Exercise Week 6 (Mentoring 3)

SQL & Relational Database

By : Oktavio Reza Putra

1. The Manager need your help to develop database design
   - The application currently comprises a single main store. However, the manager wants to develop a feature that allows the application to open additional new stores. It means the application should store information about the list of shops that will be opened and the product related to each store. So, the application can manage and display products correctly for each shop *(Hint: Beside create a new table, you need to modify products table)*
   - Additionally, there is currently no shopping cart feature in this application. The manager wants to add a shopping cart functionality. Enabling customers to store items they intend to purchase.
   - Can you also suggest any other features that should be considered?

   Now, you can proceed with your SQL syntax and provide a description of the new tables, and if you have any additional feature suggestions, feel free to mention them as well.

   Answer Here

   - **Description**

Table Name	Description	Primary Key	Foreign Keys	Unique Constraint and Additional Info
customers	This table contains data about customers of the digital media store. Customer information, such as name, address, and email, is stored in this table.	customer_id	None	None
orders	Records customer orders, linking each order to a customer, along with payment information, order date, and delivery date and linking each orders to payment type id.	order_id	customer_id from customers(customer_id), payment_type_id from payment_type_table(payment_type_id)	None
products	This table stores details about products, including their ID, type, name, size, color, price, quantity, and description. The primary key is "product_id	product_id	None	None
sales	This table contains data about employees of the digital media store. Staff information, such as name, position, hiring date, and more, can be found in this table.	sales_id	order_id from orders(order_id), product_id from products(product_id)	total_price can be calculated as price_per_unit * quantity

Table Name	Description	Primary Key	Foreign Keys	Unique Constraints and Additional Info
list_area_store_manager	Contains details about area managers overseeing multiple stores.	area_manager_id	None	manager_name, email, and asm_contact_person are unique identifiers. tenure_days must be non-zero and non-negative.
list_store_manager	Stores information on store managers who manage individual stores.	store_manager_id	None	store_manager_name, email, and sm_contact_person are unique identifiers. tenure_days must be non-zero and non-negative.
store_territory	Stores information on the territories of each store,	territory_id	None	territory_id represents unique territory identifiers (e.g., "1A", “1B”, etc).

Table Name	Description	Primary Key	Foreign Keys	Unique Constraints and Additional Info
	including province, city, and postal code.			
stores	Contains store details, including store name, address, associated managers, territory info, etc.	store_id	area_manager_id from list_area_store_manager, store_manager_id from list_store_manager, territory_id from store_territory	store_name, store_email are unique. is_active indicates store status (active/inactive).
linking_store_products	Links products to stores, indicating the availability of each product at each store.	product_id, store_id	product_id from products, store_id from stores	Ensures unique store-product combinations by composite key.
shopping_cart	Stores items that customers intend to purchase, including product quantity and last added item.	cart_id	product_id from products	quantity must more than 0.
transaction_status_type	List of Status Type	status_type_id		Unique Status Type ID
transaction_status	Tracks the status of orders (e.g., "Pending," "Paid", etc.) throughout the transaction journey.	status_order_id	order_id from orders, status_type_id from transaction_status_type	status_type_id and order_id must positive number
payment_type_table	Lists all available payment methods (e.g., BCA, Mandiri).	payment_type_id	None	payment_type must be unique and only includes specific values (e.g., "BCA", "Mandiri").
customer_payment_method	Stores payment details for customer orders, tracking payment type and amount.	payment_order_id	customer_id from customers, payment_type_id from payment_type_table	payment_order_id, payment_amount, customer_id, payment_type_id must be positive.
customer_review	Allows customers to leave reviews and ratings for purchased products.	review_id	customer_id from customers	rating values are between 1 and 5.
loyalty_points	Tracks loyalty points that customers earn, which can be used for future discounts.	loyalty_id	customer_id from customers	points must be non-negative.
product_views	Logs viewed products to recommend items based on customers' browsing history.	view_id	customer_id from customers, product_id from products	view_date tracks when a product was viewed.
security_settings	Stores security settings, such as two-factor authentication, for each customer.	setting_id	customer_id from customers	two_factor_enabled is a boolean (true or false). recovery_email provides an additional contact method for security.
customer_preferences	Stores customer preferences, such as favorite categories or subscription status.	preference_id	customer_id from customers	newsletter_subscribed is a boolean, indicating if the customer receives newsletters.
vouchers	Manages discount codes and vouchers that customers can apply to orders.	voucher_id	customer_id from customers, order_id from orders	discount_code is unique and must be valid (not expired). is_used is a boolean that indicates if the voucher has been redeemed.

- **Table Structure**

customers				
customer_id	:	integer		Primary Key (PK)
customer_name	:	varchar(50)		
gender	:	varchar(50)		
age	:	integer		
home_address	:	varchar(50)		
zip_code	:	integer		
city	:	varchar(50)		
state	:	varchar(50)		
country	:	varchar(50)		

orders				
order_id	:	integer		Primary Key (PK)
customer_id	:	integer		Foreign Key (FK) referencing customers(customer_id)
payment	:	integer		
order_date	:	varchar(50)		
delivery_date	:	varchar(50)		
payment_type_id	:	integer		Foreign Key (FK) referencing payment_type_table(payment_type_id)

products				
product_id	:	integer		Primary Key (PK)
product_type	:	varchar(100)		
product_name	:	varchar(100)		
size	:	varchar(100)		
colour	:	varchar(100)		
price	:	integer		
quantity	:	integer		
description	:	varchar(100)		

sales				
sales_id	:	integer		Primary Key (PK)
order_id	:	integer		Foreign Key (FK) referencing orders(order_id)
product_id	:	integer		Foreign Key (FK) referencing products(product_id)
price_per_unit	:	integer		
quantity	:	integer		
total_price	:	integer		

list_area_store_manager				
area_manager_id	:	serial		Primary Key (PK)
manager_name	:	varchar(255)	Not Null, Unique	Candidate Key (CK)
hire_date	:	date		
tenure_days	:	int	Not Null, Check (tenure_days > 0)	
email	:	varchar(100)	Not Null, Unique	Candidate Key (CK)
asm_contact_person	:	varchar(15)	Not Null, Unique	Candidate Key (CK)

list_store_manager				
store_manager_id	:	serial		Primary Key (PK)
store_manager_name	:	varchar(255)	Not Null, Unique	Candidate Key (CK)
hire_date	:	date		
tenure_days	:	int	Not Null, Check (tenure_days > 0)	
email	:	varchar(100)	Not Null, Unique	Candidate Key (CK)
sm_contact_person	:	varchar(15)	Not Null, Unique	Candidate Key (CK)

store_territory				
territory_id	:	varchar(10)		Primary Key (PK)
province	:	varchar(50)	Not Null	

city	:	varchar(200)	Not Null	
sub_district	:	varchar(200)		
postal_code	:	varchar(10)	Not Null	

stores				
store_id	:	serial		Primary Key (PK)
store_name	:	varchar(100)	Not Null, Unique	
opening_date	:	date		
open_hours	:	time		
close_hours	:	time		
address	:	text		
territory_id	:	varchar(20)	Not Null	Foreign Key (FK), References store_territory(territory_id)
official_store_phone_no	:	varchar(15)	Not Null	
store_email	:	varchar(100)	Not Null, Unique	Candidate Key (CK)
area_manager_id	:	int	Not Null, check(area_manager_id > 0)	Foreign Key (FK), References list_area_store_manager(area_manager_id)
store_manager_id	:	int	Not Null, Unique, check(store_manager_id > 0)	Foreign Key (FK), References list_store_manager(store_manager_id)
store_description	:	text		
is_active	:	smallint	Default 1	
create_at	:	timestamp	Default current_timestamp	
update_at	:	timestamp	Default current_timestamp	

linking_store_products				
product_id	:	int	check(product_id >= 0)	Composite Primary Key (CPK), Foreign Key (FK), References products(product_id)
store_id	:	int	check(store_id > 0)	Composite Primary Key (CPK), Foreign Key (FK), References stores(store_id)

shopping_cart				
cart_id	:	int		Primary Key (PK)
product_id	:	int	Not Null, check(product_id >= 0)	Foreign Key (FK), References products(product_id)
product_quantity_in_shopping_cart	:	int	Not Null, check(product_quantity_in_shopping_cart > 0)	
last_added_at	:	timestamp	Default current_timestamp	

transaction_status_type		
status_type_id : serial		Primary Key (PK)
status_type_name : Varchar(10)	Not Null, check(order_status in ('Cancelled', 'Pending', 'Paid', 'Returned', 'Delivered', 'Shipped')	

transaction_status		
status_order_id : varchar(100)		Primary Key (PK)
status_type_id : int	check(status_type_id > 0)	Foreign Key (FK), References transaction_status_type(status_type_id)
order_id : int	check(order_id > 0)	Foreign Key (FK), References orders(order_id)
status_description : text		
status_date : date		

payment_type_table		
payment_type_id : int		Primary Key (PK)
payment_type : varchar(50)	Not Null, Unique, Check for specific values **'BCA'**,**'Mandiri'**,**'PayLater'**,**'BRI'**,**'BNI'**,**'BSI'**	

customer_payment_method		
payment_order_id : int	check(payment_order_id > 0)	Primary Key (PK)
customer_id : int	Not Null, check(customer_id > 0)	Foreign Key (FK), References customers(customer_id)
order_date : date		
payment_type_id : int	Not Null, check(payment_type_id > 0)	Foreign Key (FK), References payment_type_table(payment_type_id)
payment_type : varchar(50)	Not Null, check payment_type in **('BCA'**,**'Mandiri'**,**'PayLater'**,**'BRI'** ,**'BNI'**,**'BSI')**	Specific value check
payment_amount : numeric	Not Null, check (payment_amount > 0)	Check (payment_amount > 0)

customer_review		
review_id : serial		Primary Key (PK)
order_id : int	Not Null, check(order_id > 0)	
customer_id : int	Not Null, check(customer_id > 0)	Foreign Key (FK), References customers(customer_id)
rating : int	Check (rating between 1 and 5)	

review_text	:	text		
review_date	:	timestamp	Default current_timestamp	

loyalty_points				
loyalty_id	:	serial		Primary Key (PK)
customer_id	:	int	Not Null, check(customer_id > 0)	Foreign Key (FK), References customers(customer_id)
points	:	int	Default 0, Check (points >= 0)	
last_updated	:	timestamp	Default current_timestamp	

product_views				
view_id	:	serial		Primary Key (PK)
customer_id	:	int	Not Null, check(customer_id > 0)	Foreign Key (FK), References customers(customer_id)
product_id	:	int	Not Null, check(product_id >= 0)	Foreign Key (FK), References products(product_id)
view_date	:	timestamp	Default current_timestamp	

security_settings				
setting_id	:	serial		Primary Key (PK)
customer_id	:	int	Not Null, check(customer_id > 0)	Foreign Key (FK), References customers(customer_id)
two_factor_enabled	:	boolean	Default FALSE	
recovery_email	:	varchar(100)		

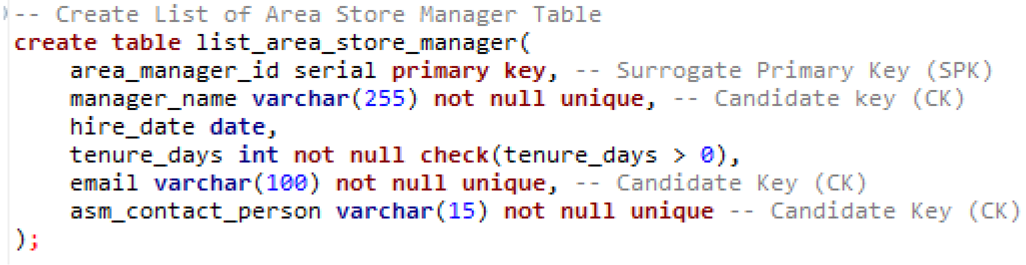
customer_preferences				
preference_id	:	serial		Primary Key (PK)
customer_id	:	int	Not Null, check(customer_id > 0)	Foreign Key (FK), References customers(customer_id)
favorite_category	:	varchar(100)		
newsletter_subscribed	:	boolean	Default FALSE	

vouchers				
voucher_id	:	serial		Primary Key (PK)
customer_id	:	int	Not Null, check(customer_id > 0)	Foreign Key (FK), References customers(customer_id)
order_id	:	int	Not Null, check(order_id > 0)	Foreign Key (FK), References orders(order_id)
discount_code	:	varchar(100)	Not Null, Unique	

discount_percentage	:	decimal(5,2)		
expiration_date	:	date	Not Null	
is_used	:	boolean	Default false	

