**Database Systems Project**

**Digital Farmers Market**

Renee Raven

3 Other Students

Names Concealed for

Privacy Concerns

MSIT 630 – Database Systems

# Table of Contents

# Introduction

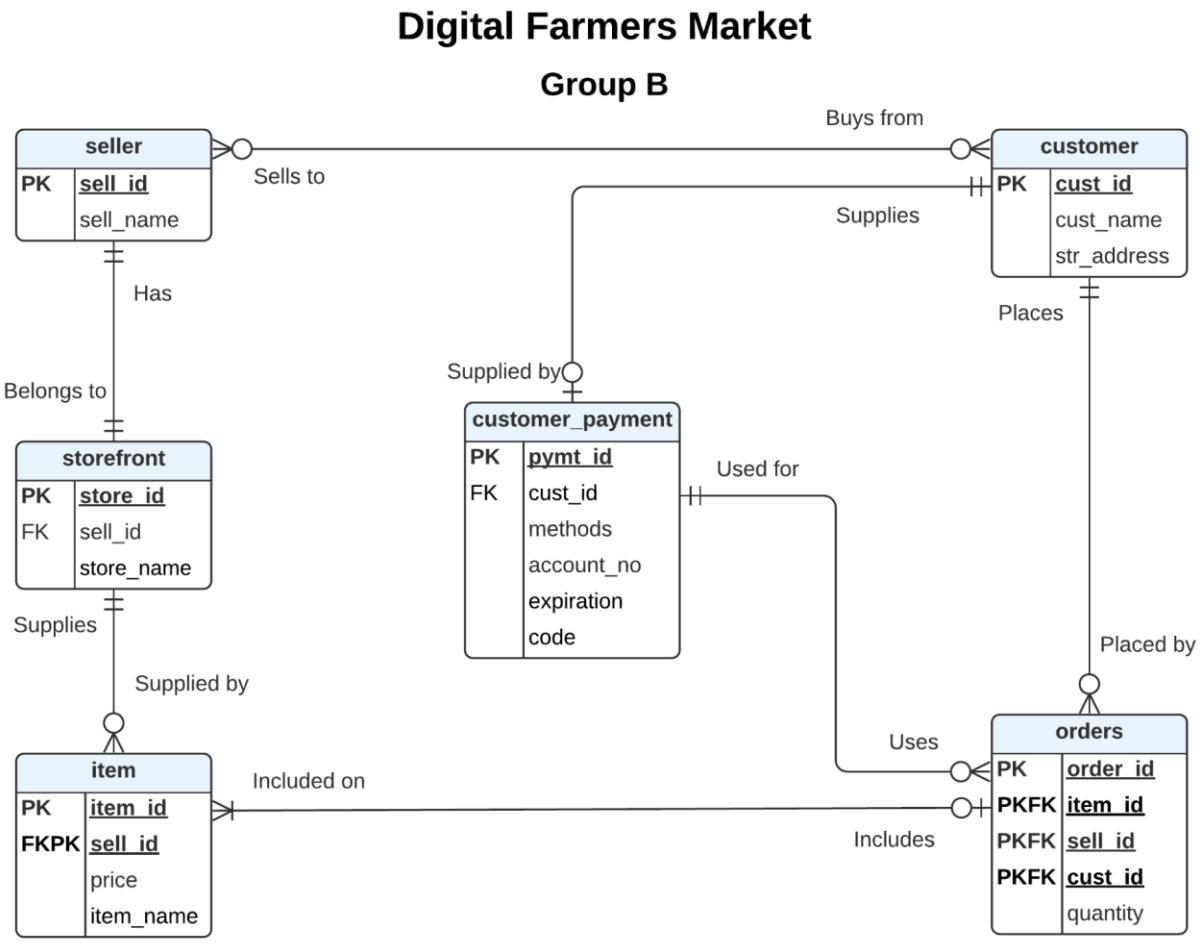
## Description of the Database

The Digital Farmers Market is an application that brings farm-fresh foods directly to the customer table. The concept is to provide an easy-to-use service which allows customers to purchase items from a Farmer's Market, and have it delivered directly to them. Each store sells a unique type of item, from herbs to fruits, and vegetables to organic items.

