be optimized for query performance, including appropriate indexing and partitioning strategies.
- Dimension tables should contain descriptive attributes that describe the characteristics of a business entity, such as product, customer, or time.
- Dimension tables should have a primary key that uniquely identifies each dimension record.
- Dimension tables should not contain any measures or numeric data, but rather contain descriptive attributes that can be used to filter, group, or aggregate the measures in fact tables.
- Dimension tables should be denormalized to reduce the number of joins required for queries.

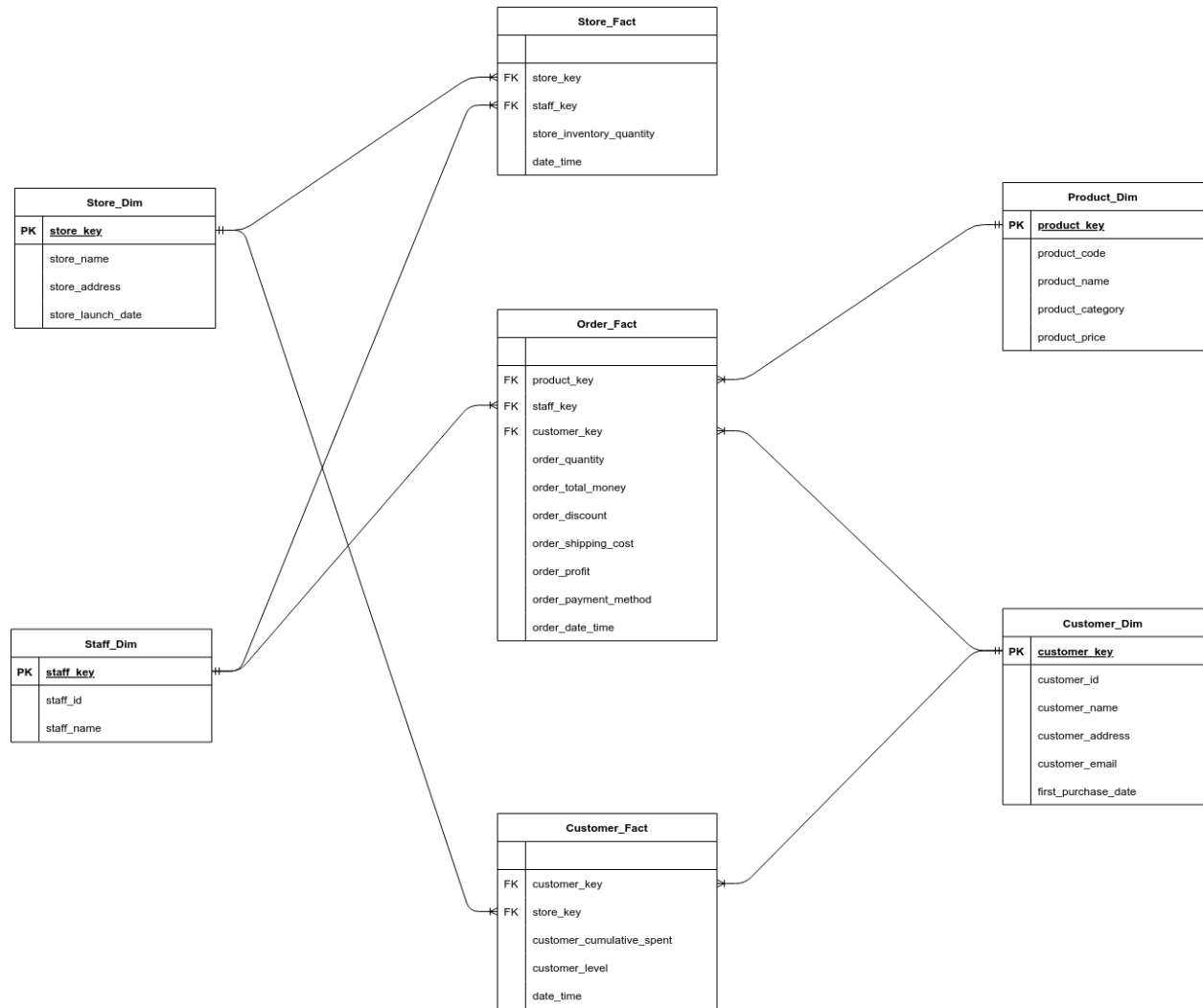
- Measures should be additive, meaning they can be summed across dimensions, or non-additive, meaning they cannot be summed across dimensions.
- Measures should be expressed using appropriate units of measure, such as currency, time, or percentage.