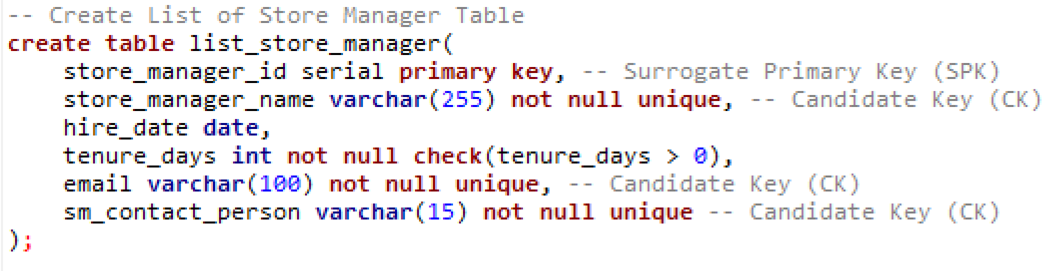
- **DDL**

1. Area Store Manager Table



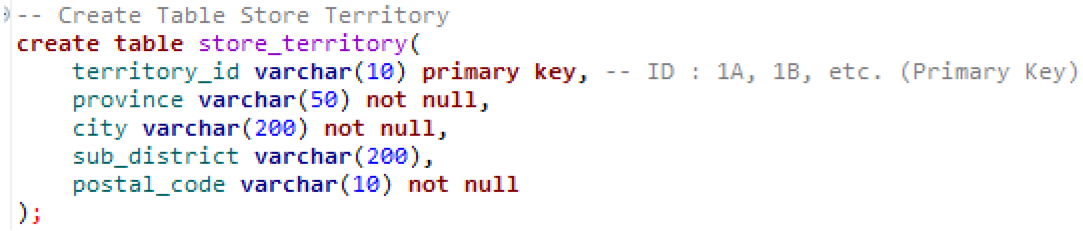
```
-- Create List of Area Store Manager Table
create table list_area_store_manager(
    area_manager_id serial primary key, -- Surrogate Primary Key (SPK)
    manager_name varchar(255) not null unique, -- Candidate key (CK)
    hire_date date,
    tenure_days int not null check(tenure_days > 0),
    email varchar(100) not null unique, -- Candidate Key (CK)
    asm_contact_person varchar(15) not null unique -- Candidate Key (CK)
);
```

2. Store Manager Table



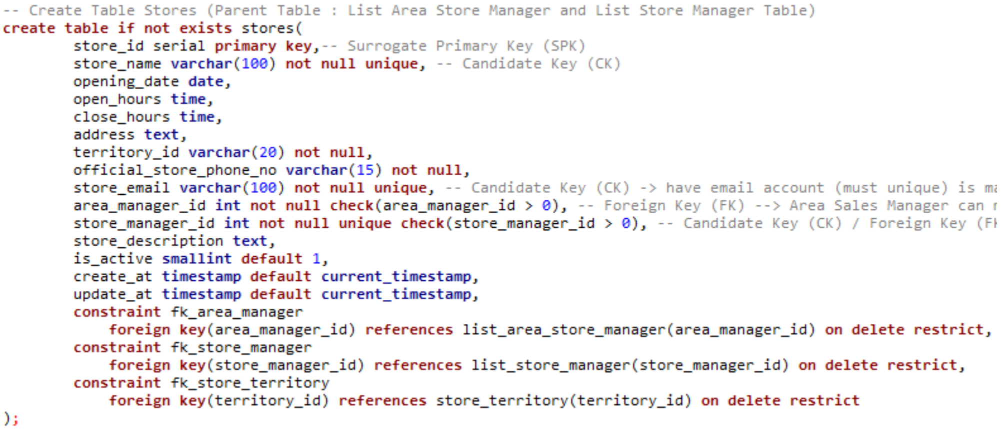
```
-- Create List of Store Manager Table
create table list_store_manager(
    store_manager_id serial primary key, -- Surrogate Primary Key (SPK)
    store_manager_name varchar(255) not null unique, -- Candidate Key (CK)
    hire_date date,
    tenure_days int not null check(tenure_days > 0),
    email varchar(100) not null unique, -- Candidate Key (CK)
    sm_contact_person varchar(15) not null unique -- Candidate Key (CK)
);
```

3. Store Territory Table



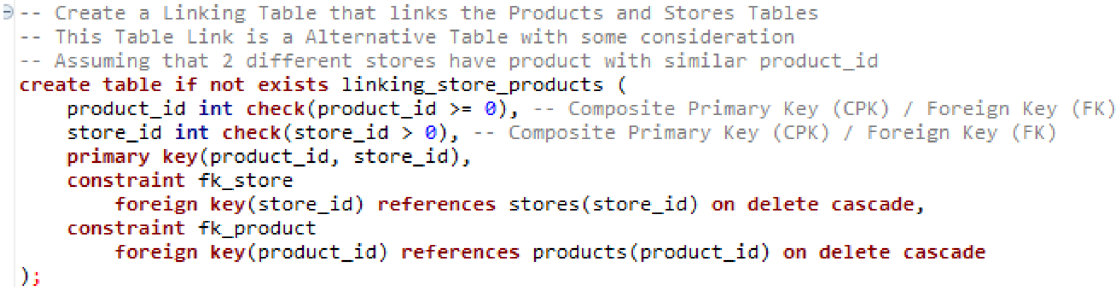
```
-- Create Table Store Territory
create table store_territory(
    territory_id varchar(10) primary key, -- ID : 1A, 1B, etc. (Primary Key)
    province varchar(50) not null,
    city varchar(200) not null,
    sub_district varchar(200),
    postal_code varchar(10) not null
);
```

4. Stores Table



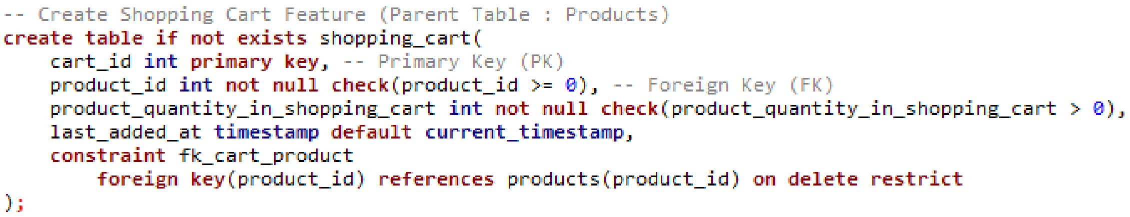
```
-- Create Table Stores (Parent Table : List Area Store Manager and List Store Manager Table)
create table if not exists stores(
        store_id serial primary key,-- Surrogate Primary Key (SPK)
        store_name varchar(100) not null unique, -- Candidate Key (CK)
        opening_date date,
        open_hours time,
        close_hours time,
        address text,
        territory_id varchar(20) not null,
        official_store_phone_no varchar(15) not null,
        store_email varchar(100) not null unique, -- Candidate Key (CK) -> have email account (must unique) is ma
        area_manager_id int not null check(area_manager_id > 0), -- Foreign Key (FK) --> Area Sales Manager can r
        store_manager_id int not null unique check(store_manager_id > 0), -- Candidate Key (CK) / Foreign Key (F
        store_description text,
        is_active smallint default 1,
        create_at timestamp default current_timestamp,
        update_at timestamp default current_timestamp,
        constraint fk_area_manager
            foreign key(area_manager_id) references list_area_store_manager(area_manager_id) on delete restrict,
        constraint fk_store_manager
            foreign key(store_manager_id) references list_store_manager(store_manager_id) on delete restrict,
        constraint fk_store_territory
            foreign key(territory_id) references store_territory(territory_id) on delete restrict
);
```

5. Table Linking Stores and Products



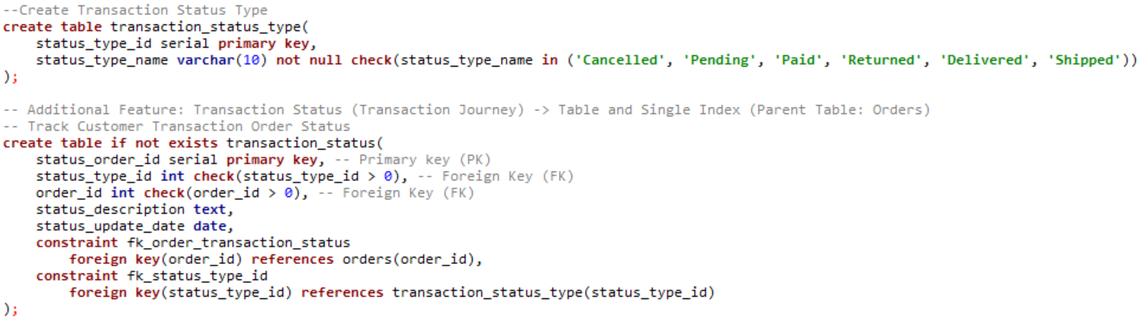
```sql
-- Create a Linking Table that links the Products and Stores Tables
-- This Table Link is a Alternative Table with some consideration
-- Assuming that 2 different stores have product with similar product_id
create table if not exists linking_store_products (
    product_id int check(product_id >= 0), -- Composite Primary Key (CPK) / Foreign Key (FK)
    store_id int check(store_id > 0), -- Composite Primary Key (CPK) / Foreign Key (FK)
    primary key(product_id, store_id),
    constraint fk_store
        foreign key(store_id) references stores(store_id) on delete cascade,
    constraint fk_product
        foreign key(product_id) references products(product_id) on delete cascade
);
```

6. Shopping Cart Table



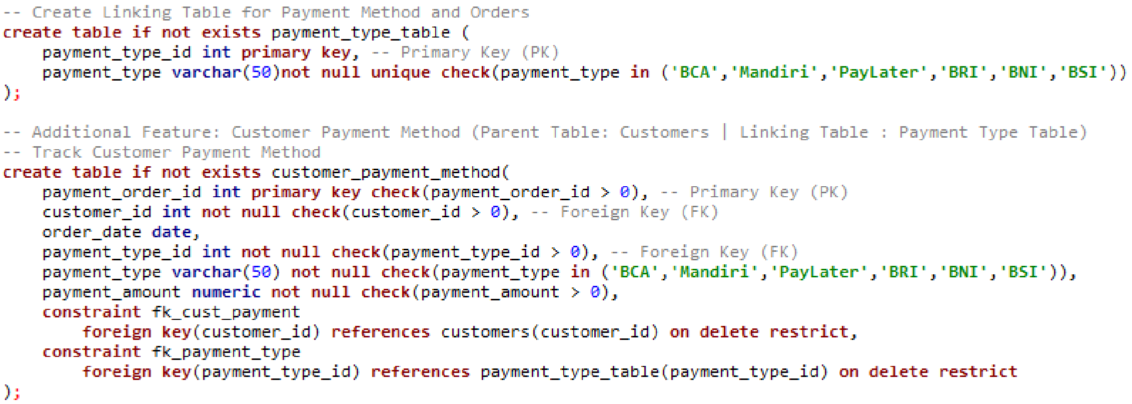
```sql
-- Create Shopping Cart Feature (Parent Table : Products)
create table if not exists shopping_cart(
    cart_id int primary key, -- Primary Key (PK)
    product_id int not null check(product_id >= 0), -- Foreign Key (FK)
    product_quantity_in_shopping_cart int not null check(product_quantity_in_shopping_cart > 0),
    last_added_at timestamp default current_timestamp,
    constraint fk_cart_product
        foreign key(product_id) references products(product_id) on delete restrict
);
```

