Modern Walk Database Design

Modern Walk(MW) is an emerging online fashion retailer based out in San Francisco which specializes in fast fashion and clothing. The site ships throughout the USA and offers everything from party looks to athleisure. The retailer procures products directly from suppliers and ships them to the customers. The business goal for Modern walk this year is to increase the sales revenue and reduce the cost expense in order to boost the profit margin by 3%.

A detailed description of entities and their relationships.

1. **Supplier**

   The supplier supplies goods for MW, and each supplier will have a unique Supplier ID. Apart from that, the database also stores the supplier's name, phone number, address, and email information. One supplier can supply one or more products, and one type of product can be supplied by different suppliers. The supply records which include supply quantity and supply date need to be stored.

2. **Product**

   Products are finished goods like hoodies, coats, swimsuits, etc., one product can supply one or more suppliers. For a given product, MW can obtain more competitive prices and lower the chance of a shortage by maintaining many sources. Attributes of products are Product ID (identifying one kind of a product), product name, Category ID and Supplier ID, and product price. Each Product also belongs to one category.

3. **Category**

   Products can be broadly classified into some categories such as sports, casuals, business, etc. There are 15 such categories and one category can have multiple products. The category is recognized by its Category ID and Category Name.

4. **Customer**

   Customer refers to those who registered on the website, no matter whether they have placed an order or not. In order to boost sales, MW needs the following information for further

customer analysis. They are a unique Customer ID associated with every customer, customer's name(first name, middle name, last name), date of birth, gender, address(street, city, state, and zip code), and email.

5. **Order**

Order is placed by the customers. One customer can make zero or many multiple orders, but one order is attributed to a single customer only. Every order is identified a unique Order ID. An order must have at least one payment record. The order contains consolidated information related to the customer who places an order, payment info, shipper info, order date, ship date, and order amount. One order can have multiple payments.

6. **Order Detail**

Order Details record information relating to different products, and the quantity of each product if applicable in a given order. One order may contain different order details but one order detail will only belong to one and only one order. Order Details is recognized by a unique Order_Detail ID and has attributes like order id, product id, and quantity.

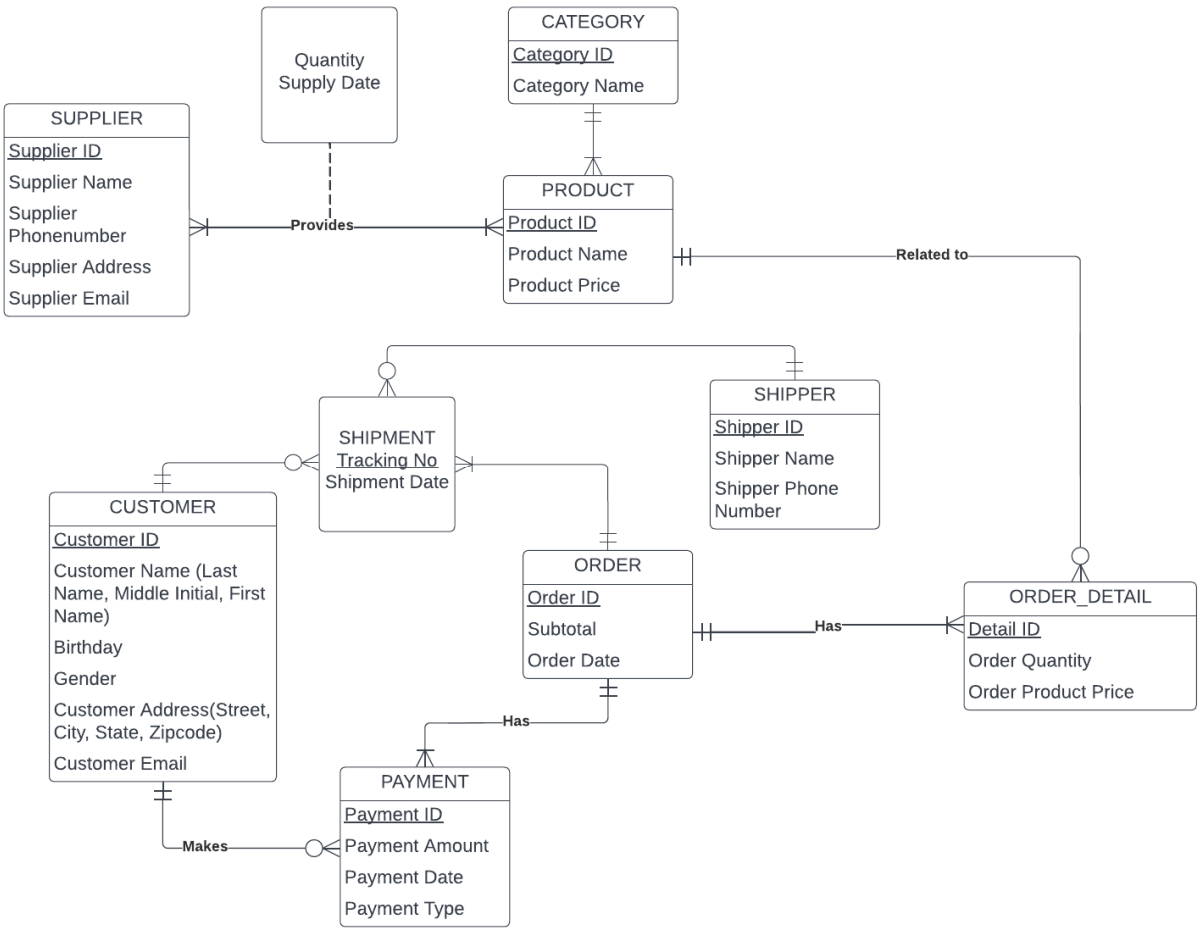
7. **Payment**

The customer can make multiple payments. Each payment will generate a unique Payment ID, and payment date, payment amount, and will store information relating to the type of payment method. There are only two types of payment methods: Credit Cards (CC) and Debit Cards (DC). The payment for one order can be completed in one method only. Also, to safeguard customers' privacy, the firm doesn't store any card-related information.