Overall, the design of a Star Schema OLAP should prioritize simplicity, efficiency, and ease of use, enabling users to quickly and easily query and analyze data to gain insights and make informed decisions for the business activities.

III. Implementation

1. Designed schema

The first step is to design the schema for the intended OLAP database. Our ultimate version contains 3 Fact tables and 4 Dimension tables, strictly following all the key principles listed above.



There are a few observations of the schema:

- Dimension tables contain self-explained descriptive attributes.
- Fact tables should contain numeric data, some measures such as total money, quantity, and profit in the Order Fact.
- One aggregated field name “cumulative spent” in the Customer Fact.
- Fact tables contain several foreign keys linking to various Dimension tables.
- Dimension tables contain some redundancy such as store_address and customer_address.

We clearly define the “grain” of each Fact table as follows:

- Store_Fact: the up-to-date status of the store (inventory quantity, the staff making the update).
- Order_Fact: the details of each order (datetime of the order, which product, which customer, in-charge staff, quantity, total money, discount).
- Customer_Fact: the up-to-date status of the customer (cumulative sum of money spent in each store, current level).

2. Imported data

In order to mimic the real-life database, we decide to use the open-source an E-Commerce dataset for the order_fact table, also the data for real names locations for the descriptive attributes in staff_dim, customer_dim and store_dim tables. All are found on Kaggle. We manipulate the provided data using Python pandas to create appropriate mock data into 7 separate .csv files.

The results obtained as follows:

	order_fact	store_fact	customer_fact	customer_dim	store_dim	staff_dim	product_dim
# rows	51290	10543	51233	38997	298	8764	42

Example for the order_fact table:

	product_key	staff_key	customer_key	order_quantity	order_total_money	order_discount	order_shipping_cost	order_profit	order_payment_method	order_date_time
0	0	83	108	1.0	102.6	0.3	4.6	46.0	credit_card	2018-01-02 10:56:33
1	1	8265	21853	1.0	158.9	0.3	11.2	112.0	credit_card	2018-07-24 20:41:37
2	2	1120	33714	5.0	529.6	0.1	3.1	31.2	credit_card	2018-11-08 08:38:49
3	3	8358	8606	1.0	85.2	0.3	2.6	26.2	credit_card	2018-04-18 19:28:06
4	4	6128	24314	1.0	191.0	0.3	16.0	160.0	credit_card	2018-08-13 21:18:39
...
51285	39	7728	4479	4.0	349.1	0.3	1.9	19.2	money_order	2018-02-28 22:59:50
51286	40	3933	4467	5.0	281.4	0.2	1.4	14.0	credit_card	2018-02-28 13:19:25
51287	41	4022	4489	1.0	97.1	0.3	4.0	39.7	credit_card	2018-02-28 10:25:07
51288	32	6522	4475	1.0	186.0	0.2	13.2	131.7	credit_card	2018-02-28 10:50:08
51289	33	3766	4586	5.0	748.4	0.3	9.9	99.4	credit_card	2018-02-28 11:09:40

51290 rows × 10 columns

We then feed the .csv files into the database: PHP SQL database and Sqlite3 my_data.db

3. Querying the database

Question 1: What product most sells throughout the year: TOP 10?

Python SQLite3:

```
import sqlite3
import pandas as pd

con =
sqlite3.Connection('/content/gdrive/MyDrive/database_project/my_data.db')

# What product most sell throughout the year: TOP 10
query = """
SELECT product_name, product_category, SUM(order_quantity) AS
total_sales_quantity
FROM product_dim A
JOIN order_fact B ON A.product_key=B.product_key
GROUP BY product_name
ORDER BY total_sales_quantity DESC
LIMIT 10;
"""
observations = pd.read_sql(query, con)
observations
```

	product_name	product_category	total_sales_quantity
0	Titak watch	Fashion	6254.0
1	Formal Shoes	Fashion	6154.0
2	Sports Wear	Fashion	6093.0
3	Running Shoes	Fashion	6064.0
4	Fossil Watch	Fashion	6050.0
5	Sneakers	Fashion	6049.0
6	Casula Shoes	Fashion	6035.0
7	Shirts	Fashion	6012.0
8	Suits	Fashion	5996.0
9	T - Shirts	Fashion	5986.0

