Final Project

Dan Kim

Overview

Our family business, which distributes textile products to the Middle East, has been running for more than 30 years. Although I am not quite sure exactly when our database was built, I know it is at least 20 years old. Since then, there have been a lot of changes in our business model, and I have felt the need for redesigning or upgrading our database to reflect the company's current operations. This project would be a great reference for the actual redevelopment.

I'm guessing that the current database was developed in SQL, but I do not have any access to it. I only have access to the software, which can query results such as the list of orders, items, payments, customers, etc. There are many things that the current software cannot handle, so we have managed those in separate Excel sheets, which has been very cumbersome for us.

For this project, I will use the relational database because there are many attributes that can be grouped together and normalized, which then can reduce redundancies in the database. Also, I will be dealing with roughly tens of thousands of records, which is relatively not huge. Considering that there have been no more than 500 new orders per year, we would not worry too much about the scalability as the data growth will not be overwhelming, but the relationships among tables will be designed simply so that the database can scale to any size. My focus should be on the general structure/relationship of the entities in the database since the actual database would contain a lot more tables and fields than this project presents.

MySQL will be used throughout this project.

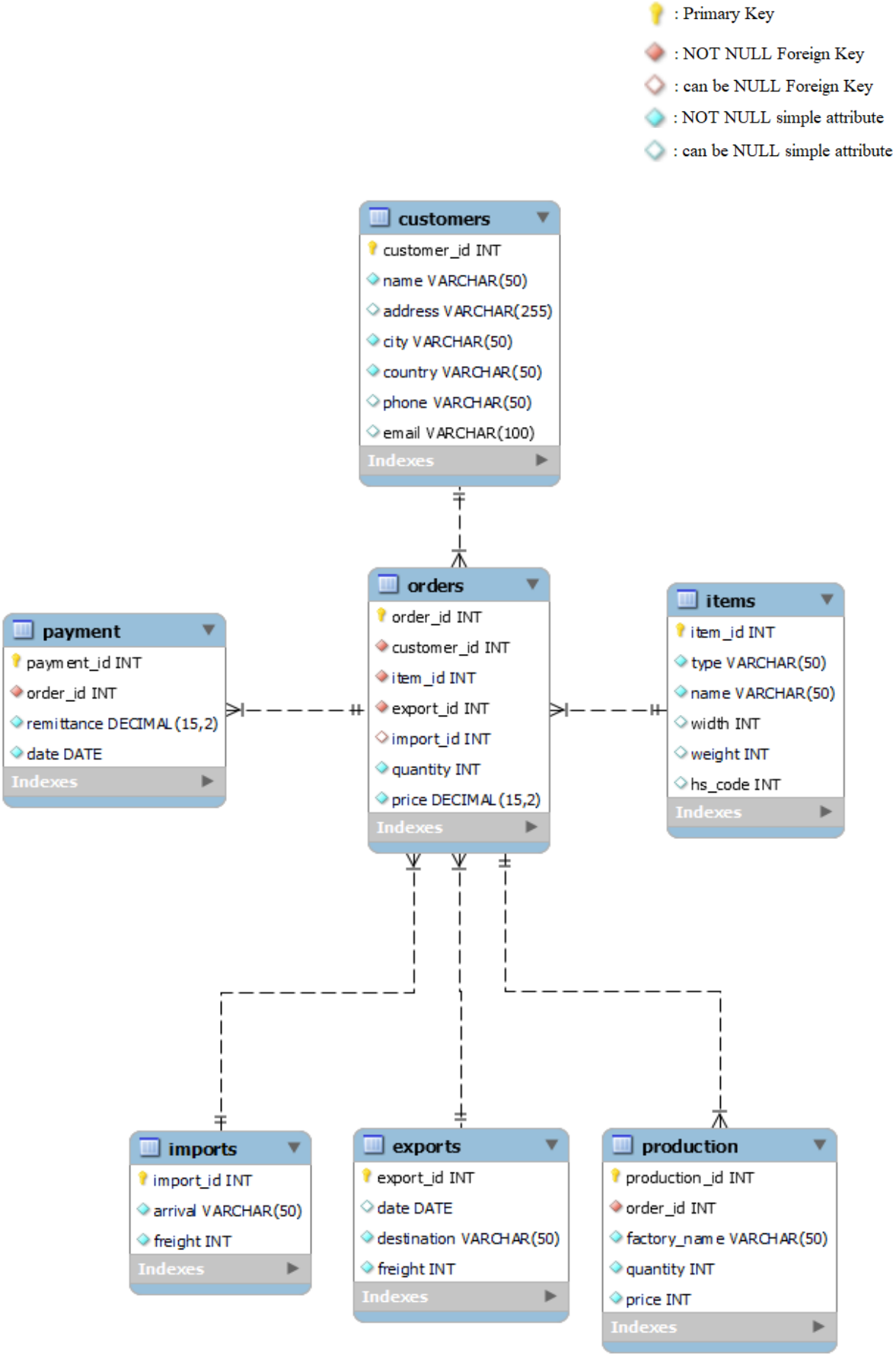
Video presentation link: https://youtu.be/wukSk9J02RI