7. Status Tracker Table (2 Tables)

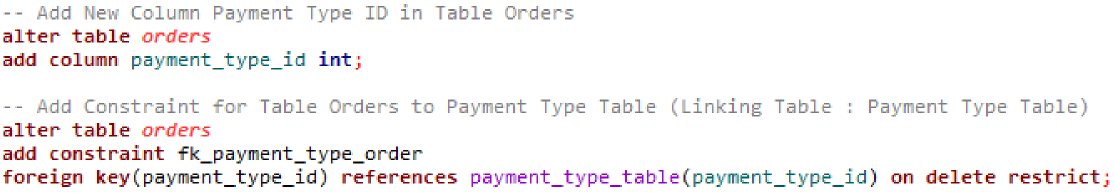


```sql
--Create Transaction Status Type
create table transaction_status_type(
    status_type_id serial primary key,
    status_type_name varchar(10) not null check(status_type_name in ('Cancelled', 'Pending', 'Paid', 'Returned', 'Delivered', 'Shipped'))
);

-- Additional Feature: Transaction Status (Transaction Journey) -> Table and Single Index (Parent Table: Orders)
-- Track Customer Transaction Order Status
create table if not exists transaction_status(
    status_order_id serial primary key, -- Primary key (PK)
    status_type_id int check(status_type_id > 0), -- Foreign Key (FK)
    order_id int check(order_id > 0), -- Foreign Key (FK)
    status_description text,
    status_update_date date,
    constraint fk_order_transaction_status
        foreign key(order_id) references orders(order_id),
    constraint fk_status_type_id
        foreign key(status_type_id) references transaction_status_type(status_type_id)
);
```

8. Payment Table (2 Tables)

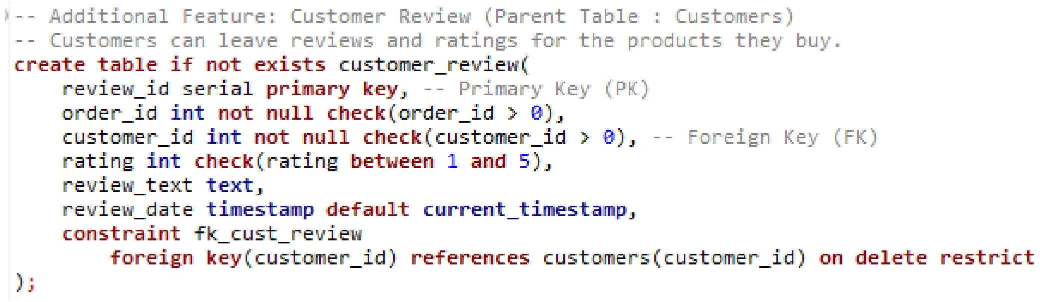


```sql
-- Create Linking Table for Payment Method and Orders
create table if not exists payment_type_table (
    payment_type_id int primary key, -- Primary Key (PK)
    payment_type varchar(50)not null unique check(payment_type in ('BCA','Mandiri','PayLater','BRI','BNI','BSI'))
);

-- Additional Feature: Customer Payment Method (Parent Table: Customers | Linking Table : Payment Type Table)
-- Track Customer Payment Method
create table if not exists customer_payment_method(
    payment_order_id int primary key check(payment_order_id > 0), -- Primary Key (PK)
    customer_id int not null check(customer_id > 0), -- Foreign Key (FK)
    order_date date,
    payment_type_id int not null check(payment_type_id > 0), -- Foreign Key (FK)
    payment_type varchar(50) not null check(payment_type in ('BCA','Mandiri','PayLater','BRI','BNI','BSI')),
    payment_amount numeric not null check(payment_amount > 0),
    constraint fk_cust_payment
        foreign key(customer_id) references customers(customer_id) on delete restrict,
    constraint fk_payment_type
        foreign key(payment_type_id) references payment_type_table(payment_type_id) on delete restrict
);
```



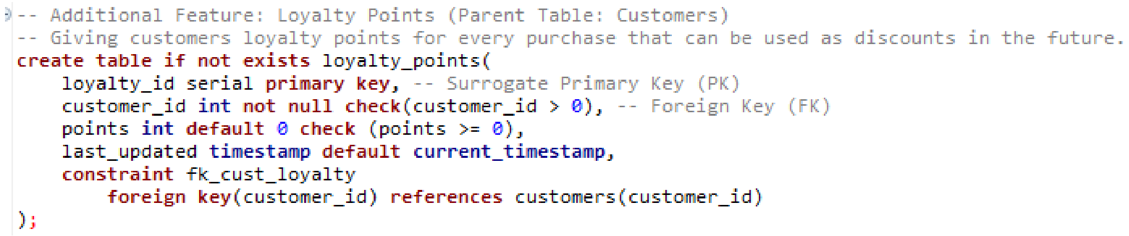
```sql
-- Add New Column Payment Type ID in Table Orders
alter table orders
add column payment_type_id int;

-- Add Constraint for Table Orders to Payment Type Table (Linking Table : Payment Type Table)
alter table orders
add constraint fk_payment_type_order
foreign key(payment_type_id) references payment_type_table(payment_type_id) on delete restrict;
```

9. Customer Review Table



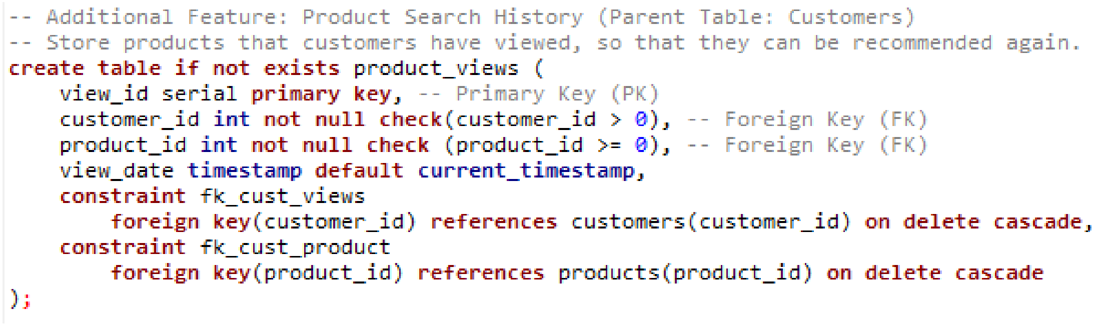
```sql
-- Additional Feature: Customer Review (Parent Table : Customers)
-- Customers can leave reviews and ratings for the products they buy.
create table if not exists customer_review(
    review_id serial primary key, -- Primary Key (PK)
    order_id int not null check(order_id > 0),
    customer_id int not null check(customer_id > 0), -- Foreign Key (FK)
    rating int check(rating between 1 and 5),
    review_text text,
    review_date timestamp default current_timestamp,
    constraint fk_cust_review
        foreign key(customer_id) references customers(customer_id) on delete restrict
);
```

10. Loyalty Point Table



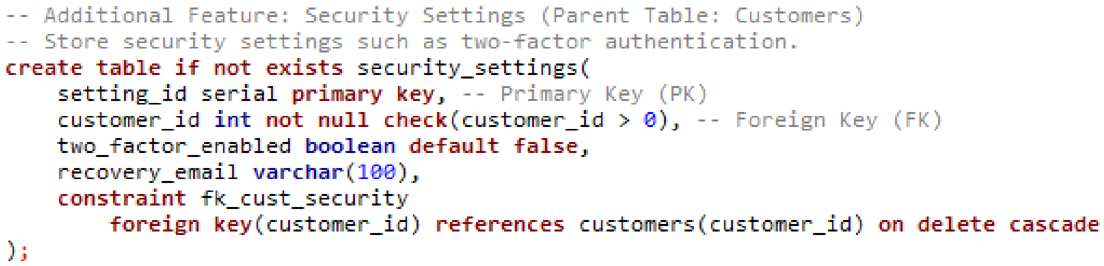
```sql
-- Additional Feature: Loyalty Points (Parent Table: Customers)
-- Giving customers loyalty points for every purchase that can be used as discounts in the future.
create table if not exists loyalty_points(
    loyalty_id serial primary key, -- Surrogate Primary Key (PK)
    customer_id int not null check(customer_id > 0), -- Foreign Key (FK)
    points int default 0 check (points >= 0),
    last_updated timestamp default current_timestamp,
    constraint fk_cust_loyalty
        foreign key(customer_id) references customers(customer_id)
);
```

11. Product Search History Table



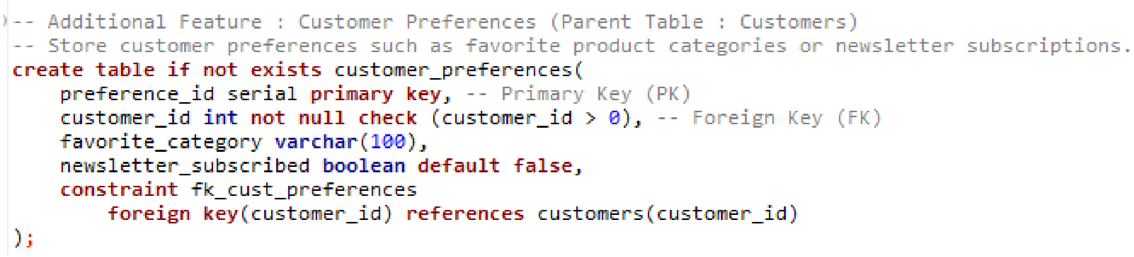
```sql
-- Additional Feature: Product Search History (Parent Table: Customers)
-- Store products that customers have viewed, so that they can be recommended again.
create table if not exists product_views (
    view_id serial primary key, -- Primary Key (PK)
    customer_id int not null check(customer_id > 0), -- Foreign Key (FK)
    product_id int not null check (product_id >= 0), -- Foreign Key (FK)
    view_date timestamp default current_timestamp,
    constraint fk_cust_views
        foreign key(customer_id) references customers(customer_id) on delete cascade,
    constraint fk_cust_product
        foreign key(product_id) references products(product_id) on delete cascade
);
```

12. Security Settings



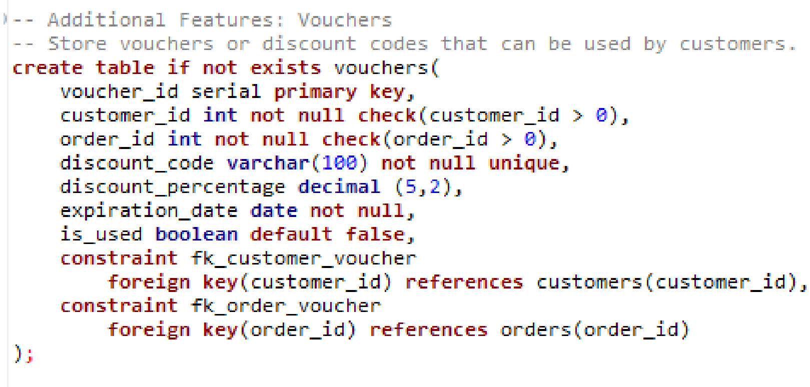
```sql
-- Additional Feature: Security Settings (Parent Table: Customers)
-- Store security settings such as two-factor authentication.
create table if not exists security_settings(
    setting_id serial primary key, -- Primary Key (PK)
    customer_id int not null check(customer_id > 0), -- Foreign Key (FK)
    two_factor_enabled boolean default false,
    recovery_email varchar(100),
    constraint fk_cust_security
        foreign key(customer_id) references customers(customer_id) on delete cascade
);
```