Question 2: What product sells the most throughout the year? - TOP 3 for each category

```

# What product most sell throughout the year: TOP 3 for each category
query = """
SELECT product_name, product_category, total_sales_quantity
FROM (SELECT product_name, product_category, total_sales_quantity, RANK()
      OVER (PARTITION BY product_category ORDER BY total_sales_quantity
            DESC) AS rank

      FROM (SELECT product_name, product_category, SUM(order_quantity) AS
total_sales_quantity
      FROM product_dim A
      JOIN order_fact B ON A.product_key=B.product_key
      GROUP BY product_name, product_category)

      GROUP BY product_name, product_category)

WHERE rank <= 3
"""
observations = pd.read_sql(query, con)
observations

```

Result:

	product_name	product_category	total_sales_quantity
0	Car Body Covers	Auto & Accessories	2040.0
1	Tyre	Auto & Accessories	2023.0
2	Car Pillow & Neck Rest	Auto & Accessories	2013.0
3	Speakers	Electronic	581.0
4	Fans	Electronic	523.0
5	Samsung Mobile	Electronic	501.0
6	Titak watch	Fashion	6254.0
7	Formal Shoes	Fashion	6154.0
8	Sports Wear	Fashion	6093.0
9	Beds	Home & Furniture	3908.0
10	Dinning Tables	Home & Furniture	3874.0
11	Sofa Covers	Home & Furniture	3852.0

Question 3: Best sellers of March : TOP 10 - Highest order total money

```
import pandas as pd
# 1> Highest order total money

query = """
SELECT staff_name, staff_id, SUM(order_total_money) AS total_sales_money
FROM staff_dim A
JOIN order_fact B ON A.staff_key=B.staff_key
WHERE date(B.order_date_time) BETWEEN '2018-03-01' AND '2018-03-31'
GROUP BY staff_id
ORDER BY total_sales_money DESC
LIMIT 10;
"""
observations = pd.read_sql(query, con)
observations
```

Result:

	staff_name	staff_id	total_sales_money
0	Paul Crowley	41347	1971.6
1	Denia Caballero	43678	1648.2
2	Marius Sabaliauskas	39982	1590.7
3	Maura Hopkins	44102	1533.9
4	Simon Eldershaw	43975	1490.5
5	Wolfgang Larrazabal	43822	1430.3
6	Dejan Drakul	43004	1422.2
7	Nele Jansegers	44445	1356.7
8	John Dooley	39797	1355.2
9	John Harris	43468	1253.7

Question 4: Best sellers of March : TOP 10 - Highest number of deals

```
query = """
SELECT staff_name, staff_id, COUNT(*) AS number_of_deals
```



```

FROM staff_dim A
JOIN order_fact B ON A.staff_key=B.staff_key
WHERE date(B.order_date_time) BETWEEN '2018-03-01' AND '2018-03-31'
GROUP BY staff_id
ORDER BY number_of_deals DESC
LIMIT 10;
"""
observations = pd.read_sql(query, con)
observations

```

Result:

	staff_name	staff_id	number_of_deals
0	Harriett Baldwin	45447	4
1	Paul Crowley	41347	4
2	Ron Kee	47654	3
3	Yoad Nevo	47413	3
4	Uwe Windhorst	46855	3
5	Colin Cunningham	46385	3
6	Hamad Al-Sagoor	46238	3
7	Byron Hayward	45839	3
8	Čaba Silađi	45303	3
9	Anth Smith	45270	3

Question 5: Stores with lowest monthly inventory during 2018, provided the date

```

query = """
SELECT store_name, store_address, MAX(store_inventory_quantity) AS
lowest_monthly_inventory, date(date_time) AS date
FROM store_dim A
JOIN store_fact B ON A.store_key=B.store_key
WHERE date BETWEEN '2018-01-01' AND '2018-12-31'
GROUP BY store_name
ORDER BY lowest_monthly_inventory ASC
LIMIT 10;
"""
observations = pd.read_sql(query, con)
observations