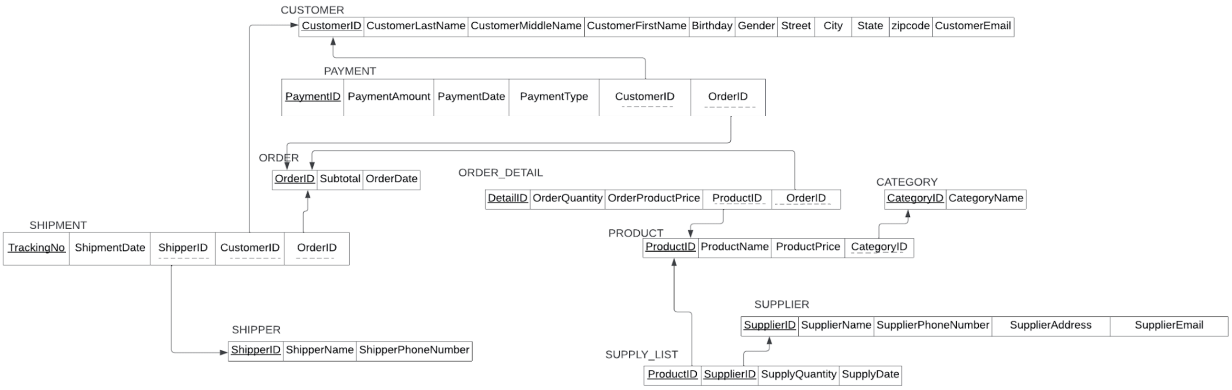
8. **Shipper**

Shippers help MW to ship products to customers. One shipper can ship multiple orders. MW also maintains a repository of a few shippers from which it has never availed any service to date. Also, one order can be shipped in many shipments, in the case of multiple products. This means that for one order, there can be one shipment or multiple shipments. Every shipper has their Shipper ID, name, and phone number.

ERD of Modern Walk



Relations



MetaData:

CUSTOMER

Attribute	Data type	Size	AUTO	Null	Constraint
Customer_id	Int	11	Yes	No	Primary
Customer_last_name	Varchar	30	No	No	
Customer_middle_name	Varchar	30	No	Yes	

Customer_first_name	Varchar	30	No	No	
Birthday	Date		No	No	
Gender	Varchar	30	No	No	
Street	Varchar	50	No	No	
City	Varchar	50	No	No	
State	Varchar	30	No	No	
Zipcode	Int	11	No	No	
Customer_email	Varchar	50	No	No	

ORDER

Attribute	**Data type**	**Size**	**AUTO**	**Null**	**Constraint**
Order_id	Int	11	Yes	No	Primary
Subtotal	Double		no	no	
Order_date	Date		no	no	

PAYMENT

Attribute	**Data type**	**Size**	**AUTO**	**Null**	**Constraint**

Payment_id	Int	11	Yes	No	Primary
Payment_amount	double		No	No	
Payment_date	date		No	Yes	
Payment_type	Varchar	30	No	No	
Customer_id	int	11		No	Foreign
Order_id	int	11		No	Foreign

CATEGORY

Attribute	**Data Type**	**Size**	**AUTO**	**Null**	**Constraint**
Category_id	Int	11	Yes	No	primary
Category_name	Varchar	50	No	No	

PRODUCT

Attribute	**Data Type**	**Size**	**AUTO**	**Null**	**Constraint**

Product_id	Int	11	Yes	No	primary
Product_name	Varchar	50	No	No	
Product_price	Double		No	No	
Category_id	Int	11	No	no	foreign

ORDER_DETAIL

Attribute	**Data Type**	**Size**	**AUTO**	**Null**	**Constraint**
Detail_id	Int	11	Yes	No	primary
Order_quantity	Int	11	No	No	
Order_Product_price	Double		No	No	
Product_id	int	11		no	foreign

Order_id	Int	11		No	foreign

SHIPPER

Attribute	**Data type**	**Size**	**AUTO**	**Null**	**Constraint**
Shipper_id	Int	11	Yes	No	Primary
Shipper_name	Varchar	30	No	No	
Shipper_phone_number	varchar	30	No	No	

SHIPMENT

Attribute	**Data type**	**Size**	**AUTO**	**Null**	**Constraint**
Tracking_No	Int	11	Yes	No	Primary
Shipment_date	Date		No	No	
Shipper_id	Int	11		No	Foreign
Customer_id	Int	11		No	Foreign
Order_id	Int	11		No	Foreign

SUPPLIER

Attribute	Data Type	Size	AUTO	Null	Constraint
Supplier_id	Int	11	Yes	No	primary
Supplier_name	Varchar	50	No	No	
Supplier_phone_number	Varchar	30	No	no	
Supplier_address	Varchar	50	No	No	
Supplier_Email	Varchar	50	No	no	

SUPPLY_LIST

Attribute	Data Type	Size	AUTO	Null	Constraint
Product_id	Int	11	No	No	Primary,foreign

Supplier_id	Int	11	no	No	Primary,foreign
Supply_quantity	Int	11	No	no	
SupplyDate	Date			No	

Implementation in SQL- DDL Create Tables

Next we converted all of the above metadata and the relations tables into a SQL database called 'modern_walk'. Firstly, for the CUSTOMER table, we wrote the following code:

```
CREATE TABLE customer_T
    (customer_id int(11) not null auto_increment,
    customer_last_name varchar(30) not null,
    customer_middle_name varchar (30),
    customer_first_name varchar (30) not null,
    birthday date not null,
    gender varchar (30) not null,
    street varchar (30) not null,
    city varchar(30) not null,
    state varchar(30) not null,
    zipcode int (11) not null,
    customer_email varchar(50) not null,
    CONSTRAINT customer_pk PRIMARY KEY (customer_id))
ENGINE=INNODB;
```

On the first line of the code, we specified that customer_id should be an integer and should auto-increment if a value was not provided. All of our variables cannot be missing or null as indicated by the 'not null' statement. The second to last line specified that customer_id is the primary key. We also name this constraint as customer_pk and can reference this later, if we need to make changes.