There are 6 entities that make up the database: Storefront, Seller, Customer, Item, Customer Payment, and Orders, which hold a variety of information, such as seller, and customer, item, order, and payment data.

# Conceptual Design

## Entity-Relationship Diagram & Mapping Cardinalities




Mapping Cardinalities:

| | |
|---|---|
| A seller has one and only one storefront. A storefront belongs to one and only one seller. | 1:1 |
| A storefront may supply none, one, or many items. An item must be supplied by one and only one storefront. | 1:M |
| An item may be included on none, one or many order(s). An order must include one or many items. | 1:M |
| A seller may sell to none, one or many customers. A customer may buy from none, one or many sellers. | N:M |
| A customer may place none, one or many order(s). An order, if it exists, must belong to one, and only one, customer. | 1:M |
| A customer none or one customer_payment info. customer_payment info must be supplied by one, and only one, customer. | 1:1 |
| An order uses one and only one customer_payment. A customer_payment can be used for none, one, or many orders. | 1:M |

# Logical Design

## Normalization Discussion

The six tables are in 3NF.

Each table meets the criteria of 1NF (every attribute is atomic and single-valued, and each table has an identified primary key).

Each table meets the criteria for 2NF. Four tables (seller, storefront, customer, and customer_payment don't have composite primary keys, so they are already in 2NF. The item table has a composite key of item_id and sell_id. The value of the non-primary key column price depends on both item_id and sell_id because different sellers may sell the same item for different amounts. Also, the value of the non-primary key column item_name depends on both item_id and sell_id because different sellers could sell the same item under a different name. Therefore, the item table meets the criteria for 2NF since there are no partial dependencies.

The orders table is also in 2NF. The orders table has a composite key of order_id, item_id, sell_id, cust_id. The value of the non-primary key column quantity depends on order_id because it is unique for every order. The value of the non-primary key column quantity depends on item_id because it is unique for every instance of the order. The value of the non-primary key column quantity depends on sell_id because different sellers may sell the same item and it is part of the composite key for the items table. The value of the non-primary key column quantity depends on cust_id because every order is placed by one unique customer. Therefore, the orders table meets the criteria for 2NF since quantity depends on all elements of the primary key.

Finally, all tables meet the criteria for 3NF because all values in non-primary key columns are determined by the primary key and there are no transitive dependencies on non-primary key columns.

## Digital Farmers Market Entity-Relationship Tables

### Storefront table

The Storefront table displays data unique to each store and seller combination. There are 5 stores with 15 total values with 3 table attributes:

- store_id (primary key)- unique ID for the store
- sell_id (foreign key)- unique ID for the seller; references the seller table
- store_name- name for each store

| Storefront table | | |
|---|---|---|
| **store_id (P)** | **sell_id (FK)** | store_name |
| FMR1001 | SEL1001 | Organic Lyfe |
| FMR1002 | SEL1002 | Herb's Herbs |
| FMR1003 | SEL1003 | Vickie for Vegans |
| FMR1004 | SEL1004 | Appleseed Farms |
| FMR1005 | SEL1005 | Founding Farmers |

### Item table

The Item table displays the combined data for each store item, seller, and list price. There are 25 items with 125 values in the table and 5 attributes:

- item_id (primary key)- unique ID for the item
- sell_id (foreign key)- unique ID for the seller, references the seller table
- item_name- name of each item
- price- list price for each item

| item table | | | |
|---|---|---|---|
| **item_id (P)** | **sell_id (FK)** | item_name | price |
| ITEM1001 | SEL1001 | Organic Orange Juice | $5 |
| ITEM1002 | SEL1001 | Organic Lemonade | $4 |
| ITEM1003 | SEL1001 | Organic Almond Milk | $3 |
| ITEM1004 | SEL1001 | Organic Dried Blueberries | $3.25 |
| ITEM1005 | SEL1001 | Organic Pumpkin seeds | $3.30 |
| ITEM1006 | SEL1002 | Parsley | $1.05 |
| ITEM1007 | SEL1002 | Basil | $1 |
| ITEM1008 | SEL1002 | Mint | $1.15 |
| ITEM1009 | SEL1002 | Cilantro | $1.60 |
| ITEM1010 | SEL1002 | Oregano | $1.30 |