```

	store_name	store_address	lowest_monthly_inventory	date
0	TOMPKINSVILLE	TOMPKINSVILLE, KY, USA	1400	2018-07-01
1	BOULDER CITY	BOULDER CITY, NV, USA	1500	2018-01-01
2	CREVE COEUR	CREVE COEUR, MO, USA	1600	2018-05-01
3	EAU CLAIRE	EAU CLAIRE, WI, USA	1600	2018-05-01
4	GREENCASTLE	GREENCASTLE, IN, USA	1600	2018-01-01
5	HILLSBORO	HILLSBORO, OH, USA	1600	2018-04-01
6	LYNN	LYNN, MA, USA	1600	2018-02-01
7	MAHNOMEN	MAHNOMEN, MN, USA	1600	2018-02-01
8	SUN CITY WEST	SUN CITY WEST, AZ, USA	1600	2018-05-01
9	ALPINE	ALPINE, TX, USA	1700	2018-03-01

Question 6: Total sales and profit per month

```
import pandas as pd
# 1> Stores with highest inventory

query = """
SELECT strftime('%m', order_date_time) as Month, SUM(order_total_money) AS
order_total_money, SUM(order_profit) AS order_profit
FROM order_fact
GROUP BY Month;
"""

observations = pd.read_sql(query, con)
observations
```

	Month	order_total_money	order_profit
0	01	680473.1	174573.6
1	02	595931.8	153288.2
2	03	759327.3	200936.8
3	04	1075123.0	277832.2
4	05	1466617.5	379386.3
5	06	1138506.8	298300.1
6	07	1429113.3	374391.6
7	08	1180375.0	306904.0
8	09	1298955.8	341558.1
9	10	1336695.8	342368.5
10	11	1588751.8	406808.7
11	12	1362317.0	354838.5

Visualization:

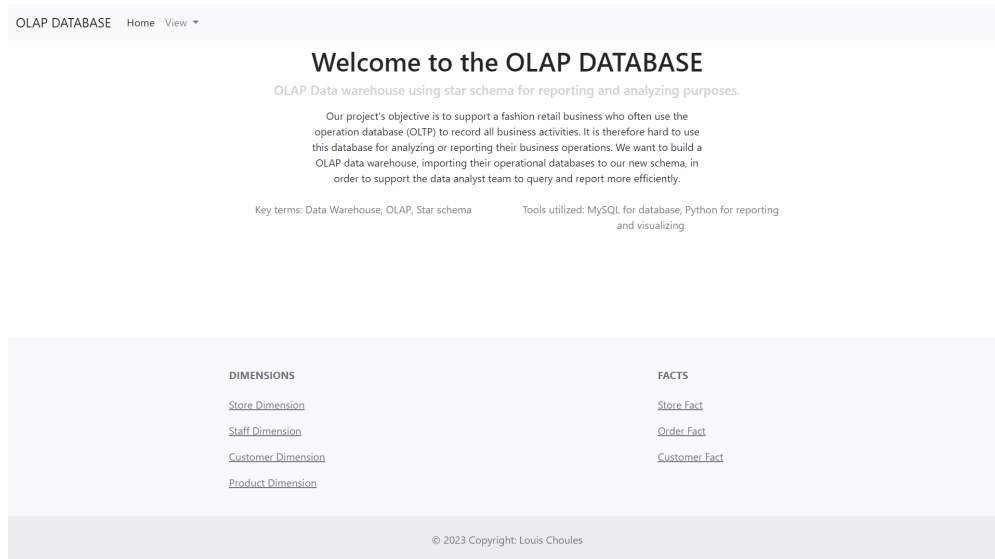
```
observations.plot(x="Month", y=['order_total_money', 'order_profit'])
```



4. Website

Dashboard:

We have an brief introduction for the website and navigation for most of the pages in the application. Each page will have navigation bar and also footer.



View Data:

All the information displayed here is paginated by 25 data. For dimension page we will have some additional action where we can add a new data, update the data and also delete the data.