13.Customer Preferences Table



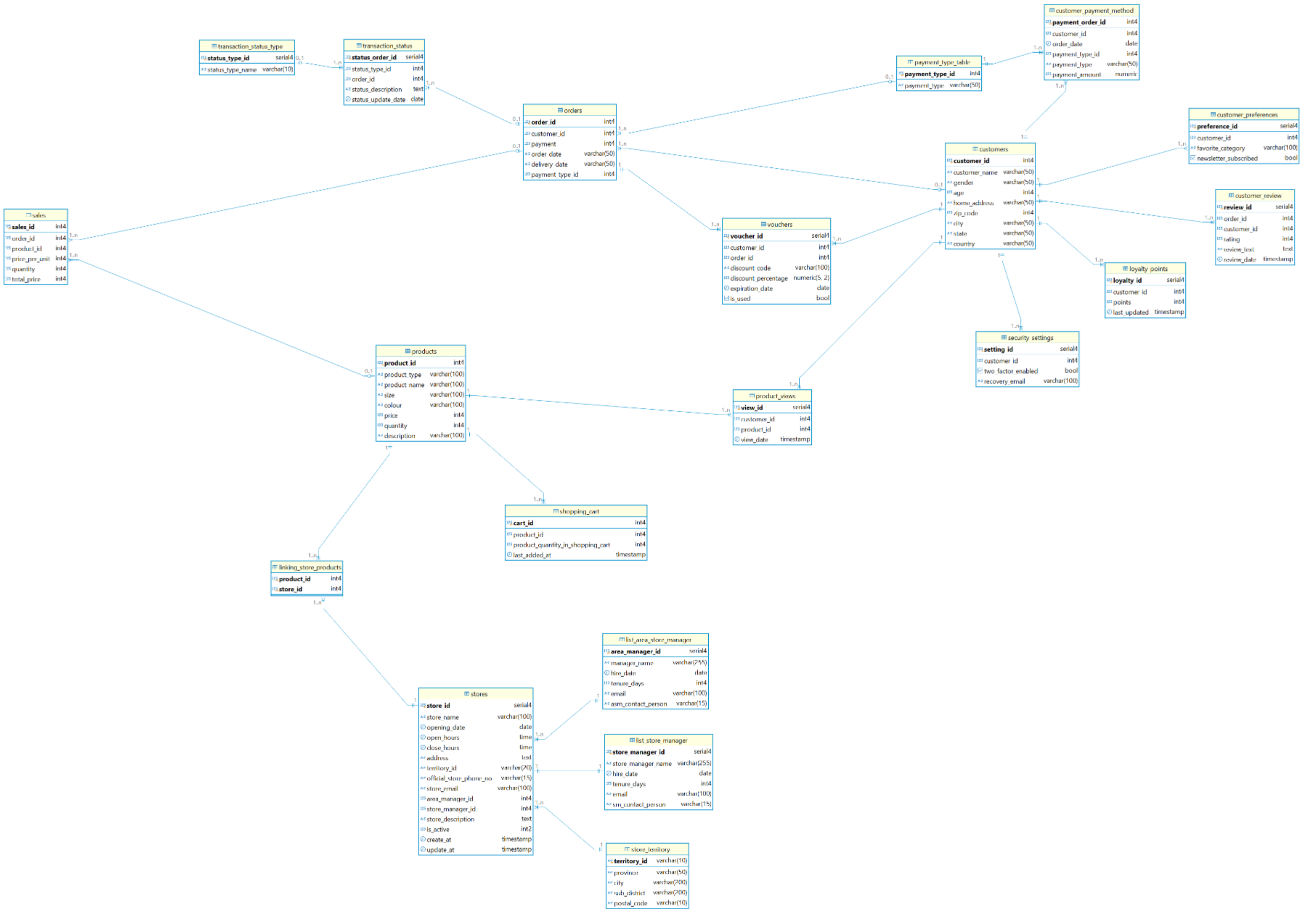
```
-- Additional Feature : Customer Preferences (Parent Table : Customers)
-- Store customer preferences such as favorite product categories or newsletter subscriptions.
create table if not exists customer_preferences(
    preference_id serial primary key, -- Primary Key (PK)
    customer_id int not null check (customer_id > 0), -- Foreign Key (FK)
    favorite_category varchar(100),
    newsletter_subscribed boolean default false,
    constraint fk_cust_preferences
        foreign key(customer_id) references customers(customer_id)
);
```

14. Vouchers



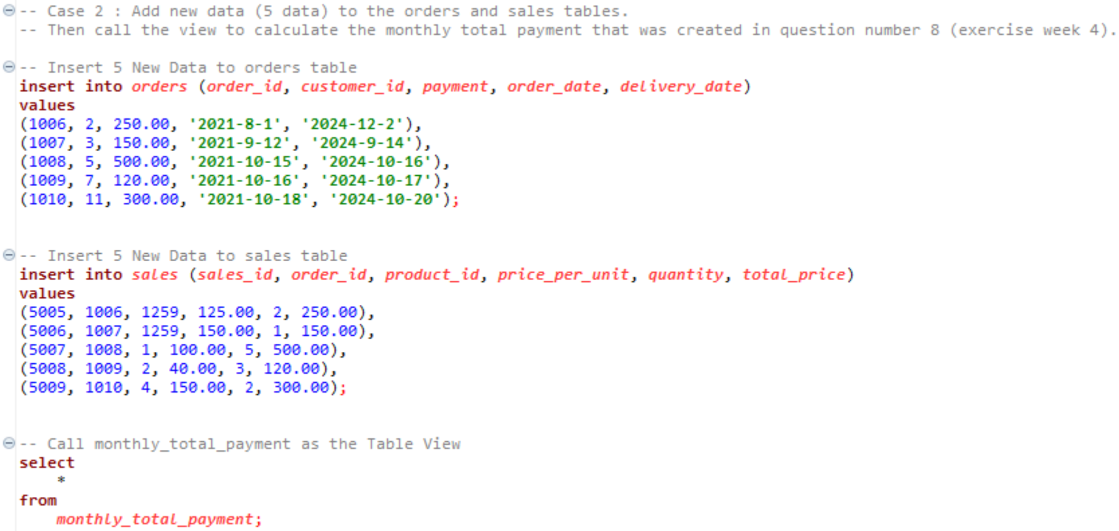
```
-- Additional Features: Vouchers
-- Store vouchers or discount codes that can be used by customers.
create table if not exists vouchers(
    voucher_id serial primary key,
    customer_id int not null check(customer_id > 0),
    order_id int not null check(order_id > 0),
    discount_code varchar(100) not null unique,
    discount_percentage decimal (5,2),
    expiration_date date not null,
    is_used boolean default false,
    constraint fk_customer_voucher
        foreign key(customer_id) references customers(customer_id),
    constraint fk_order_voucher
        foreign key(order_id) references orders(order_id)
);
```

- **ERD**



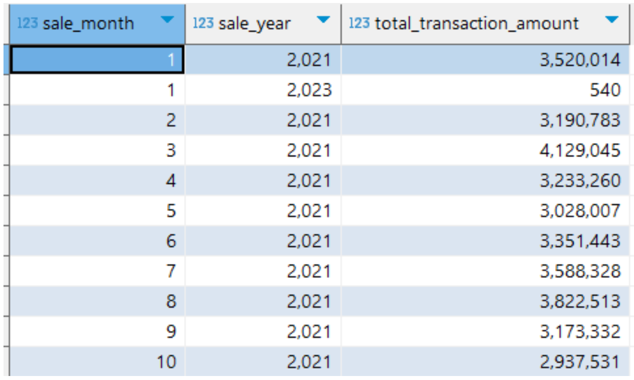
2. Add new data (5 data) to the orders and sales tables. Then call the view to calculate the monthly total payment that was created in question number 8 (exercise week 4). Check whether the query results change according to the new data entered.

   - **SQL Query Syntax:**



```
-- Case 2 : Add new data (5 data) to the orders and sales tables.
-- Then call the view to calculate the monthly total payment that was created in question number 8 (exercise week 4).

-- Insert 5 New Data to orders table
insert into orders (order_id, customer_id, payment, order_date, delivery_date)
values
(1006, 2, 250.00, '2021-8-1', '2024-12-2'),
(1007, 3, 150.00, '2021-9-12', '2024-9-14'),
(1008, 5, 500.00, '2021-10-15', '2024-10-16'),
(1009, 7, 120.00, '2021-10-16', '2024-10-17'),
(1010, 11, 300.00, '2021-10-18', '2024-10-20');


-- Insert 5 New Data to sales table
insert into sales (sales_id, order_id, product_id, price_per_unit, quantity, total_price)
values
(5005, 1006, 1259, 125.00, 2, 250.00),
(5006, 1007, 1259, 150.00, 1, 150.00),
(5007, 1008, 1, 100.00, 5, 500.00),
(5008, 1009, 2, 40.00, 3, 120.00),
(5009, 1010, 4, 150.00, 2, 300.00);


-- Call monthly_total_payment as the Table View
select
    *
from
    monthly_total_payment;
```

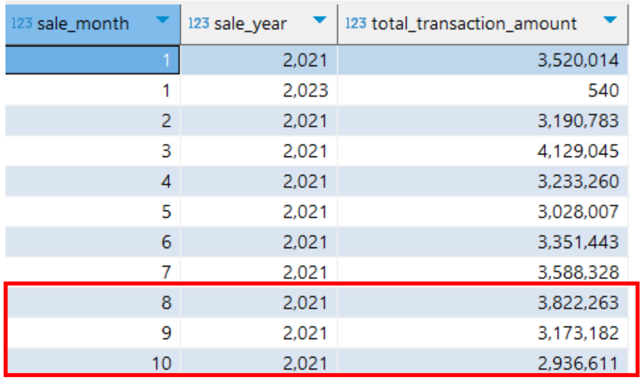
   - **Screenshot of Query Results:**



sale_month	sale_year	total_transaction_amount
1	2,021	3,520,014
1	2,023	540
2	2,021	3,190,783
3	2,021	4,129,045
4	2,021	3,233,260
5	2,021	3,028,007
6	2,021	3,351,443
7	2,021	3,588,328
8	2,021	3,822,513
9	2,021	3,173,332
10	2,021	2,937,531

   - **Description of Query Results:**

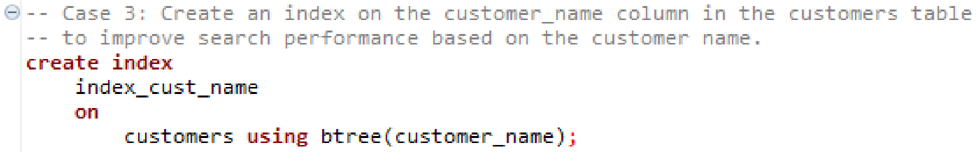
     (Table Exercise Week 4)



sale_month	sale_year	total_transaction_amount
1	2,021	3,520,014
1	2,023	540
2	2,021	3,190,783
3	2,021	4,129,045
4	2,021	3,233,260
5	2,021	3,028,007
6	2,021	3,351,443
7	2,021	3,588,328
8	2,021	3,822,263
9	2,021	3,173,182
10	2,021	2,936,611

After the new data is entered in the Transactions of August, September and October, there is a difference in the numbers in the Exercise Week 4 and Exercise Week 6 tables.

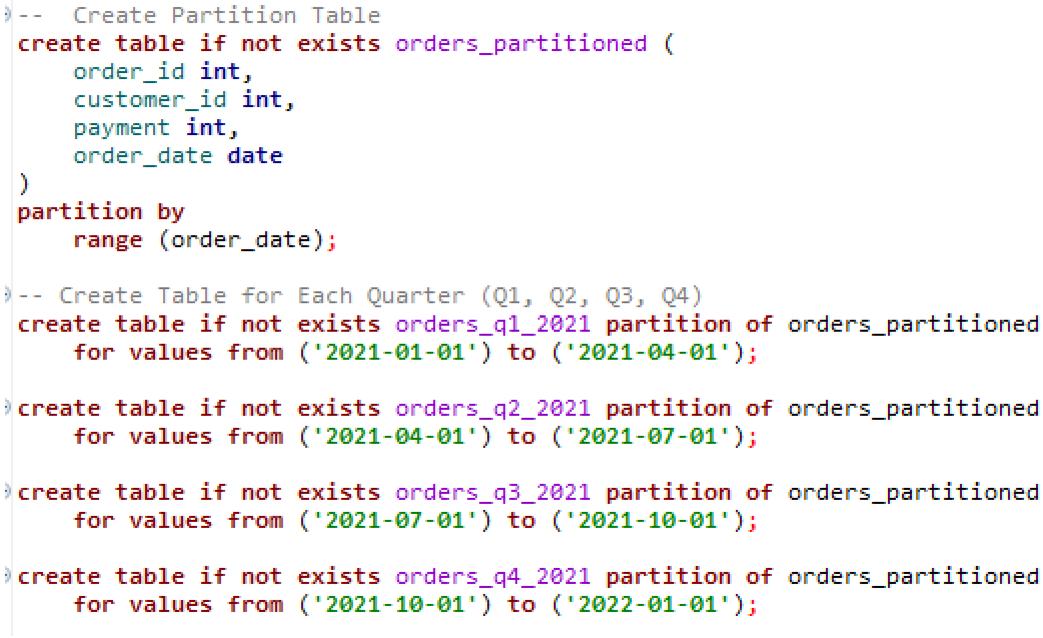
3. (10 point) Create an index on the customer_name column in the customers table to improve search performance based on the customer name.
   Answer Here
   - **Syntax**



```
-- Case 3: Create an index on the customer_name column in the customers table
-- to improve search performance based on the customer name.
create index
    index_cust_name
    on
        customers using btree(customer_name);
```