| | | | |
|---|---|---|---|
| ITEM1011 | SEL1003 | Potato | $0.89 |
| ITEM1012 | SEL1003 | Celery | $1.15 |
| ITEM1013 | SEL1003 | Cucumber | $0.87 |
| ITEM1014 | SEL1003 | Carrots | $0.65 |
| ITEM1015 | SEL1003 | Tomatoes | $0.95 |
| ITEM1016 | SEL1004 | Green Apple | $0.75 |
| ITEM1017 | SEL1004 | Golden Delicious | $0.95 |
| ITEM1018 | SEL1004 | Strawberries | $4.25 |
| ITEM1019 | SEL1004 | Mangos | $1.70 |
| ITEM1020 | SEL1004 | Pineapples | $4.20 |
| ITEM1020 | SEL1005 | Pineapples | $4.20 |
| ITEM1011 | SEL1005 | Potato | $1 |
| ITEM1014 | SEL1005 | Carrots | $0.89 |
| ITEM1009 | SEL1005 | Cilantro | $2 |
| ITEM1005 | SEL1005 | Organic Pumpkin seeds | $3.30 |

**Seller table**

The Seller table displays seller and store name data. The table includes 5 sellers with a total of 15 values and 3 attributes:

- sell_id (primary key)- unique ID for the seller
- sell_name- name for each seller
- store_name- name for each store

| Seller Table | | |
|---|---|---|
| **sell_id** | sell_name | store_name |
| SEL1001 | Jim West | Organic Lyfe |
| SEL1002 | Dan Smith | Herb's Herbs |
| SEL1003 | Sallie Mae | Vickie for Vegans |
| SEL1004 | Mike Miller | Appleseed Farms |
| SEL1005 | Bill Russ | Founding Farmers |

**Customer table**

The Customer table displays customer data. There are 5 customers with a total of 25 values and 3 attributes:

- cust_id (primary key)- unique ID for the customer
- cust_name- name of each customer
- str_address- street address for each customer for shipping

| Customer Table |
|---|

| **cust_id (PK)** | cust_name | str_address |
|---|---|---|
| CUST101 | Eva Valentine | 2300 West 7th Street |
| CUST102 | Jackson Thurman | 1301 Calico Ave |
| CUST103 | Phelicia Caterino | 422 Colonel Maynard Street |
| CUST104 | P.J. Buckwell | 639 Mabelvale Lane |
| CUST105 | Leon Kennedy | 780 Ferrada Boulevard |

**Customer_Payment table**

The Customer Payment table displays customer, payment, and account information for each customer. There are 5 customer payments with a total of 30 values and 6 attributes:

- pymt_id (primary key)- unique ID for payment
- cust_id (foreign key)- unique ID for the customer; references the customer table
- method- type of card used
- account_no- number for the customer account
- expiration- card expiration date
- code- security code for the card

| customer_payment table | | | | | |
|---|---|---|---|---|---|
| **pymt_id (PK)** | **cust_id (FK)** | method | account_no | expiration | code |
| PYMT1001 | CUST101 | visa | 4270168905630018 | 112022 | 763 |
| PYMT1002 | CUST103 | visa | 4540274978438703 | 102021 | 223 |
| PYMT1003 | CUST102 | visa | 4475578849198236 | 022025 | 129 |
| PYMT1004 | CUST104 | mastercard | 5172049973350224 | 042022 | 239 |
| PYMT1005 | CUST105 | mastercard | 5172893423842039 | 062026 | 101 |

**Orders table**

The orders table displays customer order data such as customer, item, and quantity per order. There are 12 orders with a total of 48 values and 4 attributes:

- order_id (primary key)- unique ID for the order
- item_id- unique ID for the item; references the item table
- cust_id (primary and foreign key)- unique ID for customer; references the customer table
- quantity- item quantity on each order

| orders table |
|---|

| **order_id (PK)** | item_id | **cust_id (PK, FK)** | quantity |
|---|---|---|---|
| PO1001 | ITEM1003 | CUST101 | 1 |
| PO1001 | ITEM1004 | CUST101 | 3 |
| PO1002 | ITEM1005 | CUST102 | 5 |
| PO1002 | ITEM1005 | CUST102 | 2 |
| PO1002 | ITEM1007 | CUST102 | 1 |
| PO1003 | ITEM1008 | CUST103 | 2 |
| PO1004 | ITEM1009 | CUST104 | 4 |
| PO1005 | ITEM1010 | CUST104 | 2 |
| PO1006 | ITEM1011 | CUST105 | 3 |
| PO1006 | ITEM1012 | CUST105 | 1 |
| PO1007 | ITEM1013 | CUST105 | 2 |
| PO1008 | ITEM1003 | CUST105 | 1 |