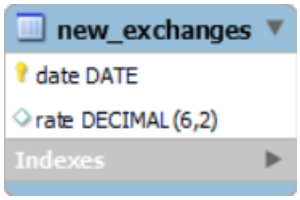
Disclaimer

The data used in this project are randomly generated. Since the actual data in our company contains a lot of confidential information, I decided not to use those. Manually making arbitrary changes to the actual data would have been optimal, but I could not find a way that would not be too time-consuming although the random generation also took an unexpectedly long time to make it seem reasonable. Some numbers in this randomly generated data might be unrealistic, but I believe it is in an acceptable range for the queries used in this project.

Schema

The actual database would be much complicated than what is presented here, but for the purpose of this project, I narrowed it down to the most significant entities and attributes. Please refer to the diagram in the next page for more details.


: Primary Key
: NOT NULL Foreign Key
: can be NULL Foreign Key
: NOT NULL simple attribute
: can be NULL simple attribute
customers
customer_id INT
name VARCHAR(50)
address VARCHAR(255)
city VARCHAR(50)
country VARCHAR(50)
phone VARCHAR(50)
email VARCHAR(100)
Indexes
orders
order_id INT
customer_id INT
item_id INT
export_id INT
import_id INT
quantity INT
price DECIMAL(15,2)
Indexes
payment
payment_id INT
order_id INT
remittance DECIMAL(15,2)
date DATE
Indexes
items
item_id INT
type VARCHAR(50)
name VARCHAR(50)
width INT
weight INT
hs_code INT
Indexes
imports
import_id INT
arrival VARCHAR(50)
freight INT
Indexes
exports
export_id INT
date DATE
destination VARCHAR(50)
freight INT
Indexes
production
production_id INT
order_id INT
factory_name VARCHAR(50)
quantity INT
price INT
Indexes



(The details of this table are explained later in this document)

*** Creating the Database**

```
CREATE DATABASE dankim_corporation;
USE dankim_corporation;
```

*** Creating the Tables**

- The tables need to be created in the order shown below.

```
CREATE TABLE customers (

  customer_id       INT NOT NULL AUTO_INCREMENT,
  name              VARCHAR(50) NOT NULL UNIQUE,
  address           VARCHAR(255),
  city              VARCHAR(50) NOT NULL,
  country           VARCHAR(50) NOT NULL,
  phone             VARCHAR(50) UNIQUE,
  email             VARCHAR(100) UNIQUE,
  PRIMARY KEY       (customer_id)
);
```

- 50 rows
- Primary key: *customer_id*

```
CREATE TABLE items (

  item_id           INT NOT NULL AUTO_INCREMENT,
  type              VARCHAR(50) NOT NULL,
  name              VARCHAR(50) NOT NULL UNIQUE,
  width             INT,
  weight            INT,
  hs_code           INT,
  PRIMARY KEY       (item_id)
);
```