Secondly, we created the ORDER table in a similar manner:

```
CREATE TABLE order_T
(order_id int(11) not null auto_increment,
subtotal double not null,
order_date date not null,
CONSTRAINT order_pk PRIMARY KEY (order_id));
```

The PAYMENT table, however, had additional constraints for us to consider:

```
/*Create payment table*/
CREATE TABLE payment_T
(payment_id int(11) not null auto_increment,
payment_amount double not null,
payment_date date not null,
payment_type varchar(30) not null,
customer_id int(11) not null,
order_id int(11) not null,
CONSTRAINT payment_pk PRIMARY KEY (payment_id),
CONSTRAINT payment_fk1 FOREIGN KEY (customer_id)
REFERENCES customer_T (customer_id)
ON UPDATE CASCADE ON DELETE RESTRICT,
CONSTRAINT payment_fk2 FOREIGN KEY (order_id)
REFERENCES order_T (order_id)
ON UPDATE CASCADE ON DELETE RESTRICT);
```

Since each payment corresponds to both a customer and an order, we needed to include customer_id and order_id in the table. These two ids are foreign keys and are defined to be as such in the constraints section of the code. The line 'REFERENCES customer_t (customer_id)' specifies that the foreign key of customer_id in the payments table references the primary key in the customer table (customer_id). Next, we indicate that if the primary key changes in the customer table, this change should 'cascade down to the payments tables as well. The 'ON DELETE RESTRICT' command prevents the user from deleting the customer record from the customer table if there already exists a record in the payment tables with that customer_id.

Next is the CATEGORYtable.

```
/*Create category table*/
CREATE TABLE category_T
(category_id int(11) not null auto_increment,
category_name varchar (50) not null,
CONSTRAINT category_pk PRIMARY KEY (category_id))
ENGINE=INNODB;
```

This is similarly created like customer and order tables.

Then, the PRODUCT table.

```
/*Create product table*/
CREATE TABLE product_T
(product_id int(11) not null auto_increment,
product_name varchar(50) not null,
product_price double not null,
category_id int(11) not null,
CONSTRAINT product_pk PRIMARY KEY (product_id),
CONSTRAINT product_fk FOREIGN KEY (category_id)
REFERENCES category_T (category_id)
ON UPDATE CASCADE ON DELETE RESTRICT);
```

In the product table, apart from the primary key constraint, i.e. product_id, there is a foreign key constraint of category_id. The line 'REFERENCE' category_t (category_id)' specifies that the foreign key of category_id in the product table references the primary key in the category table (category_id). The 'ON UPDATE CASCADE' command indicates that if the primary key changes in the category table, this change should 'cascade down' to the product table as well. The 'ON DELETE RESTRICT' command prevents the user from deleting the category record from the product table if there already exists a record in the product table with that category_id.

Next is the ORDER_DETAIL table.

```
/*Create order_detail table*/
CREATE TABLE order_detail_T
(detail_id int(11) not null auto_increment,
order_quantity int(11) not null,
order_product_price double not null,
product_id int(11) not null,
order_id int(11) not null,
CONSTRAINT order_detail_pk PRIMARY KEY (detail_id),
CONSTRAINT order_detail_fk1 FOREIGN KEY (product_id)
REFERENCES product_T (product_id) ON UPDATE CASCADE ON DELETE RESTRICT,
CONSTRAINT order_detail_fk2 FOREIGN KEY (order_id)
REFERENCES order_T (order_id)
ON UPDATE CASCADE ON DELETE RESTRICT);
```

Here, apart from the primary key constraint of detail_id, we have two additional constraints. They are product_id and order_id, which are foreign keys in this table, each having its REFERENCE from product table and order table, respectively, where they are the primary keys.

The 'ON UPDATE CASCADE' command indicates that if the primary key changes in the product and order tables, this change should 'cascade down' to the order_detail table as well. The 'ON DELETE RESTRICT' command prevents the user from deleting the product and order record from the order_detail table if there already exist records in the order_detail table with the product_id and order_id.

Next is the SHIPPER table.

```
/*Create shipper table*/
CREATE TABLE shipper_T
(shipper_id int(11) not null auto_increment,
shipper_name varchar (30) not null,
shipper_phone_number varchar(30) not null,
CONSTRAINT shipper_pk PRIMARY KEY (shipper_id));
```

This is similarly created like customer, order and category tables.

The next table in the sequence is the SHIPMENT table.

```
/*Create shipment table*/
CREATE TABLE shipment_T
(tracking_no int(11) not null auto_increment,
shipment_date date not null,
shipper_id int(11) not null,
customer_id int(11) not null,
order_id int(11) not null,
CONSTRAINT shipment_pk PRIMARY KEY (tracking_no),
CONSTRAINT shipment_fk1 FOREIGN KEY (shipper_id)
REFERENCES shipper_T (shipper_id) ON UPDATE CASCADE ON DELETE RESTRICT,
CONSTRAINT shipment_fk2 FOREIGN KEY (customer_id)
REFERENCES customer_T (customer_id) ON UPDATE CASCADE ON DELETE RESTRICT,
CONSTRAINT shipment_fk3 FOREIGN KEY (order_id)
REFERENCES order_T (order_id) ON UPDATE CASCADE ON DELETE RESTRICT);
```

Here, apart from the primary key constraint of tracking_no, we have three additional constraints. They are shipper_id, customer_id, and order_id, which are foreign keys in this table, each having its REFERENCE from the shipment, customer, and order tables, respectively, where they are the primary keys. The 'ON UPDATE CASCADE' command indicates that if the primary key changes in the shipment, customer and order tables, this change should 'cascade down to the shipment table as well. The 'ON DELETE RESTRICT' command prevents the user from

deleting the shipment, customer, and order records from the shipment table if there are existing records in the shipment table with the shipper_id, customer_id, and order_id.