- Store Dimension
- Staff Dimension
- Customer Dimension
- Product Dimension

The 'Store Dimension' page has a navigation bar at the top with 'OLAP DATABASE', 'Home', and 'View' (with a dropdown arrow). The main heading is 'Store Dimension' with an 'Add' button (with a plus icon) to its right. Below the heading is a table with the following data:

Store Key	Store Name	Store Address	Store Launch Date	Action
1	WASHINGTON	WASHINGTON, GA, USA	2017-03-16	Edit Delete
2	ARTESIA	ARTESIA, NM, USA	2015-12-21	Edit Delete
3	NOCONA	NOCONA, TX, USA	2016-03-16	Edit Delete
4	ROME	ROME, NY, USA	2017-06-03	Edit Delete
5	DECORAH	DECORAH, IA, USA	2017-10-18	Edit Delete
6	NORTH KANSAS CITY	NORTH KANSAS CITY, MO, USA	2015-04-27	Edit Delete
7	KENEDY	KENEDY, TX, USA	2014-03-29	Edit Delete
8	VALPARAISO	VALPARAISO, IN, USA	2016-09-29	Edit Delete
9	BLAKELY	BLAKELY, GA, USA	2017-09-17	Edit Delete
10	LAFAYETTE	LAFAYETTE, IN, USA	2016-01-23	Edit Delete
11	SULPHUR SPRINGS	SULPHUR SPRINGS, TX, USA	2015-10-26	Edit Delete
12	AHOSKIE	AHOSKIE, NC, USA	2014-02-22	Edit Delete

Example for Store Dimension page

For the fact page, we only able to see the information in it from the database.

- Store Fact
- Order Fact
- Customer Fact

OLAP DATABASE Home View

Store Fact			
Store Key - Store Name	Staff Key - Staff Name	Store Inventory Quantity	Date Time
1 - WASHINGTON	5727 - Isabel Luisa	1200	2017-04-01
1 - WASHINGTON	5727 - Isabel Luisa	1900	2017-05-01
1 - WASHINGTON	5727 - Isabel Luisa	900	2017-06-01
1 - WASHINGTON	6809 - Vicki Benckert	1400	2017-07-01
1 - WASHINGTON	8079 - Christina Alessi	1700	2017-08-01
1 - WASHINGTON	8079 - Christina Alessi	1000	2017-09-01
1 - WASHINGTON	8079 - Christina Alessi	1800	2017-10-01
1 - WASHINGTON	6809 - Vicki Benckert	500	2017-11-01
1 - WASHINGTON	8079 - Christina Alessi	1600	2017-12-01
1 - WASHINGTON	6809 - Vicki Benckert	1900	2018-01-01
1 - WASHINGTON	6809 - Vicki Benckert	800	2018-02-01
1 - WASHINGTON	5727 - Isabel Luisa	600	2018-03-01
1 - WASHINGTON	5727 - Isabel Luisa	2000	2018-04-01
1 - WASHINGTON	5727 - Isabel Luisa	600	2018-05-01
1 - WASHINGTON	8079 - Christina Alessi	900	2018-06-01
1 - WASHINGTON	5727 - Isabel Luisa	500	2018-07-01
1 - WASHINGTON	8079 - Christina Alessi	700	2018-08-01

Example for Store Fact page

Add:

We can add some new information related to specific table as listed as below:

- Store Dimension
- Staff Dimension
- Customer Dimension
- Product Dimension

OLAP DATABASE Home View

Insert Store Dimension

Store Name

Store Address

Store Launch Date

mn/dd/yyyy

Submit

DIMENSIONS

[Store Dimension](#)
[Staff Dimension](#)
[Customer Dimension](#)
[Product Dimension](#)

FACTS

[Store Fact](#)
[Order Fact](#)
[Customer Fact](#)