## SQL DDL Statements

The DDL below is used to create each database table. Each table has primary keys, foreign keys, and each attribute's associate data type.

```
create table customer
  (cust_id       varchar(7),
  cust_name      varchar(30),
  str_address    varchar(50),
  primary key (cust_id)
  );

create table seller
  (sell_id       varchar(7),
  sell_name      varchar(30),
  primary key (sell_id)
  );

create table storefront
  (store_id      varchar(7),
  sell_id        varchar(7),
  store_name     varchar(30),
  primary key (store_id),
  foreign key (sell_id) references seller (sell_id)
  on delete set null
  );

CREATE TABLE item
  (item_id       varchar(8),
  sell_id        varchar(7),
```

```
  item_name       varchar(30),
  price           numeric(3,2),
  primary key (item_id, sell_id),
  foreign key (sell_id) references seller (sell_id)
  on delete cascade
  );

CREATE TABLE orders
  (order_id       varchar(7),
  item_id         varchar(8),
  sell_id         varchar(7),
  cust_id         varchar(7),
  quantity        numeric(10),
  primary key (order_id, cust_id, item_id, sell_id),
  foreign key (cust_id) references customer (cust_id)
  on delete cascade,
  foreign key (item_id, sell_id) references item (item_id, sell_id)
  on delete cascade
  );
```

```
CREATE TABLE customer_payment
  (pymt_id       varchar(8),
  cust_id        varchar(7),
  methods        VARCHAR(20),
  account_no     VARCHAR(50),
  expiration     numeric(6),
  code           numeric(3),
  primary key (pymt_id),
  foreign key (cust_id) references customer (cust_id)
  on delete set null
  );
```

## Verification of Table Creation

| Number | Elapsed | Statement | Feedback | Rows |
|---|---|---|---|---|
| 1 | 0.04 | create table customer (cust_id varchar(7), cust | Table created. | 0 |
| 2 | 0.03 | create table seller (sell_id varchar(7), sell_n | Table created. | 0 |
| 3 | 0.04 | create table storefront -- changed to storefront (store_i | Table created. | 0 |
| 4 | 0.04 | create table item (item_id varchar(8), sell_id | Table created. | 0 |
| 5 | 0.04 | create table orders (order_id varchar(7), item_i | Table created. | 0 |
| 6 | 0.04 | create table customer_payment (pymt_id varchar(8), | Table created. | 0 |

## SQL DML Statements

The DDL statements below are used to insert all values into their respective tables.

Inserting Customer values:

```
Insert into customer values ('CUST101', 'Eva Valentine', '2300 West 7th Street');
insert into customer values ('CUST102', 'Jackson Thurman', '1301 Calico Ave');
insert into customer values ('CUST103', 'Phelicia Caterino', '422 Colonel Maynard Street');
insert into customer values ('CUST104', 'P.J. Buckwell', '639 Mabelvale Lane');
insert into customer values ('CUST105', 'Leon Kennedy', '780 Ferrada Boulevard');
```

Inserting Seller values:

```
insert into seller values ('SEL1001', 'Jim West');
insert into seller values ('SEL1002', 'Dan Smith');
insert into seller values ('SEL1003', 'Sallie Mae');
insert into seller values ('SEL1004', 'Mike Miller');
insert into seller values ('SEL1005', 'Bill Russ');
```

Inserting Storefront values:

```
insert into storefront values ('FMR1001', 'SEL1001', 'Organic Lyfe');
insert into storefront values ('FMR1002', 'SEL1002', 'Herb's Herbs');
insert into storefront values ('FMR1003', 'SEL1003', 'Vickie for Vegans');
insert into storefront values ('FMR1004', 'SEL1004', 'Appleseed Farms');
insert into storefront values ('FMR1005', 'SEL1005', 'Founding Farmers');
```