Then, comes the SUPPLIER table.

```
/*Create supplier table*/
CREATE TABLE supplier_T
(supplier_id int(11) not null auto_increment,
supplier_name varchar (50) not null,
supplier_phone_number varchar(30) not null,
supplier_address varchar(50) not null,
supplier_email varchar (50) not null,
CONSTRAINT supplier_pk PRIMARY KEY (supplier_id));
```

This is a relatively simple table, created similarly to the customer, order, category, and shipper table, without any foreign key references.

The last table we created is the SUPPLY_LIST table.

```
/*Create supply_list table*/
CREATE TABLE supply_list_T
(product_id int(11) not null,
supplier_id int(11) not null,
supply_quantity int(11) not null,
supply_date date not null,
CONSTRAINT supply_list_pk PRIMARY KEY (product_id, supplier_id),
CONSTRAINT supply_list_fk1 FOREIGN KEY (product_id)
REFERENCES product_T (product_id) ON UPDATE CASCADE ON DELETE RESTRICT,
CONSTRAINT supply_list_fk2 FOREIGN KEY (supplier_id)
REFERENCES supplier_T (supplier_id) ON UPDATE CASCADE ON DELETE RESTRICT);
```

Here, there is a composite primary key, which consists of product_id and supplier_id. Also, the product_id and supplier_id are the foreign keys in this table, each having its REFERENCE from the product and supplier tables, respectively, where they are the primary keys. The 'ON UPDATE CASCADE' command indicates that if the primary key changes in the product and supplier tables, this change should 'cascade down' to the supply_list table as well. The 'ON DELETE RESTRICT' command prevents the user from deleting the product and supplier records from the supply_list table if there are already existing records in the supply_list table with the product_id and supplier_id.

Implementation in SQL- DDL Insert Values

We have inserted values in different tables.

CUSTOMER Table

```
/* Insert values to customer table */
INSERT INTO customer_t (customer_last_name,customer_middle_name,customer_first_name,birthday,gender,street,city,state,zi
VALUES ('Burks','hans','debra','1999-01-02','male','9273 thomes ave','san francisco','CA','94117','burkshans@gmail.com')
('Kasha','Pals','todd','1985-11-21','male','769 west road','seattle','WA','98101','kptodd@gmail.com'),
('Tameka','dounce','fisher','2002-06-12','male','107 river drive','new york city','NY','10005','tamekafisher258@yahoo.co
('Lyndsey','lee','bean','1982-01-15','female','15 brown street','salt lake city','UT','84109','leebean0115@gmail.com'),
('pamela','wooden','newman','1989-08-20','female','8550 spurce drive','boston','MA','22108','pamela1989@outlook.com'),
('Jacquline','ricey','duncan','1998-09-13','female','476 cheastnut ave','campell','CA','95008','kricey0913@qq.com'),
('Alfreds','futterkiste','moreal','1978-11-26','male','476 cheastnut ave','Fomey','TX','97516','alfredm1126d@yahoo.com')
('ana','taqueria','newton','1999-06-27','female','3550 golden gate ave','san jose','CA','95126','ananewtono2036@hotmail.
```

customer_id	customer_last_na...	customer_middle_na...	customer_first_na...	birthday	gender	street	city	state	zipcode	customer_email
1	Burks	hans	debra	1999-01-02	male	9273 thomes ave	san francisco	CA	94117	burkshans@gmail.com
2	Kasha	Pals	todd	1985-11-21	male	769 west road	seattle	WA	98101	kptodd@gmail.com
3	Tameka	dounce	fisher	2002-06-12	male	107 river drive	new york city	NY	10005	tamekafisher258@yahoo.
4	Lyndsey	lee	bean	1982-01-15	female	15 brown street	salt lake city	UT	84109	leebean0115@gmail.com
5	pamela	wooden	newman	1989-08-20	female	8550 spurce drive	boston	MA	22108	pamela1989@outlook.cor
6	Jacquline	ricey	duncan	1998-09-13	female	476 cheastnut ave	campell	CA	95008	kricey0913@qq.com
7	Alfreds	futterkiste	moreal	1978-11-26	male	476 cheastnut ave	Fomey	TX	97516	alfredm1126d@yahoo.cor
8	ana	taqueria	newton	1999-06-27	female	3550 golden gate ave	san jose	CA	95126	ananewtono2036@hotma
9	Burks	hans	debra	1999-01-02	male	9273 thomes ave	san francisco	CA	94117	burkshans@gmail.com
10	Kasha	Pals	todd	1985-11-21	male	769 west road	seattle	WA	98101	kptodd@gmail.com
11	Tameka	dounce	fisher	2002-06-12	male	107 river drive	new york city	NY	10005	tamekafisher258@yahoo.
12	Lyndsey	lee	bean	1982-01-15	female	15 brown street	salt lake city	UT	84109	leebean0115@gmail.com
13	pamela	wooden	newman	1989-08-20	female	8550 spurce drive	boston	MA	22108	pamela1989@outlook.cor
14	Jacquline	ricey	duncan	1998-09-13	female	476 cheastnut ave	campell	CA	95008	kricey0913@qq.com
15	Alfreds	futterkiste	moreal	1978-11-26	male	476 cheastnut ave	Fomey	TX	97516	alfredm1126d@yahoo.cor
16	ana	taqueria	newton	1999-06-27	female	3550 golden gate ave	san jose	CA	95126	ananewtono2036@hotma
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

We did not explicitly specify customer id in the above code because the field is set to auto increment in our DDL definition of the table. The fields that are entered must match the date type that we specified earlier. Otherwise, we would get an error message. We repeat the process for the other tables below.