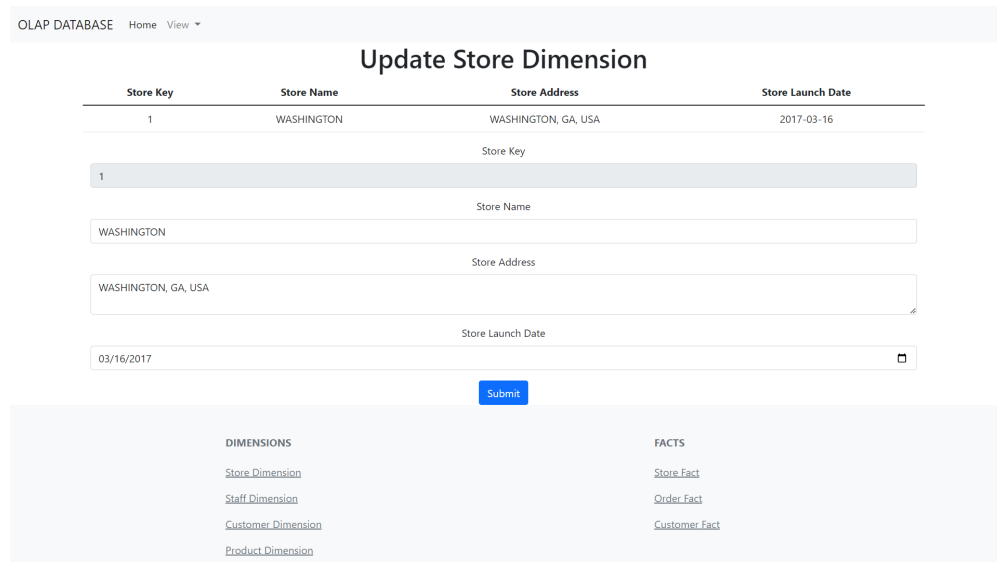
© 2023 Copyright: Louis Choules

Example for Insert Store Dimension page

Update:

It will display the old information record and we can modify the information about it to specific table as listed as below:

- Store Dimension
- Staff Dimension
- Customer Dimension
- Product Dimension



OLAP DATABASE Home View ▾

Update Store Dimension

Store Key	Store Name	Store Address	Store Launch Date
1	WASHINGTON	WASHINGTON, GA, USA	2017-03-16

Store Key

1

Store Name

WASHINGTON

Store Address

WASHINGTON, GA, USA

Store Launch Date

03/16/2017

Submit

DIMENSIONS

- [Store Dimension](#)
- [Staff Dimension](#)
- [Customer Dimension](#)
- [Product Dimension](#)

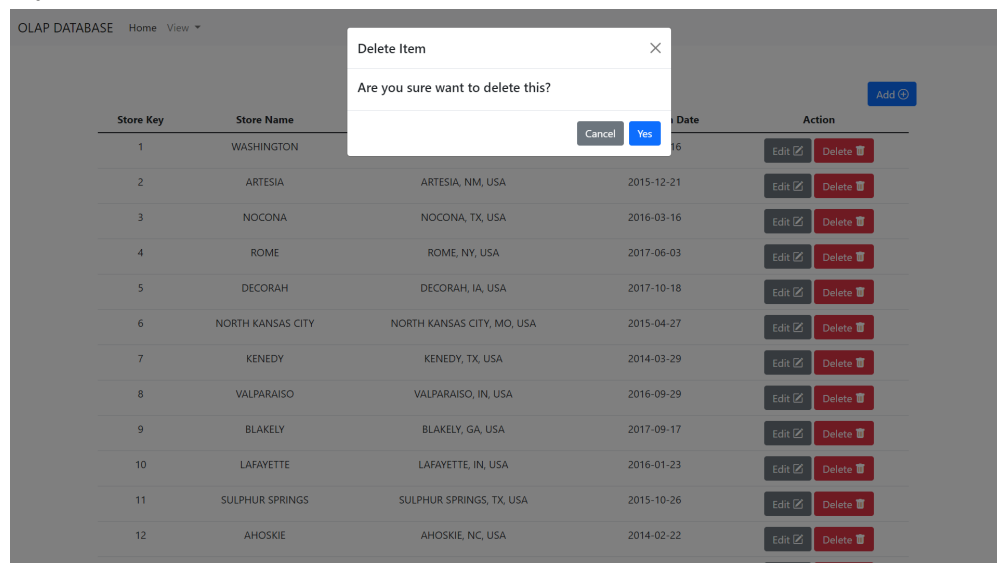
FACTS

- [Store Fact](#)
- [Order Fact](#)
- [Customer Fact](#)

Example for Update Store Dimension page

Delete:

It will display confirmation modal for delete the data, if we press **Yes** then it will delete the data.



OLAP DATABASE Home View ▾

Delete Item

Are you sure want to delete this?