Inserting Item values:

```
insert into item values ('ITEM1001', 'SEL1001', 'Organic Orange Juice', '5.00');
insert into item values ('ITEM1002', 'SEL1001', 'Organic Lemonade', '4.00');
insert into item values ('ITEM1003', 'SEL1001', 'Organic Almond Milk', '3.00');
insert into item values ('ITEM1004', 'SEL1001', 'Organic Dried Blueberries', '3.25');
```

```
insert into item values ('ITEM1005', 'SEL1001', 'Organic Pumpkin Seeds', '3.30');
insert into item values ('ITEM1006', 'SEL1002', 'Parsley', '1.05');
insert into item values ('ITEM1007', 'SEL1002', 'Basil', '1.00');
insert into item values ('ITEM1008', 'SEL1002', 'Mint', '1.15');
insert into item values ('ITEM1009', 'SEL1002', 'Cilantro', '1.60');
insert into item values ('ITEM1010', 'SEL1002', 'Oregano', '1.30');
insert into item values ('ITEM1011', 'SEL1003', 'Potato', '0.89');
insert into item values ('ITEM1012', 'SEL1003', 'Celery', '1.15');
insert into item values ('ITEM1013', 'SEL1003', 'Cucumber', '0.87');
insert into item values ('ITEM1014', 'SEL1003', 'Carrots', '0.65');
insert into item values ('ITEM1015', 'SEL1003', 'Tomatoes', '0.95');
insert into item values ('ITEM1016', 'SEL1004', 'Green Apple', '0.75');
insert into item values ('ITEM1017', 'SEL1004', 'Golden Delicious', '0.95');
insert into item values ('ITEM1018', 'SEL1004', 'Strawberries', '4.25');
insert into item values ('ITEM1019', 'SEL1004', 'Mangos', '1.70');
insert into item values ('ITEM1020', 'SEL1004', 'Pineapples', '4.20');
insert into item values ('ITEM1020', 'SEL1005', 'Pineapples', '4.20');
insert into item values ('ITEM1011', 'SEL1005', 'Potato', '1.00');
insert into item values ('ITEM1014', 'SEL1005', 'Carrots', '0.89');
insert into item values ('ITEM1009', 'SEL1005', 'Cilantro', '2.00');
insert into item values ('ITEM1005', 'SEL1005', 'Organic Pumpkin Seeds', '3.30');
```

Inserting Orders values:

```
insert into orders values ('PO1001', 'ITEM1003', 'SEL1001','CUST101', '1');
insert into orders values ('PO1001', 'ITEM1004', 'SEL1001', 'CUST101', '3');
insert into orders values ('PO1002', 'ITEM1005', 'SEL1001', 'CUST102', '5');
insert into orders values ('PO1002', 'ITEM1005', 'SEL1005', 'CUST102', '2');
insert into orders values ('PO1002', 'ITEM1007', 'SEL1002', 'CUST102', '1');
insert into orders values ('PO1003', 'ITEM1008', 'SEL1002', 'CUST103', '2');
insert into orders values ('PO1004', 'ITEM1009', 'SEL1005', 'CUST104', '4');
```

```
insert into orders values ('PO1005', 'ITEM1010', 'SEL1002', 'CUST104', '2');
insert into orders values ('PO1006', 'ITEM1011', 'SEL1003', 'CUST105', '3');
insert into orders values ('PO1006', 'ITEM1012', 'SEL1003', 'CUST105', '1');
insert into orders values ('PO1007', 'ITEM1013', 'SEL1003', 'CUST105', '2');
insert into orders values ('PO1008', 'ITEM1003', 'SEL1001', 'CUST105', '1');
```

Inserting Customer_Payment values:

```
insert into customer_payment values
('PYMT1001','CUST101','visa','4270168905630018','112022','763');
insert into customer_payment values
('PYMT1002','CUST102','visa','4540274978438703','102021','223');
insert into customer_payment values
('PYMT1003','CUST103','visa','4475578849198236','022025','129');
insert into customer_payment values
('PYMT1004','CUST104','mastercard','5172049973350224','042022','239');
insert into customer_payment values
('PYMT1005','CUST105','mastercard','5172893423842039','062026','101');
```

## Drop table

The following script is used to drop tables within the database.

```
drop table customer_payment;
drop table orders;
drop table item;
drop table storefront;
drop table seller;
drop table customer;
```

## SQL Queries

The query statement below shows customer orders and other pertinent information from multiple tables.