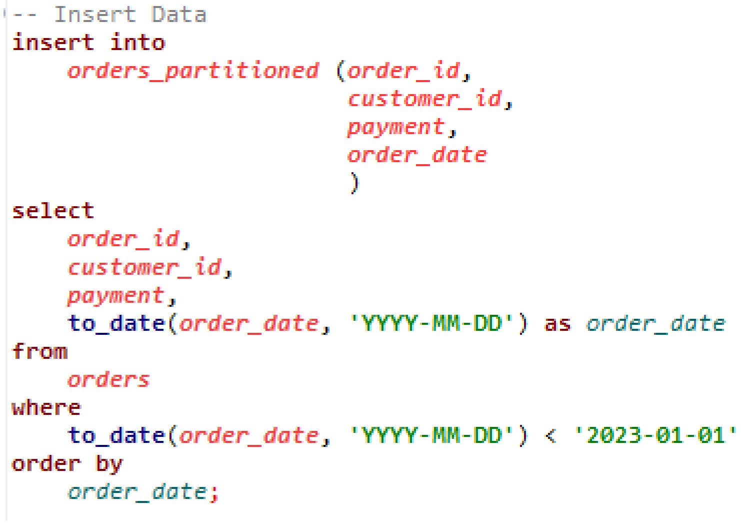
4. Partition the orders table based on the order_date column to enhance query performance involving date ranges. For instance, frequent searches often require filtering by date ranges, and partitioning can significantly optimize such queries. (for example: Q1,Q2,Q3,Q4 from 2021)
   - Create partition tables and insert data from the orders table.
   - Do query from partition table

   Answer Here
   - **Syntax: Create Partition**



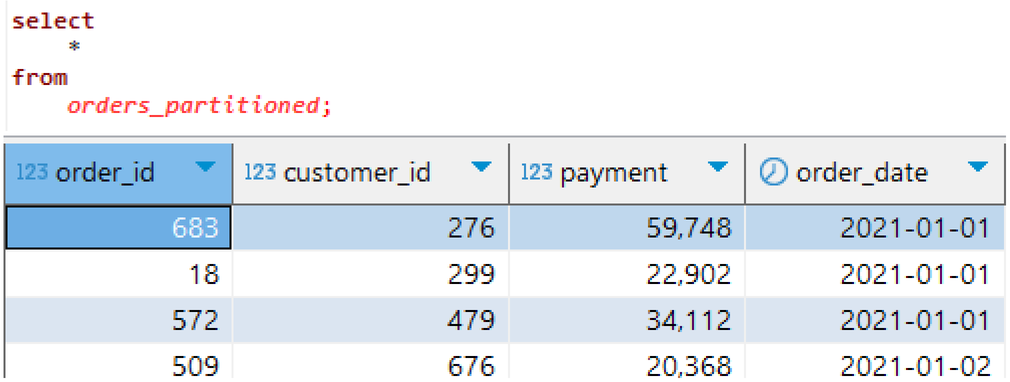
```
-- Create Partition Table
create table if not exists orders_partitioned (
    order_id int,
    customer_id int,
    payment int,
    order_date date
)
partition by
    range (order_date);

-- Create Table for Each Quarter (Q1, Q2, Q3, Q4)
create table if not exists orders_q1_2021 partition of orders_partitioned
    for values from ('2021-01-01') to ('2021-04-01');

create table if not exists orders_q2_2021 partition of orders_partitioned
    for values from ('2021-04-01') to ('2021-07-01');

create table if not exists orders_q3_2021 partition of orders_partitioned
    for values from ('2021-07-01') to ('2021-10-01');

create table if not exists orders_q4_2021 partition of orders_partitioned
    for values from ('2021-10-01') to ('2022-01-01');
```

   - **Syntax: Insert Data**



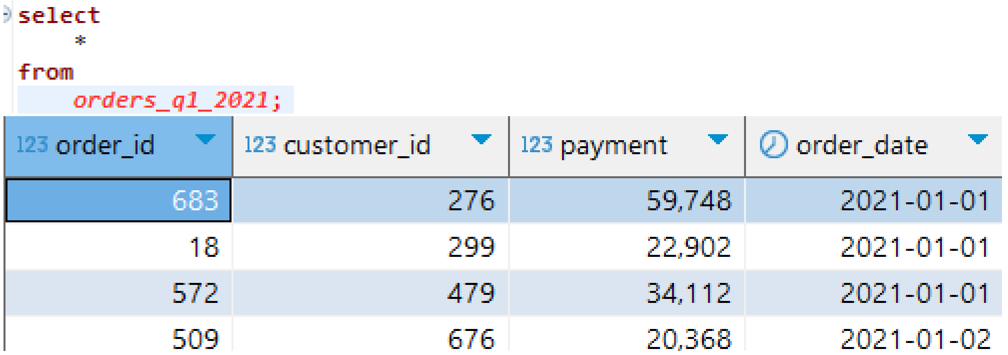
```
-- Insert Data
insert into
    orders_partitioned (order_id,
                        customer_id,
                        payment,
                        order_date
                        )
select
    order_id,
    customer_id,
    payment,
    to_date(order_date, 'YYYY-MM-DD') as order_date
from
    orders
where
    to_date(order_date, 'YYYY-MM-DD') < '2023-01-01'
order by
    order_date;
```

- **Syntax: Select from partition table**



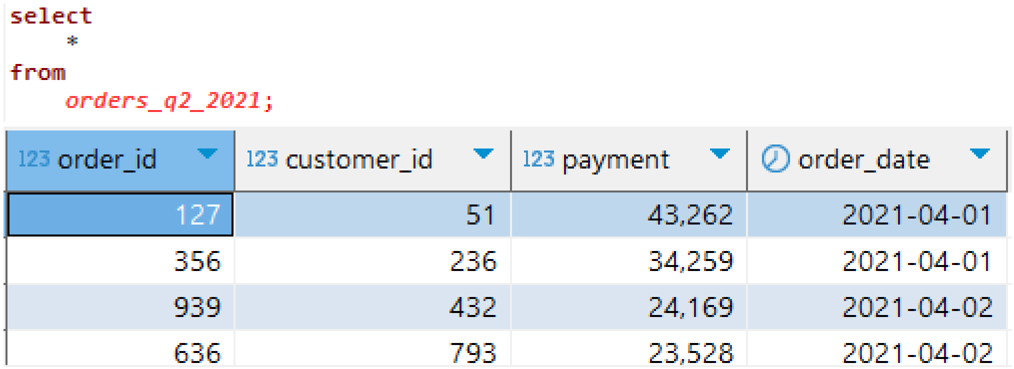
```
select
    *
from
    orders_partitioned;
```

order_id	customer_id	payment	order_date
683	276	59,748	2021-01-01
18	299	22,902	2021-01-01
572	479	34,112	2021-01-01
509	676	20,368	2021-01-02



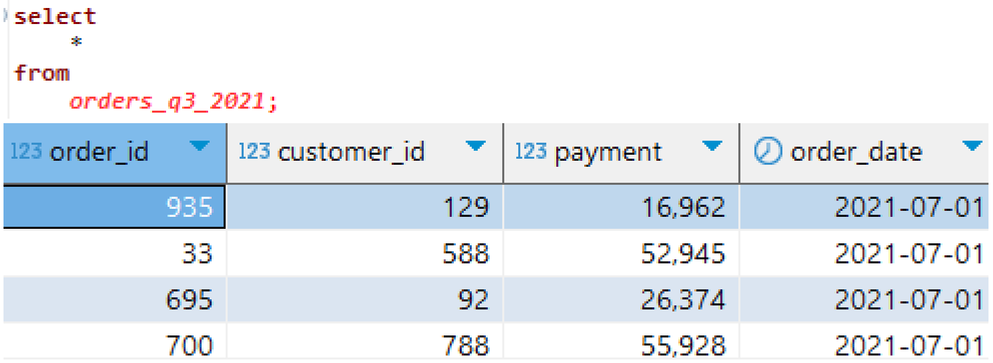
```
select
    *
from
    orders_q1_2021;
```

order_id	customer_id	payment	order_date
683	276	59,748	2021-01-01
18	299	22,902	2021-01-01
572	479	34,112	2021-01-01
509	676	20,368	2021-01-02



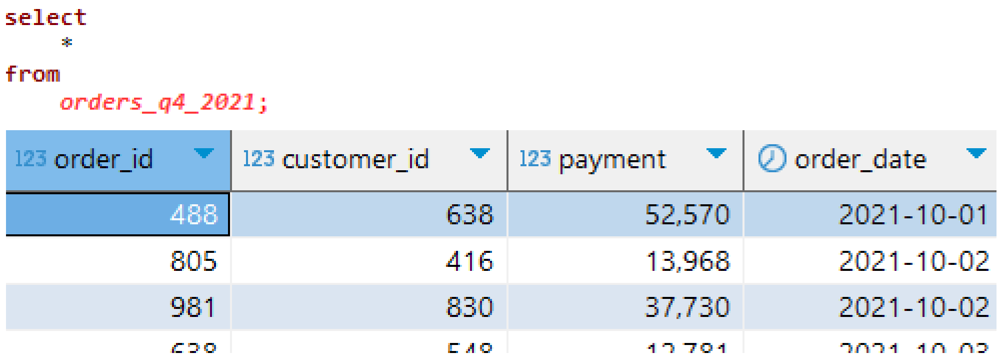
```
select
    *
from
    orders_q2_2021;
```

order_id	customer_id	payment	order_date
127	51	43,262	2021-04-01
356	236	34,259	2021-04-01
939	432	24,169	2021-04-02
636	793	23,528	2021-04-02



```
select
    *
from
    orders_q3_2021;
```

order_id	customer_id	payment	order_date
935	129	16,962	2021-07-01
33	588	52,945	2021-07-01
695	92	26,374	2021-07-01
700	788	55,928	2021-07-01