- 60 rows
- Primary key: *item_id*

```sql
CREATE TABLE orders (

  order_id          INT NOT NULL AUTO_INCREMENT,
  customer_id       INT NOT NULL,
  item_id           INT NOT NULL,
  export_id         INT NOT NULL,
  import_id         INT,
  quantity          INT NOT NULL,
  price             DECIMAL(15,2) NOT NULL,
  PRIMARY KEY       (order_id),
  CONSTRAINT        customers
    FOREIGN KEY     (customer_id)
    REFERENCES      customers(customer_id)
    ON DELETE CASCADE,
  CONSTRAINT        items
    FOREIGN KEY     (item_id)
    REFERENCES      items(item_id)
    ON DELETE CASCADE,
  CONSTRAINT        exports
    FOREIGN KEY     (export_id)
    REFERENCES      exports(export_id)
    ON DELETE CASCADE,
  CONSTRAINT        imports
    FOREIGN KEY     (import_id)
    REFERENCES      imports(import_id)
    ON DELETE CASCADE
);
```

- 1,000 rows
- *import_id* can be *NULL* because some items are produced locally.
- *price* means the selling price in USD.
- Primary key: *order_id*
- Foreign keys:

  customer_id, item_id, export_id, import_id

```sql
CREATE TABLE payment (

  payment_id        INT NOT NULL AUTO_INCREMENT,
  order_id          INT NOT NULL,
  remittance        DECIMAL(15,2) NOT NULL,
  date              DATE NOT NULL,
  PRIMARY KEY       (payment_id),
  CONSTRAINT        orders
    FOREIGN KEY     (order_id)
    REFERENCES      orders(order_id)
    ON DELETE CASCADE
);
```

- 2,500 rows
- One or more *payment_id* can be linked to one *order_id* since customers often make multiple payments for one order.
- *remittance* means the received amount from the customers in USD.
- Primary key: *payment_id*
- Foreign key: *order_id*

```sql
CREATE TABLE exports (

  export_id         INT NOT NULL AUTO_INCREMENT,
  date              DATE,
  destination       VARCHAR(50) NOT NULL,
  freight           INT NOT NULL,
  PRIMARY KEY       (export_id)
);
```

- 250 rows
- Primary key: *export_id*
- One *export_id* can be linked to multiple *order_id.*
- *date* means the shipped date (arbitrarily created for the last 3 years).
- *freight* is in KRW (Korean Won).

```sql
CREATE TABLE imports (

  import_id         INT NOT NULL AUTO_INCREMENT,
  arrival           VARCHAR(50) NOT NULL,
  freight           INT NOT NULL,
  PRIMARY KEY       (import_id)
);
```

- 70 rows
- Primary key: *import_id*
- One *import_id* can be linked to multiple *order_id.*
- *freight* is in KRW.

```sql
CREATE TABLE production (

  production_id     INT NOT NULL AUTO_INCREMENT,
  order_id          INT NOT NULL,
  factory_name      VARCHAR(50) NOT NULL,
  quantity          INT NOT NULL,
  price             INT NOT NULL,
  PRIMARY KEY       (production_id),
  CONSTRAINT        production_orders
    FOREIGN KEY     (order_id)
    REFERENCES      orders(order_id)
    ON DELETE CASCADE
);
```

- 2,300 rows
- Primary key: *production_id*
- Multiple *production_id* can be linked to one *order_id* because the production usually takes place in more than one factory sequentially.
- *price* is in KRW.
- Foreign key: *order_id*

*** Exchange rate**

Since the selling price and the payment received are in USD and the costs (freight for exports and imports, price for production) in KRW, we need a table for exchange rates between USD and KRW.

```
CREATE TABLE exchanges (

  date              DATE NOT NULL UNIQUE,
  rate              DECIMAL(6,2) NOT NULL,
  PRIMARY KEY       (date)
);

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/exchange_rate.csv'
INTO TABLE dankim_corporation.exchanges
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

exchange_rate.csv is the actual data for the last 3 years taken from Hana bank in Korea (www.kebhana.com), and it is stored in the table called *exchanges*. However, the data does not exist for weekends and holidays, so if *export.date* is one of those days, the relevant exchange rate would not be available. To resolve this issue, I created another table called *datefile*, which contains just dates for the last 3 years without any missing dates.

(*exchange_rate.csv* and *date.csv* are also attached for reference)

```
CREATE TABLE datefile (

  date              DATE NOT NULL UNIQUE
);

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/date.csv'
INTO TABLE dankim_corporation.datefile
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

This *datefile* table is LEFT JOINed with *exchanges*, and for the weekends and holidays, the exchange rate is filled with that of the last working day.

```
CREATE TABLE new_exchanges AS
    SELECT
      datefile.date AS date,
      CASE WHEN exchanges.rate IS NULL THEN @prev ELSE @prev := exchanges.rate END AS rate
    FROM datefile
    LEFT JOIN exchanges
      ON datefile.date = exchanges.date
    ORDER BY datefile.date ASC;
```

In result, we now have all the exchanges rates for the last 3 years (stored in *new_exchanges*). This table is not connected to any of other tables because it is unnecessary, but we must make sure that the dates are unique in the tables used for the exchange rates.