Query #1- Query 1 displays the total cost of each item ordered multiplied by quantity

```
SELECT o.order_id, o.cust_id, i.item_name, i.price, o.quantity as qty, (i.price*o.quantity) as total_cost
 FROM item i, orders o
 WHERE i.item_id = o.item_id
 ORDER BY o.order_id;
```

Sample Result:

| ORDER_ID | CUST_ID | ITEM_NAME | PRICE | QTY | TOTAL_COST |
|---|---|---|---|---|---|
| PO1001 | CUST101 | Organic Almond Milk | 3 | 1 | 3 |
| PO1001 | CUST101 | Organic Dried Blueberries | 3.25 | 3 | 9.75 |
| PO1002 | CUST102 | Organic Pumpkin Seeds | 3.3 | 5 | 16.5 |
| PO1002 | CUST102 | Organic Pumpkin Seeds | 3.3 | 5 | 16.5 |
| PO1002 | CUST102 | Organic Pumpkin Seeds | 3.3 | 2 | 6.6 |
| PO1002 | CUST102 | Basil | 1 | 1 | 1 |

Query #2- Query 2 is a multi-table query that displays each item, and the seller name, and associated store.

```
SELECT f.store_name, i.item_name, s.sell_name
FROM seller s, item i, storefront f
WHERE  s.sell_id = i.sell_id and f.sell_id = s.sell_id
```

Sample Result:

| STORE_NAME | ITEM_NAME | SELL_NAME |
|---|---|---|
| Organic Lyfe | Organic Orange Juice | Jim West |
| Organic Lyfe | Organic Lemonade | Jim West |
| Organic Lyfe | Organic Almond Milk | Jim West |
| Organic Lyfe | Organic Dried Blueberries | Jim West |
| Organic Lyfe | Organic Pumpkin Seeds | Jim West |
| Herb's Herbs | Parsley | Dan Smith |

Query #3- Query 3 displays the average price per order with a quantity greater than 3, and grouped by item name, order ID, and item ID.

```
SELECT i.item_id, i.item_name, o.order_id, o.quantity, avg(i.price) as average_price
FROM orders o, item i
WHERE i.item_id = o.item_id
GROUP BY i.item_id, i.item_name, o.order_id, o.quantity
HAVING quantity > 3
ORDER BY average_price;
```

Sample Result:

| ITEM_ID | ITEM_NAME | ORDER_ID | QUANTITY | AVERAGE_PRICE |
|---|---|---|---|---|
| ITEM1009 | Cilantro | PO1004 | 4 | 1.8 |
| ITEM1005 | Organic Pumpkin Seeds | PO1002 | 5 | 3.3 |

Query #4- Query 4 returns a summation of the items ordered for Leon Kennedy. It also checks for null values within the order.

```
SELECT sum(quantity)
FROM orders
WHERE cust_id = 'CUST105' and cust_id IS NOT NULL;
```

Sample Result:

| SUM(QUANTITY) |
| --- |
| 7 |

Query #5- Subquery that returns the item with a price greater than Organic Lyfe

```
SELECT item_name, price
FROM item
WHERE sell_id = 'SEL1001' and
                price > (select avg (price)
                FROM item
                WHERE sell_id = 'SEL1001');
```

Sample Result:

| ITEM_NAME | PRICE |
| --- | --- |
| Organic Orange Juice | 5 |
| Organic Lemonade | 4 |

Query #6- returns the item, price, and seller for Potato

```
SELECT i.item_name, i.price, s.sell_name
FROM item i, seller s, orders o
WHERE s.sell_id = i.sell_id and i.item_id = o.item_id and i.item_name = 'Potato'
```

Sample Result:

| ITEM_NAME | PRICE | SELL_NAME |
| --- | --- | --- |
| Potato | .89 | Sallie Mae |
| Potato | 1 | Bill Russ |

Query #7- returns store name, item name, and seller for Organic Lyfe

SELECT f.store_name, i.item_name, s.sell_name

FROM seller s, item i, storefront f

WHERE s.sell_id = i.sell_id and f.sell_id = s.sell_id and f.store_name = 'Organic Lyfe'