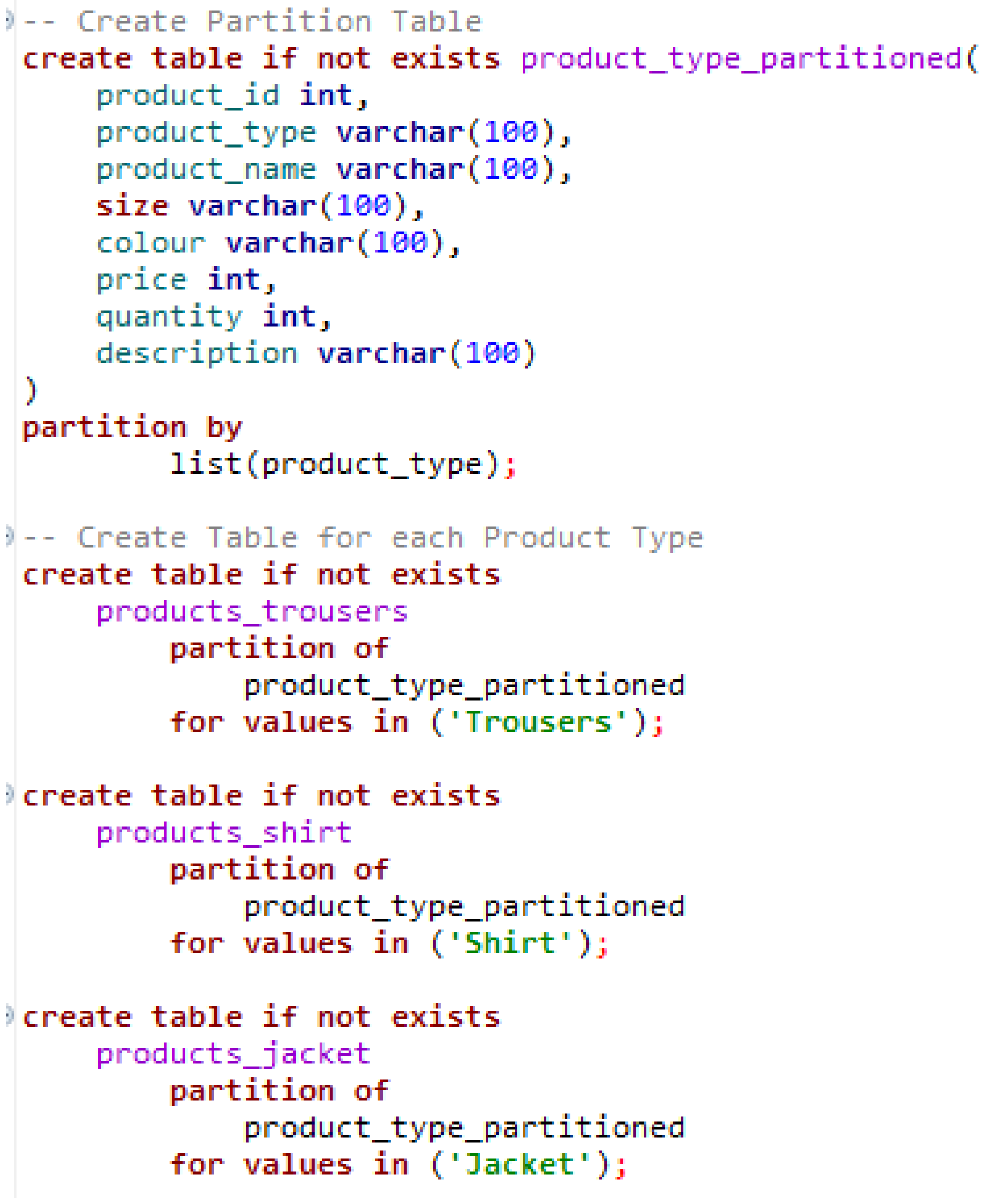
```
select
    *
from
    orders_q4_2021;
```

123 order_id	123 customer_id	123 payment	order_date
488	638	52,570	2021-10-01
805	416	13,968	2021-10-02
981	830	37,730	2021-10-02

5. Searches often use filters based on product type. Create a partition to enhanced the search process
    - Create a partition table and insert data from the products table.
    - Do query from partition table

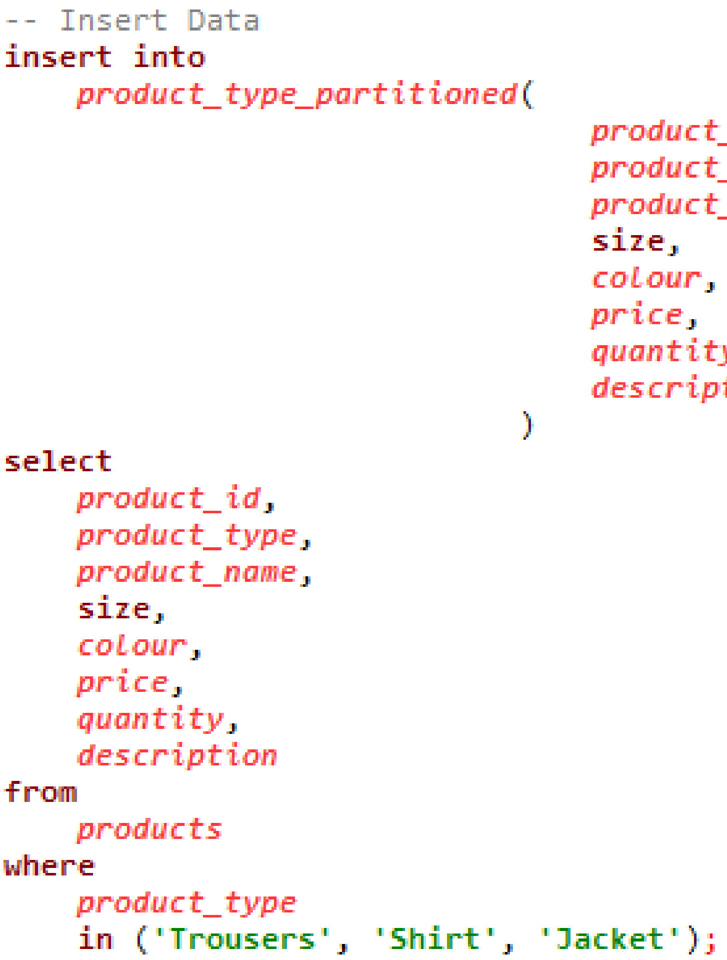
Answer Here

- **Syntax: Create Partition**



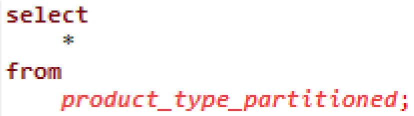
```
-- Create Partition Table
create table if not exists product_type_partitioned(
    product_id int,
    product_type varchar(100),
    product_name varchar(100),
    size varchar(100),
    colour varchar(100),
    price int,
    quantity int,
    description varchar(100)
)
partition by
        list(product_type);

-- Create Table for each Product Type
create table if not exists
    products_trousers
        partition of
            product_type_partitioned
        for values in ('Trousers');

create table if not exists
    products_shirt
        partition of
            product_type_partitioned
        for values in ('Shirt');

create table if not exists
    products_jacket
        partition of
            product_type_partitioned
        for values in ('Jacket');
```

- **Syntax: Insert Data**

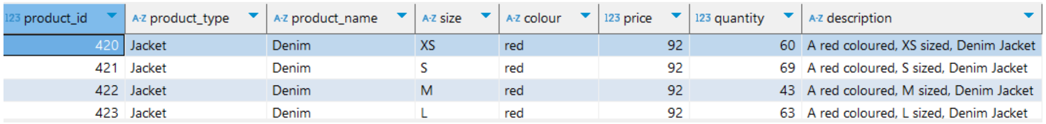


```
-- Insert Data
insert into
    product_type_partitioned(
                            product_id,
                            product_type,
                            product_name,
                            size,
                            colour,
                            price,
                            quantity,
                            description
                        )
select
    product_id,
    product_type,
    product_name,
    size,
    colour,
    price,
    quantity,
    description
from
    products
where
    product_type
    in ('Trousers', 'Shirt', 'Jacket');
```

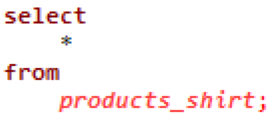
- **Syntax: Select from partition table**



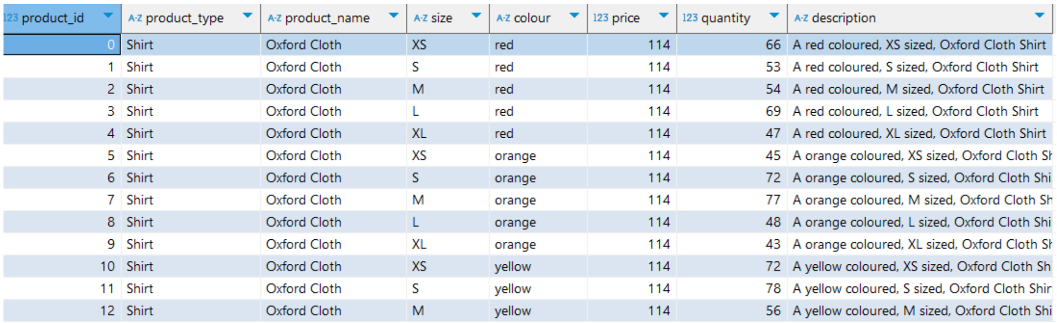
```
select
    *
from
    product_type_partitioned;
```



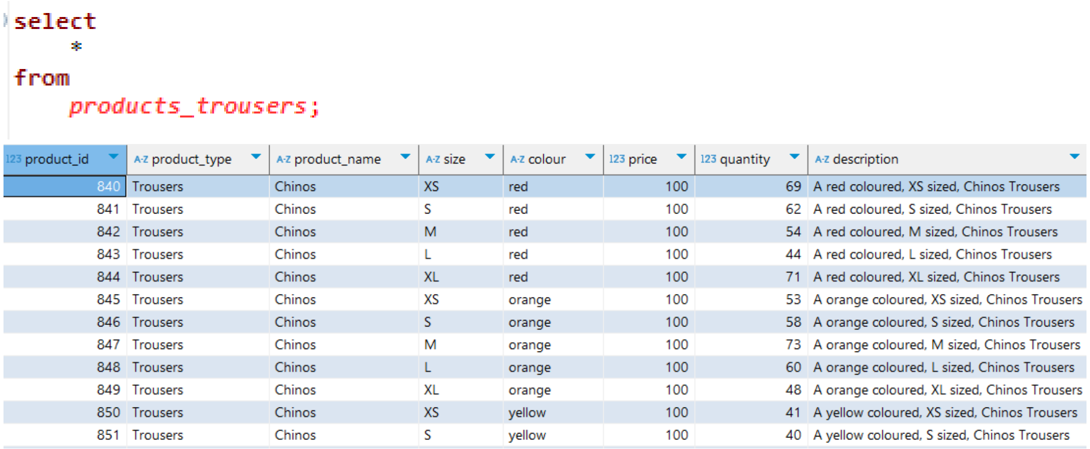
product_id	product_type	product_name	size	colour	price	quantity	description
420	Jacket	Denim	XS	red	92	60	A red coloured, XS sized, Denim Jacket
421	Jacket	Denim	S	red	92	69	A red coloured, S sized, Denim Jacket
422	Jacket	Denim	M	red	92	43	A red coloured, M sized, Denim Jacket
423	Jacket	Denim	L	red	92	63	A red coloured, L sized, Denim Jacket



```
select
    *
from
    products_shirt;
```

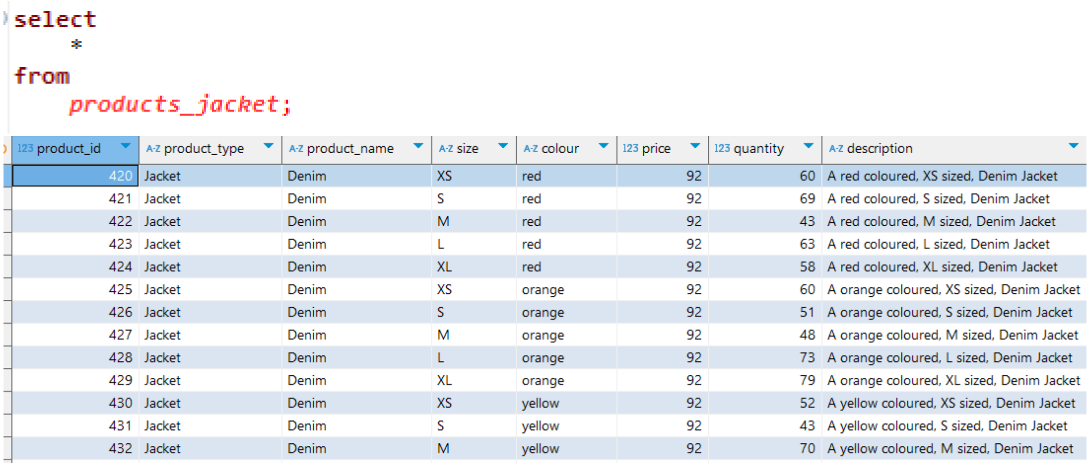


product_id	product_type	product_name	size	colour	price	quantity	description
0	Shirt	Oxford Cloth	XS	red	114	66	A red coloured, XS sized, Oxford Cloth Shirt
1	Shirt	Oxford Cloth	S	red	114	53	A red coloured, S sized, Oxford Cloth Shirt
2	Shirt	Oxford Cloth	M	red	114	54	A red coloured, M sized, Oxford Cloth Shirt
3	Shirt	Oxford Cloth	L	red	114	69	A red coloured, L sized, Oxford Cloth Shirt
4	Shirt	Oxford Cloth	XL	red	114	47	A red coloured, XL sized, Oxford Cloth Shirt
5	Shirt	Oxford Cloth	XS	orange	114	45	A orange coloured, XS sized, Oxford Cloth Sh
6	Shirt	Oxford Cloth	S	orange	114	72	A orange coloured, S sized, Oxford Cloth Shi
7	Shirt	Oxford Cloth	M	orange	114	77	A orange coloured, M sized, Oxford Cloth Sh
8	Shirt	Oxford Cloth	L	orange	114	48	A orange coloured, L sized, Oxford Cloth Shi
9	Shirt	Oxford Cloth	XL	orange	114	43	A orange coloured, XL sized, Oxford Cloth Sh
10	Shirt	Oxford Cloth	XS	yellow	114	72	A yellow coloured, XS sized, Oxford Cloth Sh
11	Shirt	Oxford Cloth	S	yellow	114	78	A yellow coloured, S sized, Oxford Cloth Shir
12	Shirt	Oxford Cloth	M	yellow	114	56	A yellow coloured, M sized, Oxford Cloth Shi



```
select
        *
from
        products_trousers;
```