(In addition to the above, I will have to find a way to automatically retrieve from the web and update the daily exchange rate into the database for the purpose of the scalability. It is not done here because it is out of scope of this course.)

Queries

1. General Information about each Order

```
SELECT
  orders.order_id AS order_no,
  customers.name AS customer_name,
  items.name AS item_name,
  orders.quantity AS quantity,
  orders.price AS price,
  orders.quantity * orders.price AS amount,
  exports.date AS shipped_date
FROM orders
JOIN customers
  USING (customer_id)
JOIN items
  USING (item_id)
JOIN exports
  USING (export_id)
ORDER BY order_no ASC;
```

- Top 5 results

order_no	customer_name	item_name	quantity	price	amount	shipped_date
1	Botsford Group	alias	8822	4.58	40404.76	2020-01-22
2	Koelpin, Kshlerin and Kuphal	reiciendis	198541	3.90	774309.90	2019-05-27
3	Schmitt-Ziemann	ut	89704	4.64	416226.56	2021-12-13
4	Konopelski-Erdman	voluptatem	73644	1.56	114884.64	2019-05-10
5	Hessel, Schamberger and West	dolorum	128225	1.40	179515.00	2021-03-08

2. Current Balance for each Order

```
SELECT
  orders.order_id AS order_no,
  orders.quantity * orders.price AS invoice_amount,
  SUM(payment.remittance) AS received_amount,
  SUM(payment.remittance) - orders.quantity * orders.price AS balance
FROM orders
JOIN payment
  USING (order_id)
GROUP BY order_no
ORDER BY order_no ASC;
```

- Top 5 results

order_no	invoice_amount	received_amount	balance
1	40404.76	102648.00	62243.24
2	774309.90	95704.00	-678605.90
3	416226.56	88327.00	-327899.56
4	114884.64	44545.00	-70339.64
5	179515.00	88459.00	-91056.00

(balance: (-) means the amount to be received)

3. Current Balance for each Customer

The temporary table below is the same as the query#2 above.

```
CREATE TEMPORARY TABLE tbl2 (
    SELECT
      orders.order_id AS order_no,
      orders.quantity * orders.price AS invoice_amount,
      SUM(payment.remittance) AS received_amount,
      SUM(payment.remittance) - orders.quantity * orders.price AS balance
    FROM orders
    JOIN payment
      USING (order_id)
    GROUP BY order_no
    ORDER BY order_no ASC);
```

Now, using *tbl2*, we can make joins with *orders* and *customers* tables to get what we want.

```
SELECT
  customers.name AS customer_name,
  SUM(tbl2.balance) AS balance
FROM tbl2
JOIN orders
  ON tbl2.order_no = orders.order_id
JOIN customers
  ON orders.customer_id = customers.customer_id
GROUP BY customers.name;
```

- Top 5 results

customer_name	balance
Botsford Group	-3955747.75
Koelpin, Kshlerin and Kuphal	-3749927.31
Schmitt-Ziemann	-3613436.24
Konopelski-Erdman	-3982180.61
Hessel, Schamberger and West	-4190825.01

4-1. Monthly Sales

```
SELECT
  YEAR(exports.date) as year,
  MONTH(exports.date) as month,
  SUM(orders.quantity * orders.price) AS amount
FROM orders
JOIN exports
  USING (export_id)
WHERE exports.date IS NOT NULL
GROUP BY month, year
ORDER BY exports.date ASC;
```