ORDER Table

```
/* Insert values to order table */
INSERT INTO order_t (subtotal,order_date)
VALUES  (833,'2022-10-25'),
(108,'2022-10-26'),
(700,'2022-10-26'),
(1274,'2022-10-27'),
(280,'2022-10-28'),
(80,'2022-10-28'),
(258,'2022-11-02'),
(392,'2022-11-02'),
(610,'2022-11-03');
```

order_id	subtotal	order_date
1	833	2022-10-25
2	108	2022-10-26
3	700	2022-10-26
4	1274	2022-10-27
5	280	2022-10-28
6	80	2022-10-28
7	258	2022-11-02
8	392	2022-11-02
9	610	2022-11-03
NULL	NULL	NULL

PAYMENT Table

```
/* Insert values to payemnt table */
INSERT INTO payment_t (payment_amount,payment_date,payment_type,customer_id,order_id)
VALUES (833,'2022-10-26','DC',2,1),
(108,'2022-10-26','CC',3,2),
(700,'2022-10-26','CC',6,3),
(1274,'2022-10-27','DC',8,4),
(280,'2022-10-28','CC',6,5),
(80,'2022-10-28','DC',5,6),
(258,'2022-11-02','CC',4,7),
(392,'2022-11-02','DC',3,8),
(600,'2022-11-03','DC',2,9),
(10,'2022-11-04','DC',2,9);
```

payment_id	payment_amou...	payment_date	payment_type	customer_id	order_id
1	833	2022-10-26	DC	2	1
2	108	2022-10-26	CC	3	2
3	700	2022-10-26	CC	6	3
4	1274	2022-10-27	DC	8	4
5	280	2022-10-28	CC	6	5
6	80	2022-10-28	DC	5	6
7	258	2022-11-02	CC	4	7
8	392	2022-11-02	DC	3	8
9	600	2022-11-03	DC	2	9
10	10	2022-11-04	DC	2	9
NULL	NULL	NULL	NULL	NULL	NULL

CATEGORY Table

```sql
/* Insert values to category table */
INSERT INTO category_t (category_name)
VALUES ('coat'),
('shoes'),
('bag'),
('accessories'),
('beachwear'),
('pants'),
('lingerie'),
('dress'),
('activewear'),
('suits');
```

category_id	category_name
1	coat
2	shoes
3	bag
4	accessories
5	beachwear
6	pants
7	lingerie
8	dress
9	activewear
10	suits

PRODUCT Table

```sql
/* Insert values to product table */
INSERT INTO product_t (product_name,product_price,category_id)
VALUES ('moonboot',175,2),
('browns base layer ski top',149,9),
('browsn base layer ski leggins',120,9),
('Guccy single breasted tailored coat',108,1),
('medium carolyn tote bag',350,3),
('down feather 4bar overcoat',395,1),
('medusa greek key bar',140,7),
('halterneck evening dress',1274,8),
('marina jumpsuit',120,10),
('strapless swimsut',69,5),
('logo-patch padded zip-up vest',450,9),
('interlock print silk scarf',95,4),
('duchesse cargo trouser',76,6),
('slouchy 105mm leather boots',780,3),
('bon bon crystal bucket',469,4),
('cat eye fram sunglasses',196,5),
('barocco-panel logo devore robe',80,7),
('Bea single-breasted boxy blazer',258,10),
('Parka II oversized ski jacket',610,9);
```

product_...	product_name	product_pri...	category_id
1	moonboot	175	2
2	browns base layer ski top	149	9
3	browsn base layer ski leggins	120	9
4	Guccy single breasted tailored coat	108	1
5	medium carolyn tote bag	350	3
6	down feather 4bar overcoat	395	1
7	medusa greek key bar	140	7
8	halterneck evening dress	1274	8
9	marina jumpsuit	120	10
10	strapless swimsut	69	5
11	logo-patch padded zip-up vest	450	9
12	interlock print silk scarf	95	4
13	duchesse cargo trouser	76	6
14	slouchy 105mm leather boots	780	3
15	bon bon crystal bucket	469	4
16	cat eye fram sunglasses	196	5
17	barocco-panel logo devore robe	80	7
18	Bea single-breasted boxy blazer	258	10
19	Parka II oversized ski jacket	610	9

ORDER_DETAIL Table

```
/* Insert values to order_detail_t table */
INSERT INTO order_detail_t (order_quantity,order_product_price,product_id,order_id)
VALUES (1,175,1,1),
(2,149,2,1),
(3,120,3,1),
(1,108,4,2),
(2,350,5,3),
(1,1274,8,4),
(2,140,7,5),
(1,80,17,6),
(1,258,18,7),
(2,196,16,8),
(1,610,19,9);
```

detail_id	order_quanti...	order_product_pr...	product_...	order_id
1	1	175	1	1
2	2	149	2	1
3	3	120	3	1
4	1	108	4	2
5	2	350	5	3
6	1	1274	8	4
7	2	140	7	5
8	1	80	17	6
9	1	258	18	7
10	2	196	16	8
11	1	610	19	9

SHIPPER Table

```
/* Insert values to shipper_t table */
INSERT INTO shipper_t (shipper_name,shipper_phone_number)
VALUES ('fastly','800-288-2828'),
('speedy','258-2583-522'),
('Rapid Crew','147-025-3691'),
('Maersk','963-145-2035'),
('Flashy','415-203-9652'),
('cosco','825-914-8635');
```

shipper_id	shipper_name	shipper_phone_num...
1	fastly	800-288-2828
2	speedy	258-2583-522
3	Rapid Crew	147-025-3691
4	Maersk	963-145-2035
5	Flashy	415-203-9652
6	cosco	825-914-8635