product_id	product_type	product_name	size	colour	price	quantity	description
840	Trousers	Chinos	XS	red	100	69	A red coloured, XS sized, Chinos Trousers
841	Trousers	Chinos	S	red	100	62	A red coloured, S sized, Chinos Trousers
842	Trousers	Chinos	M	red	100	54	A red coloured, M sized, Chinos Trousers
843	Trousers	Chinos	L	red	100	44	A red coloured, L sized, Chinos Trousers
844	Trousers	Chinos	XL	red	100	71	A red coloured, XL sized, Chinos Trousers
845	Trousers	Chinos	XS	orange	100	53	A orange coloured, XS sized, Chinos Trousers
846	Trousers	Chinos	S	orange	100	58	A orange coloured, S sized, Chinos Trousers
847	Trousers	Chinos	M	orange	100	73	A orange coloured, M sized, Chinos Trousers
848	Trousers	Chinos	L	orange	100	60	A orange coloured, L sized, Chinos Trousers
849	Trousers	Chinos	XL	orange	100	48	A orange coloured, XL sized, Chinos Trousers
850	Trousers	Chinos	XS	yellow	100	41	A yellow coloured, XS sized, Chinos Trousers
851	Trousers	Chinos	S	yellow	100	40	A yellow coloured, S sized, Chinos Trousers



```
select
        *
from
        products_jacket;
```

product_id	product_type	product_name	size	colour	price	quantity	description
420	Jacket	Denim	XS	red	92	60	A red coloured, XS sized, Denim Jacket
421	Jacket	Denim	S	red	92	69	A red coloured, S sized, Denim Jacket
422	Jacket	Denim	M	red	92	43	A red coloured, M sized, Denim Jacket
423	Jacket	Denim	L	red	92	63	A red coloured, L sized, Denim Jacket
424	Jacket	Denim	XL	red	92	58	A red coloured, XL sized, Denim Jacket
425	Jacket	Denim	XS	orange	92	60	A orange coloured, XS sized, Denim Jacket
426	Jacket	Denim	S	orange	92	51	A orange coloured, S sized, Denim Jacket
427	Jacket	Denim	M	orange	92	48	A orange coloured, M sized, Denim Jacket
428	Jacket	Denim	L	orange	92	73	A orange coloured, L sized, Denim Jacket
429	Jacket	Denim	XL	orange	92	79	A orange coloured, XL sized, Denim Jacket
430	Jacket	Denim	XS	yellow	92	52	A yellow coloured, XS sized, Denim Jacket
431	Jacket	Denim	S	yellow	92	43	A yellow coloured, S sized, Denim Jacket
432	Jacket	Denim	M	yellow	92	70	A yellow coloured, M sized, Denim Jacket

6. Given the Students table as shown below:

student_id	first_name	last_name	project_id	project
1000	Kira	Granger	P-0011	E-commerce Website
1001	Katherine	Erlich	P-0012	IoT Program
1000	Kira	Granger	P-0013	Book Catalog Website
1003	Shannon	Black	P-0014	VR Game

Answer Here

- **Explain the condition of the table (whether it satisfies 1NF, 2NF, and 3NF)**
  1NF : This table satisfies 1NF because there are no multivalues (columns contain only single values)

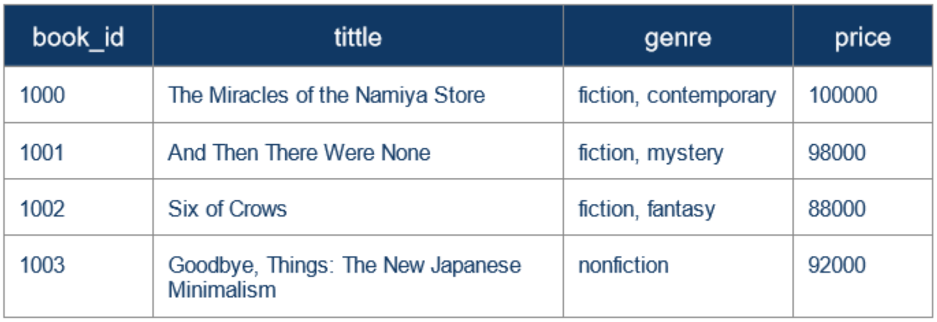
  2NF:

  - FD1 : student_id -> first_name, last_name
  - FD2 : project_id -> project

  - This table does not satisfy 2NF because there is still a partial dependency because first_name and last_name only depend on student_id and also project only depend on project_id (not the combination of student_id and project_id). So, there must be 2 tables, namely the student table and the project table

- **Transform it into a form that meets the requirements of 3NF.**
  3NF:
  - This table does not fulfill 3NF because the table does not satisfy the 2NF Condition
  - To fulfill 3NF, the table must fulfill 2NF, and there must be no transitive dependencies ***(non-primary attributes must not depend on other non-primary attributes)***.

7. Given the Books table as shown below:

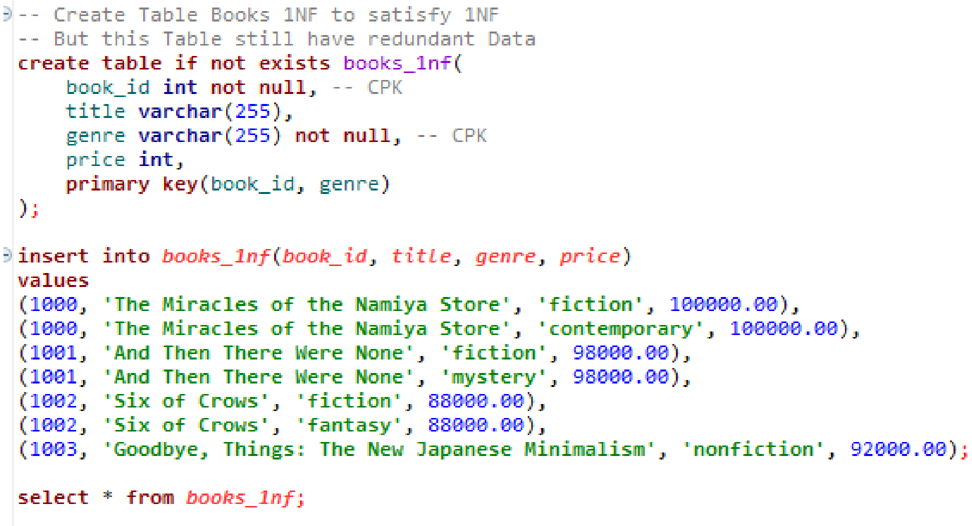


book_id	tittle	genre	price
1000	The Miracles of the Namiya Store	fiction, contemporary	100000
1001	And Then There Were None	fiction, mystery	98000
1002	Six of Crows	fiction, fantasy	88000
1003	Goodbye, Things: The New Japanese Minimalism	nonfiction	92000

- **Explain the condition of the table (whether it satisfies 1NF, 2NF, and 3NF)**
  1NF (First Normal Form):
  - A table satisfies 1NF if each column contains atomic (indivisible) values, meaning no repeating groups or multiple values in a single cell.

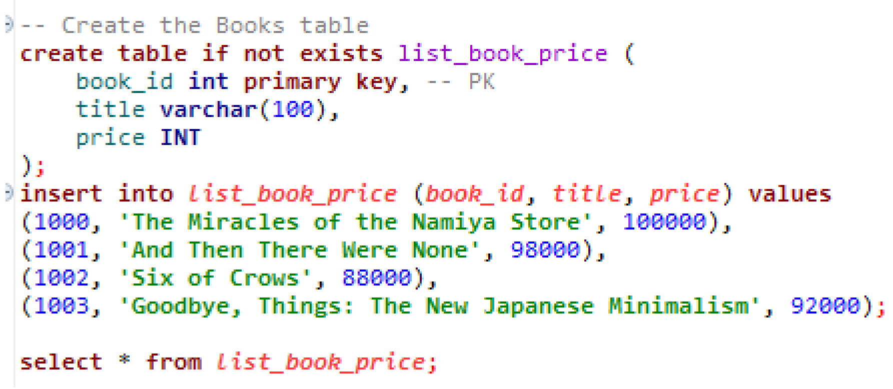
  In this table, the genre column contains multiple values for some rows (e.g., "fiction, contemporary" or "fiction, mystery"). This violates 1NF because these are not atomic values.

  Conclusion: The table does not satisfy 1NF due to the non-atomic genre column. So new table must created to satisfy 1NF

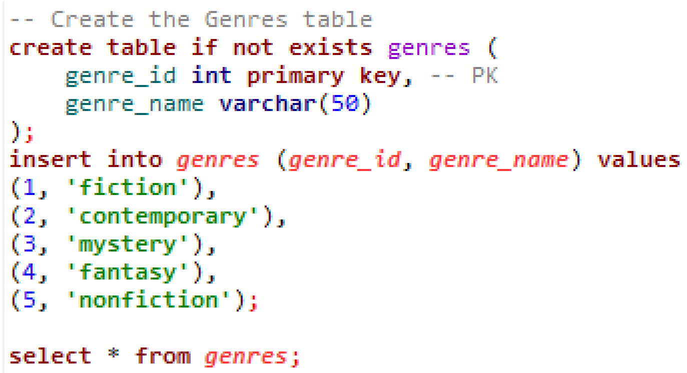


```
-- Create Table Books 1NF to satisfy 1NF
-- But this Table still have redundant Data
create table if not exists books_1nf(
    book_id int not null, -- CPK
    title varchar(255),
    genre varchar(255) not null, -- CPK
    price int,
    primary key(book_id, genre)
);

insert into books_1nf(book_id, title, genre, price)
values
(1000, 'The Miracles of the Namiya Store', 'fiction', 100000.00),
(1000, 'The Miracles of the Namiya Store', 'contemporary', 100000.00),
(1001, 'And Then There Were None', 'fiction', 98000.00),
(1001, 'And Then There Were None', 'mystery', 98000.00),
(1002, 'Six of Crows', 'fiction', 88000.00),
(1002, 'Six of Crows', 'fantasy', 88000.00),
(1003, 'Goodbye, Things: The New Japanese Minimalism', 'nonfiction', 92000.00);

select * from books_1nf;
```