- Results

year	month	amount
2019	5	5273292.94
2019	6	5022859.63
2019	7	9274230.17
2019	8	6115128.41
2019	9	2191945.15
2019	10	7538834.95
2019	11	6284135.08
2019	12	10171025.14

year	month	amount
2020	1	11856656.26
2020	2	10238134.95
2020	3	12122224.61
2020	4	12246143.50
2020	5	9198291.32
2020	6	9718526.12
2020	7	6946137.82
2020	8	6788974.58
2020	9	7904123.39
2020	10	5247685.50
2020	11	6681762.45
2020	12	4105134.26

year	month	amount
2021	1	7274693.56
2021	2	5243248.43
2021	3	8790190.09
2021	4	3284401.65
2021	5	10030195.89
2021	6	9235425.04
2021	7	4337200.99
2021	8	5363558.40
2021	9	6827568.64
2021	10	9141068.61
2021	11	5570930.54
2021	12	7110329.77

year	month	amount
2022	1	9141599.28
2022	2	5210149.21
2022	3	3543628.48
2022	4	9121144.29
2022	5	1189801.98

4-2. Yearly Sales

```
SELECT
  YEAR(exports.date) as year,
  SUM(orders.quantity * orders.price) AS amount
FROM orders
JOIN exports
  USING (export_id)
WHERE exports.date IS NOT NULL
GROUP BY year
ORDER BY year ASC;
```

- Results

year	amount
2019	51871451.47
2020	103053794.76
2021	82208811.61
2022	28206323.24

5. Cost and Profit for each Order

```
SELECT
  order_id AS order_no,
  (orders.quantity * orders.price) AS amount,
  exports.freight / new_exchanges.rate AS export_cost,
  imports.freight / new_exchanges.rate AS import_cost,
  SUM(production.quantity * production.price / new_exchanges.rate) AS production_cost,
  (orders.quantity * orders.price) - exports.freight/new_exchanges.rate - imports.freight/new_exchanges.rate -
  SUM(production.quantity * production.price / new_exchanges.rate) AS profit
FROM orders
JOIN exports
  USING (export_id)
JOIN imports
  USING (import_id)
JOIN production
  USING (order_id)
JOIN new_exchanges
  ON exports.date = new_exchanges.date
GROUP BY orders.order_id
ORDER BY orders.order_id;
```

- Top 5 results

order_no	amount	export_cost	import_cost	production_cost	profit
1	40404.76	4194.6156	475.5326	26165.1355	9569.4763
2	774309.90	3793.3270	2021.9377	296148.0425	472346.5928
3	416226.56	7008.1740	948.4078	271603.2509	136666.7272
4	114884.64	7843.4673	3150.1102	71911.0794	31979.9832
5	179515.00	6793.3904	1092.6153	229675.7014	-58046.7071

6. Find the Width, Weight, and HS_Code for Item (using stored procedure)

```
DELIMITER $$

CREATE PROCEDURE findSpec (
  IN itemName VARCHAR(50),
  OUT oWidth INT,
  OUT oWeight INT,
  OUT oHs_code INT)

    BEGIN
      SELECT
        width, weight, hs_code INTO oWidth, oWeight, oHs_code
      FROM items
      WHERE name = itemName;

    END $$

DELIMITER ;
```

- Find the information for the item named ‘illum’.

```
CALL findSpec("illum", @oWidth, @oWeight, @oHs_code);
SELECT @oWidth, @oWeight, @oHs_code;
```

@oWidth	@oWeight	@oHs_code
56	341	540752

7. Find the Most Recent Selling Price for Item

```
DELIMITER $$

CREATE FUNCTION findPrice(itemName VARCHAR(50))
RETURNS DECIMAL(6,2) DETERMINISTIC

BEGIN
  DECLARE oPrice DECIMAL(6,2);

  IF itemName IS NOT NULL AND itemName != ''
  THEN
        SELECT
              orders.price INTO oPrice
            FROM orders
            JOIN exports
              USING (export_id)
            RIGHT JOIN
                (SELECT
                  items.item_id,
                  items.name AS item_name,
                  MAX(exports.date) AS date
                FROM orders
                JOIN items
                  USING (item_id)
                JOIN exports
                  USING (export_id)
                GROUP BY items.item_id) AS max_date_by_item
            ON orders.item_id = max_date_by_item.item_id
              AND exports.date = max_date_by_item.date
            WHERE item_name = itemName;
  ELSE SET oPrice = -1;
  END IF;
```

```
  RETURN oPrice;

END $$

DELIMITER ;
```

- Find the most recent selling price for the item named ‘illum’.

```
SELECT findPrice('illum');
```

findPrice('illum')
2.91