SHIPMENT Table

```
/* Insert values to shipment table */

INSERT INTO shipment_t (tracking_no,shipment_date,shipper_id,customer_id,order_id)
VALUES (1000001,'2022-10-28',1,2,1);
INSERT INTO shipment_t (shipment_date,shipper_id,customer_id,order_id)
VALUES ('2022-11-4',3,8,2),
 ('2022-11-5',6,7,3),
('2022-10-28',4,3,4),
('2022-11-4',2,4,9),
('2022-11-4',5,6,8),
 ('2022-11-4',3,1,7);
```

	tracking_...	shipment_date	shipper_id	customer_id	order_id
►	1000001	2022-10-28	1	2	1
	1000002	2022-11-04	3	8	2
	1000003	2022-11-05	6	7	3
	1000004	2022-10-28	4	3	4
	1000005	2022-11-04	2	4	9
	1000006	2022-11-04	5	6	8
	1000007	2022-11-04	3	1	7

SUPPLIER Table

```
/* Insert values to supplier table */
INSERT INTO supplier_t (supplier_name,supplier_phone_number,supplier_address,supplier_email)
VALUES ('AIM garments','259-025-1472','25 Maple Street Carmel, IN','aim25@gmail.com'),
('Aplex Mills','417-325-1025','660 Oak Street Cary, NC','aplex@yahoo.com'),
('Hollingsworth','102-022-1458','660 golden gate Street Fairlawn, OH','hollingsworth@outlook.com'),
('Tabb','810-025-3252','1520 Main street Southfield, MI','tabbthebest@gmail.com'),
('Emco','926-596-1466','580 16th Ave Great Neck, NY','emco2528@yahoo.com'),
('Saint Gobain','597-225-2678','652 milkway Rd Gaffney, SC','sg5285@gmail.com');
```

supplier_id	supplier_name	supplier_phone_num...	supplier_address	supplier_email
1	AIM garments	259-025-1472	25 Maple Street Carmel, IN	aim25@gmail.com
2	Aplex Mills	417-325-1025	660 Oak Street Cary, NC	aplex@yahoo.com
3	Hollingsworth	102-022-1458	660 golden gate Street Fairlawn, OH	hollingsworth@outlook.com
4	Tabb	810-025-3252	1520 Main street Southfield, MI	tabbthebest@gmail.com
5	Emco	926-596-1466	580 16th Ave Great Neck, NY	emco2528@yahoo.com
6	Saint Gobain	597-225-2678	652 milkway Rd Gaffney, SC	sg5285@gmail.com
7	AIM garments	259-025-1472	25 Maple Street Carmel, IN	aim25@gmail.com
8	Aplex Mills	417-325-1025	660 Oak Street Cary, NC	aplex@yahoo.com
9	Hollingsworth	102-022-1458	660 golden gate Street Fairlawn, OH	hollingsworth@outlook.com
10	Tabb	810-025-3252	1520 Main street Southfield, MI	tabbthebest@gmail.com
11	Emco	926-596-1466	580 16th Ave Great Neck, NY	emco2528@yahoo.com
12	Saint Gobain	597-225-2678	652 milkway Rd Gaffney, SC	sg5285@gmail.com

SUPPLY_LIST Table

```
/* Insert values to supply_list table */
INSERT INTO supply_list_t (product_id,supplier_id,supply_quantity,supply_date)
VALUES (1,2,10,'2022-05-06'),
(2,3,20,'2022-06-04'),
(3,4,15,'2022-05-06'),
(4,3,80,'2022-06-12'),
(5,6,10,'2022-05-12'),
(6,1,22,'2022-07-16'),
(7,5,16,'2022-08-06'),
(8,2,32,'2022-09-06'),
(16,5,6,'2022-10-06'),
(17,6,9,'2022-07-12'),
(18,4,20,'2022-07-22'),
(19,3,15,'2022-05-18');
```

product_...	supplier_id	supply_quanti...	supply_date
1	2	10	2022-05-06
2	3	20	2022-06-04
3	4	15	2022-05-06
4	3	80	2022-06-12
5	6	10	2022-05-12
6	1	22	2022-07-16
7	5	16	2022-08-06
8	2	32	2022-09-06
16	5	6	2022-10-06
17	6	9	2022-07-12
18	4	20	2022-07-22
19	3	15	2022-05-18

Implementation in SQL- DML Queries

5 Queries

```
/* 1.checking which customer has not placed orders */
SELECT c.*
FROM customer_t c LEFT JOIN  payment_t p
ON c.customer_id = p.customer_id
WHERE payment_id IS NULL;
```

customer_id	customer_last_name	customer_middle_name	customer_first_name	birthday	gender	street	city	state	zipcode	customer_email
1	Burks	hans	debra	1999-01-02	male	9273 thomes ave	san francisco	CA	94117	burkshans@gmail.com
7	Alfreds	futterkiste	moreal	1978-11-26	male	476 cheastnut ave	Fomey	TX	97516	alfredm1126d@yahoo.com