Cancel Yes

Store Key	Store Name	Store Address	Store Launch Date	Action
1	WASHINGTON	WASHINGTON, GA, USA	2017-03-16	Edit Delete
2	ARTESIA	ARTESIA, NM, USA	2015-12-21	Edit Delete
3	NOCONA	NOCONA, TX, USA	2016-03-16	Edit Delete
4	ROME	ROME, NY, USA	2017-06-03	Edit Delete
5	DECORAH	DECORAH, IA, USA	2017-10-18	Edit Delete
6	NORTH KANSAS CITY	NORTH KANSAS CITY, MO, USA	2015-04-27	Edit Delete
7	KENEDY	KENEDY, TX, USA	2014-03-29	Edit Delete
8	VALPARAISO	VALPARAISO, IN, USA	2016-09-29	Edit Delete
9	BLAKELY	BLAKELY, GA, USA	2017-09-17	Edit Delete
10	LAFAYETTE	LAFAYETTE, IN, USA	2016-01-23	Edit Delete
11	SULPHUR SPRINGS	SULPHUR SPRINGS, TX, USA	2015-10-26	Edit Delete
12	AHOSKIE	AHOSKIE, NC, USA	2014-02-22	Edit Delete

Example for Delete Store Dimension modal

IV. Database Normalization (Anomalies)

It is very important to point out that normalization and denormalization are two opposing strategies for structuring data in a relational database.

Normalization is the process of organizing data in a database so that it meets certain design goals, such as minimizing data redundancy, maintaining data integrity, and reducing anomalies that can arise from data updates, deletions, and insertions. The goal of normalization is to create a highly structured, well-organized database schema that conforms to a set of rules called normal forms. Normalization involves breaking up a larger table into smaller tables and creating relationships between them using foreign keys, to eliminate redundant data and ensure that data is stored in a consistent, efficient manner.

Denormalization, on the other hand, is the process of intentionally adding redundant data to a database schema, in order to improve query performance, simplify data retrieval, or achieve other performance-related goals. The goal of denormalization is to create a more flattened, less structured database schema that can return query results more quickly and efficiently. Denormalization typically involves combining related tables into a single table or duplicating data across multiple tables, to reduce the need for joins and other expensive query operations.

While normalization involves breaking up larger tables into smaller tables to eliminate redundant data and ensure consistent data storage, denormalization intentionally adds redundant data to improve query performance and simplify data retrieval. As a result, normalization and denormalization is a trade-off between data consistency, storage efficiency, and query performance.

In short, our designed database, following the OLAP principles, does have so-called “anomalies”, but on purpose for faster and more efficient performance. There will be upsides and downsides as we will point out in the next sections.

V. Advantages & Disadvantages

Because of the major differences between the database design principles with the normal transactional database, the denormalized OLAP structure holds both advantages and disadvantages as follows:

Key Advantages:

- **Fast query response time:** OLAP is designed to handle complex queries on large datasets quickly and efficiently. This can enable users to analyze and extract insights from data in real-time.

- **Flexible data analysis:** OLAP enables users to slice and dice data along multiple dimensions, such as time, geography, product, or customer. This allows for flexible and in-depth data analysis that can uncover hidden patterns and trends.
- **Aggregated views:** OLAP provides aggregated views of data, which can simplify analysis and make it easier to identify trends and outliers.
- **Enhanced reporting capabilities:** OLAP supports sophisticated reporting capabilities, including drill-down, roll-up, and pivot table functionality, that enable users to customize and refine reports to meet their specific needs.
- **Scalability:** OLAP can scale to handle very large datasets, making it suitable for organizations with complex data management needs.

Disadvantages:

- **Complexity:** OLAP is a complex technology that requires specialized skills and expertise to implement and manage effectively.
- **Data quality issues:** OLAP depends on accurate and consistent data, and any data quality issues can negatively impact query results and analysis.
- **Cost:** OLAP can be expensive to implement and maintain, particularly for organizations with large datasets or complex data structures. Updating an OLAP data warehouse requires meticulous planning and implementation.
- **Limited transactional capabilities:** OLAP is not designed for transactional processing, which can limit its usefulness for certain types of data management tasks.
- **Data security and privacy concerns:** OLAP requires access to sensitive data, which can raise security and privacy concerns if not managed properly.

In conclusion, as the world of business is becoming increasingly data-driven, the OLAP Data Warehouse with Star Schema is particularly valuable for analyzing business activities, hence gaining a competitive edge by leveraging the power of data to make informed decisions and identify opportunities for growth.