Sample Result:

| STORE_NAME | ITEM_NAME | SELL_NAME |
|---|---|---|
| Organic Lyfe | Organic Orange Juice | Jim West |
| Organic Lyfe | Organic Lemonade | Jim West |
| Organic Lyfe | Organic Almond Milk | Jim West |
| Organic Lyfe | Organic Dried Blueberries | Jim West |
| Organic Lyfe | Organic Pumpkin Seeds | Jim West |

Query #8- Returns the order information for purchase order 1001

SELECT o.order_id, o.cust_id, i.item_name, i.price, o.quantity as qty, (i.price*o.quantity) as total_cost

FROM item i, orders o

WHERE i.item_id = o.item_id and order_id = 'PO1001';

Sample Result:

| ORDER_ID | CUST_ID | ITEM_NAME | PRICE | QTY | TOTAL_COST |
|---|---|---|---|---|---|
| PO1001 | CUST101 | Organic Almond Milk | 3 | 1 | 3 |
| PO1001 | CUST101 | Organic Dried Blueberries | 3.25 | 3 | 9.75 |

Query #9- Returns total cost for purchase order 1001

SELECT sum((i.price*o.quantity)) as total_cost

FROM item I, orders o

WHERE i.item_id = o.item_id and o.order_id = 'PO1001';

Sample Result:

| TOTAL_COST |
| --- |
| 12.75 |

Query #10- Returns an average for all Organic Lyfe products

SELECT avg (price) as average_price_of_Organic_Lyfe_products

FROM item

WHERE sell_id = 'SEL1001';

Sample Result:

| AVERAGE_PRICE_OF_ORGANIC_LYFE_PRODUCTS |
| --- |
| 3.71 |

Query #11- Returns the item and price for Organic Lyfe with prices greater than the average price for Organic Lyfe.

SELECT item_name, price

FROM item

WHERE sell_id = 'SEL1001' and price > (SELECT avg (price)

FROM item

WHERE sell_id = 'SEL1001');

Sample Result:

| ITEM_NAME | PRICE |
| --- | --- |
| Organic Orange Juice | 5 |
| Organic Lemonade | 4 |

Query #12- Returns order information with average price with any order quantity greater than 2

```
SELECT i.item_id, i.item_name, o.order_id, o.quantity, avg(i.price) as average_price
FROM orders o, item i
WHERE i.item_id = o.item_id
GROUP BY i.item_id, i.item_name, o.order_id, o.quantity
HAVING quantity > 2
ORDER BY average_price;
```

Sample Result:

| ITEM_ID | ITEM_NAME | ORDER_ID | QUANTITY | AVERAGE_PRICE |
|---|---|---|---|---|
| ITEM1011 | Potato | PO1006 | 3 | .945 |
| ITEM1009 | Cilantro | PO1004 | 4 | 1.8 |
| ITEM1004 | Organic Dried Blueberries | PO1001 | 3 | 3.25 |
| ITEM1005 | Organic Pumpkin Seeds | PO1002 | 5 | 3.3 |

Query #13- Returns the 3 invoices for Customer Leon Kennedy with subtotal as final for the order.

```
SELECT o.cust_id, o.order_id, i.item_name, sum(i.price * o.quantity) as total_price
FROM orders o, item i
WHERE o.item_id = i.item_id and o.cust_id = 'CUST105'
GROUP BY ROLLUP (o.cust_id, o.order_id, i.item_name);
```

Sample Result:

| CUST_ID | ORDER_ID | ITEM_NAME | TOTAL_PRICE |
|---|---|---|---|
| CUST105 | PO1006 | Celery | 1.15 |
| CUST105 | PO1006 | Potato | 5.67 |
| CUST105 | PO1006 | - | 6.82 |
| CUST105 | PO1007 | Cucumber | 1.74 |
| CUST105 | PO1007 | - | 1.74 |
| CUST105 | PO1008 | Organic Almond Milk | 3 |

# References

Bagui, S., & Earp, R. (2011). The basic ER diagram: A data modeling schema. *Database Design Using Entity-Relationship Diagrams*, 95–128. https://doi.org/10.1201/9781439861776-8

Goelman, D., & Dietrich, S. W. (2018). A visual introduction to conceptual database design for all. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. https://doi.org/10.1145/3159450.3159555