In this first query we look at which customer is associated with a missing payment. To get this information we LEFT JOIN the payment table onto the customer table on customer_id. If there is no corresponding match in the payment table, then payment_id will be null in the output. We use this condition to get customers who have not placed orders.

```
/* 2.checking the number of order of each state  */
SELECT state,count(o.order_id) as number_of_order
FROM customer_t c LEFT JOIN payment_t p
ON c.customer_id = p.customer_id
LEFT JOIN order_t o
ON p.order_id = o.order_id
GROUP BY c.state;
```

state	number_of_order
CA	3
WA	3
NY	2
UT	1
MA	1
TX	0

In the above query, we are counting up the number of orders by state. Since state is only present in the customer table and orders are tracked in the order table, we need to merge the two tables. We do this with the help of the payment table, which has both customer_id and order_id. We use LEFT JOIN to connect all three tables, this ensures that we keep all of the records in customer and drop those that do not have a match in customer.

```
/*3.find the customer who lives in CA whose the amount of order is grater than 200 and who purchased product_id = 5 */
SELECT c.*
FROM customer_t c
INNER JOIN payment_t t ON c.customer_id = t.customer_id
INNER JOIN order_t o ON t.order_id = o.order_id
INNER JOIN order_detail_t od ON o.order_id = od.order_id
WHERE c.state = 'CA' and O.subtotal > 200 and od.product_id =5;
```

customer_id	customer_last_name	customer_middle_name	customer_first_name	birthday	gender	street	city	state	zipcode	customer_email
6	Jacquline	ricey	duncan	1998-09-13	female	476 cheastnut ave	campell	CA	95008	kricey0913@qq.com

```
/* 4.list all the order information after 5% discount*/
SELECT * , subtotal *0.95 AS discounted_price FROM order_T;
```

order_id	subtotal	order_date	discounted_price
1	833	2022-10-25	791.3499999999999
2	108	2022-10-26	102.6
3	700	2022-10-26	665
4	1274	2022-10-27	1210.3
5	280	2022-10-28	266
6	80	2022-10-28	76
7	258	2022-11-02	245.1
8	392	2022-11-02	372.4
9	610	2022-11-03	579.5

This query is straightforward: we extract all of the variables from the order table and create a new variable called discounted_price by multiplying subtotal by 95%.

```
SELECT *
FROM supplier_t s LEFT JOIN supply_list_t sl
ON s.supplier_id =sl.supplier_id
WHERE sl.supplier_id IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

supplier_id	supplier_name	supplier_phone_number	supplier_address	supplier_email	product_id	supplier_id	supply_quantity	supply_date

In this query, we again utilize LEFT JOIN to find the records that are in one table but not in the other one. We merge the supply list table onto the supplier table - we keep all of the records in the supplier table and only matching records from the supplier list table. In our case, all suppliers have already provided some products so the output contains nothing.

Views

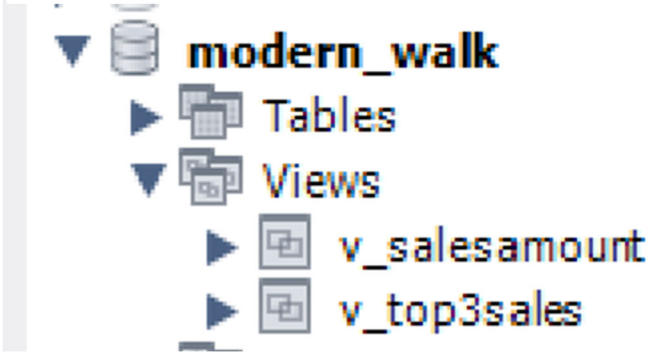
Two views are created in this part:

```
/* View*/

/*1. calculate the total amount of sales*/
CREATE VIEW v_salesamount AS
SELECT sum(subtotal) AS total_amount_of_sales
FROM order_T;

SELECT * FROM v_salesamount;
```

total_amount_of_sales
4535

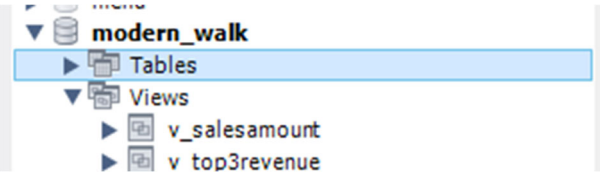


The above view captures the sum of all sales in the database. Views and procedures are a convenient way to capture and store redundant queries that are run frequently. One advantage of views, is that only the query instructions are stored during at the time of execution.The table is created when the user asks for it in another query.

```sql
/*2. get the top 3 highest revenue  of product */
CREATE VIEW v_top3revenue AS
SELECT product_T.product_id, product_T.product_name, sum(order_quantity * order_detail_T.order_product_price) AS product_revenue
FROM order_detail_T INNER JOIN product_T
ON order_detail_T.product_id = product_T.product_id
GROUP BY order_detail_T.product_id
ORDER BY  product_revenue DESC
LIMIT 3;
```

```sql
SELECT * FROM v_top3revenue;
```

product_id	product_name	product_revenue
8	halterneck evening dress	1274
5	medium carolyn tote bag	700
19	Parka II oversized ski jacket	610



The above view combines the order detail table with the product table using an INNER JOIN, this means that the query will keep only records which share the same product id. We also calculate an aggregated variable, product_revenue. We group by product_id and sum the product of order_quantity and order_product_price. The output is also limited to 3 records and sorted in descending order. This way we see the top 3 products with the highest revenue.

Procedures

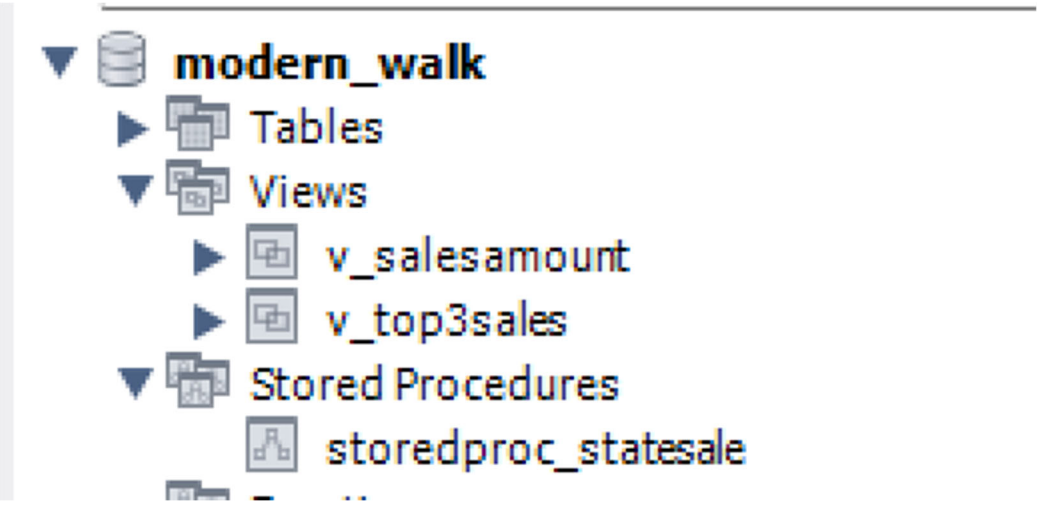
Two procedures are created, one is to calculate the total revenue by state and another is to show the supplier information by taking product id as a parameter.

```
/*Procedure*/

 /*1. calculate the total amount of sales by state*/
DELIMITER $$
CREATE procedure storedproc_statesale(state_name VARCHAR(50))
BEGIN
SELECT c.state,
CASE
WHEN sum(o.subtotal) IS NULL THEN  0
ELSE sum(o.subtotal)
END AS total_sales
FROM customer_t c LEFT JOIN payment_t p ON c.customer_id = p.customer_id
LEFT JOIN order_t o ON o.order_id = p.order_id
WHERE state_name = c.state;
END $$
DELIMITER ;

CALL storedproc_statesale('CA');
```

state	total_sales
CA	2254



The above procedure creates a convenient way to access sales by state. The procedure also checks for missing values in the subtotal field and replaces them with 0. At the top of the

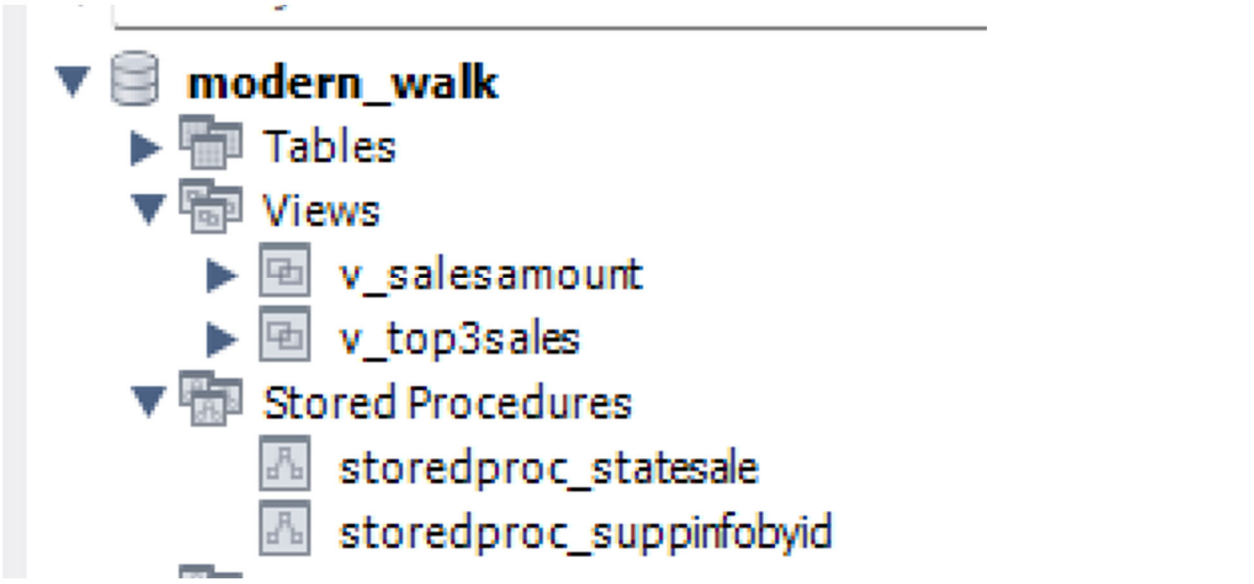
procedure we initialize a user-defined variable that will allow us to filter the tables further down. The input variable is set to be character and limited to 50 characters total. This new variable is referenced in the where statement.

```
    /*2. get the supplier info by entering specific product id*/
DELIMITER $$
CREATE procedure storedproc_suppinfobyid(prod_id INT)
BEGIN
SELECT p.product_id,
CASE  WHEN s.supplier_name IS NULL THEN  'the product has not been supplied yet'
ELSE s.supplier_name
END AS supplier_name,
s.supplier_id,s.supplier_address,s.supplier_phone_number,s.supplier_email
FROM product_t p LEFT JOIN supply_list_t sl
ON p.product_id = sl.product_id
LEFT JOIN supplier_t s ON sl.supplier_id = s.supplier_id
WHERE prod_id = p.product_id;
END $$
DELIMITER ;
```

```
CALL storedproc_suppinfobyid(17);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_id	supplier_name	supplier_id	supplier_address	supplier_phone_number	supplier_email
17	Saint Gobain	6	652 milkway Rd Gaffney, SC	597-225-2678	sg5285@gmail.com



This procedure allows users to easily access supplier information for a given product. The user specifies the product id and the procedure returns all of the supplier information for that product id.

Conclusion

Modern Walk was pleased with our work and has retained us to maintain their ever-growing database. Prior to our involvement, Modern Walk was struggling to keep up with the data needs of their fast growing business. It had become clear to them that a structured method of storing and accessing data was needed - excel workbooks would not do the trick! This is why they decided to hire us! Modern Walk can now focus on growing their business as opposed to managing their data. We believe that our mission and goals have been accomplished!