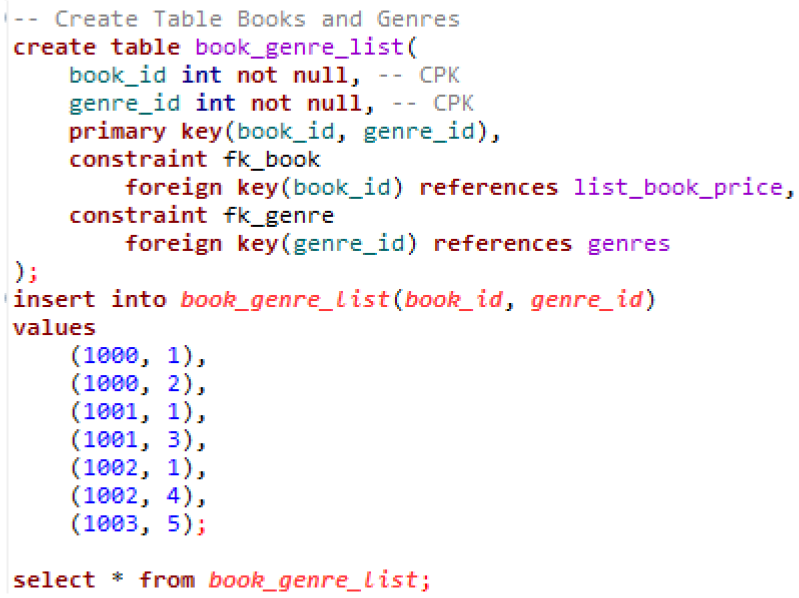
  But, The Table books_1nf has satisfied 1NF but still have redundant data. So new table must created again



```sql
-- Create the Books table
create table if not exists list_book_price (
    book_id int primary key, -- PK
    title varchar(100),
    price INT
);
insert into list_book_price (book_id, title, price) values
(1000, 'The Miracles of the Namiya Store', 100000),
(1001, 'And Then There Were None', 98000),
(1002, 'Six of Crows', 88000),
(1003, 'Goodbye, Things: The New Japanese Minimalism', 92000);

select * from list_book_price;
```



```sql
-- Create the Genres table
create table if not exists genres (
    genre_id int primary key, -- PK
    genre_name varchar(50)
);
insert into genres (genre_id, genre_name) values
(1, 'fiction'),
(2, 'contemporary'),
(3, 'mystery'),
(4, 'fantasy'),
(5, 'nonfiction');

select * from genres;
```



```sql
-- Create Table Books and Genres
create table book_genre_list(
    book_id int not null, -- CPK
    genre_id int not null, -- CPK
    primary key(book_id, genre_id),
    constraint fk_book
        foreign key(book_id) references list_book_price,
    constraint fk_genre
        foreign key(genre_id) references genres
);
insert into book_genre_list(book_id, genre_id)
values
    (1000, 1),
    (1000, 2),
    (1001, 1),
    (1001, 3),
    (1002, 1),
    (1002, 4),
    (1003, 5);

select * from book_genre_list;
```

2NF (Second Normal Form):

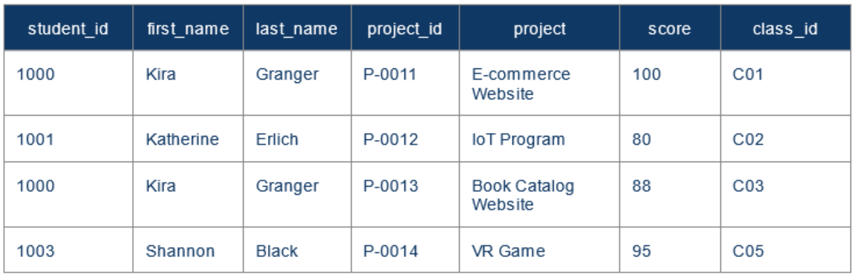
- A table satisfies 2NF because it is satisfies 1NF after create new table and all non-primary key attributes are fully functionally dependent on the primary key.
- Assuming book_id is the primary key, each attribute (title, genre, price) is fully dependent on book_id, so there’s no partial dependency here.

- **Transform it into a form that meets the requirements of 3NF.**

  3NF (Third Normal Form):

  - A table satisfies 3NF if it is in 2NF and has no transitive dependencies (no non-primary key column should depend on another non-primary key column).
  - In this table, there are no transitive dependencies. So, it is satisfy 3NF

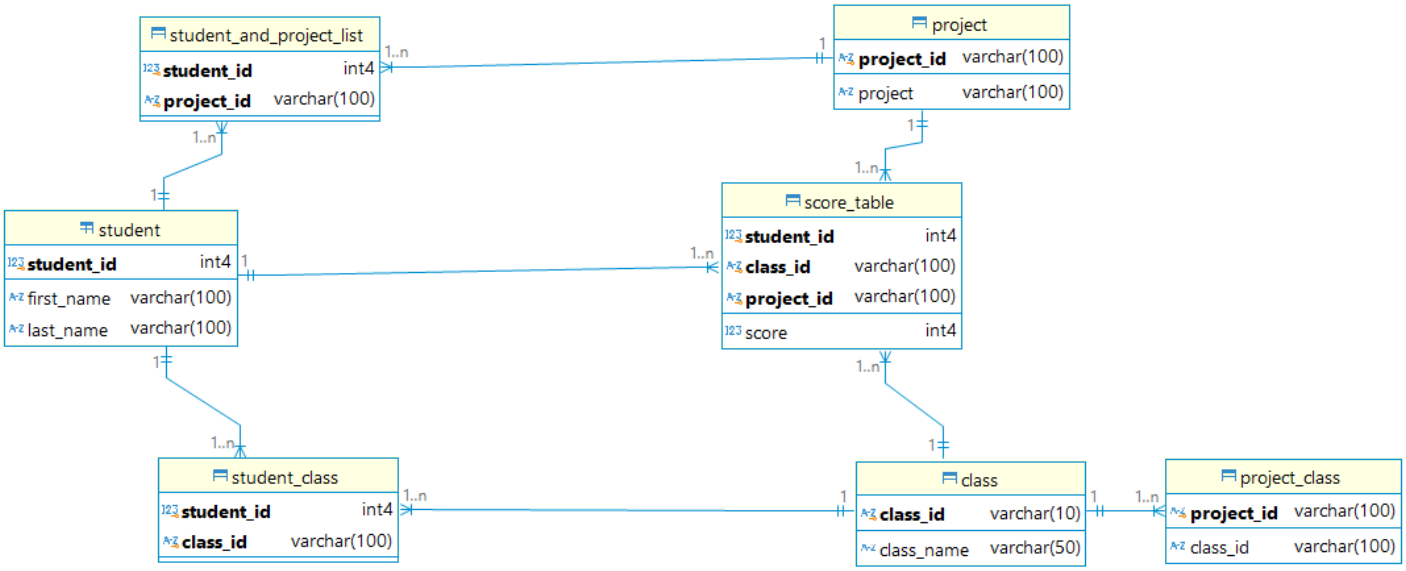
8. Table structure in number 6 should be modified so that it may record information about the **class** that provided the **project assignment as well as the score of the project the student worked on.** Implement the table structure that was created using the ddl syntax.



student_id	first_name	last_name	project_id	project	score	class_id
1000	Kira	Granger	P-0011	E-commerce Website	100	C01
1001	Katherine	Erlich	P-0012	IoT Program	80	C02
1000	Kira	Granger	P-0013	Book Catalog Website	88	C03
1003	Shannon	Black	P-0014	VR Game	95	C05

Answer Here

- **Table Structure**



student				
student_id	:	int		Primary Key (PK)
first_name	:	varchar(100)	Not Null	
last_name	:	varchar(100)	Not Null	

project				
project_id	:	varchar(100)		Primary Key (PK)
project	:	varchar(100)	Not Null	

student_and_project_list		
student_id : int	Not Null	Composite Primary Key (CPK), Foreign Key (FK) Referencing student(student_id)
project_id : varchar(100)	Not Null	Composite Primary Key (CPK), Foreign Key (FK) Referencing project(project_id)

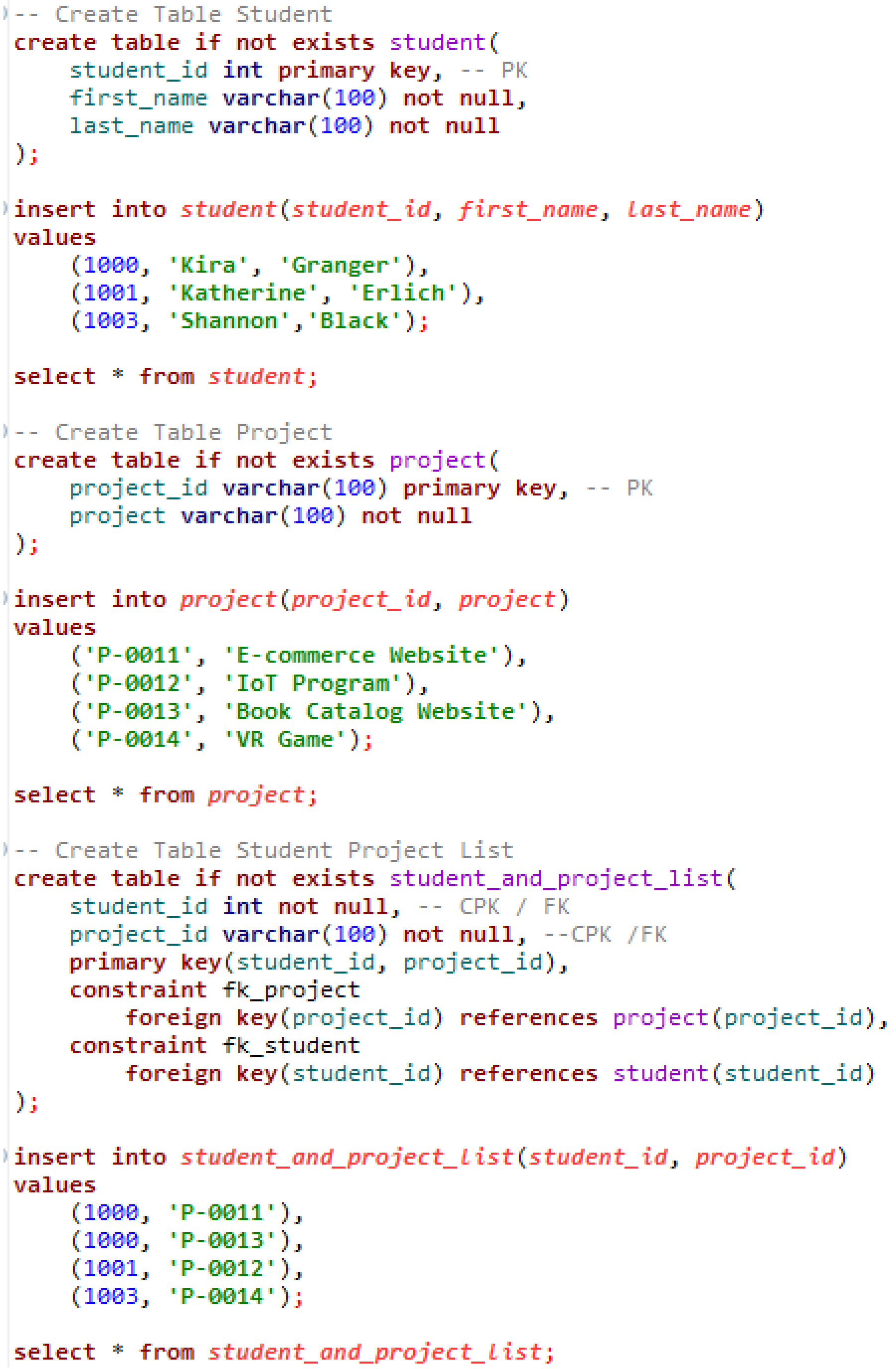
class		
class_id : varchar(10)		Primary Key (PK)
class_name : varchar(50)	Not Null	

student_class		
student_id : int	Not Null	Composite Primary Key (CPK), Foreign Key (FK) referencing student(student_id)
class_id : varchar(100)	Not Null	Composite Primary Key (CPK), Foreign Key (FK) referencing class(class_id)

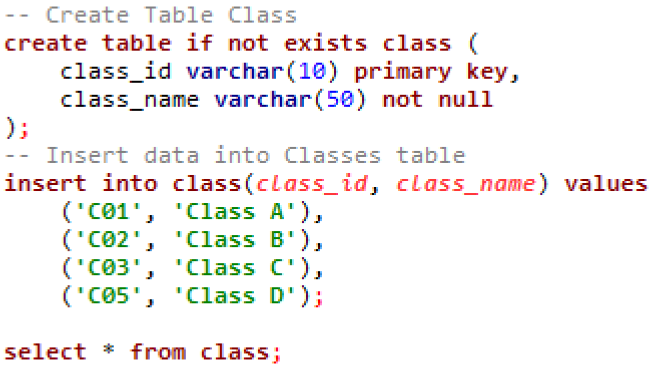
project_class		
project_id : varchar(100)		Primary Key (PK)
class_id : varchar(100)	Not Null	Foreign Key (FK) referencing class(class_id)

score_table		
student_id : int	Not Null	Composite Primary Key (CPK), Foreign Key (FK) referencing student(student_id)
class_id : varchar(100)	Not Null	Primary Key (CPK), Foreign Key (FK) referencing class(class_id)
project_id : varchar(100)	Not Null	Composite Primary Key (CPK), Foreign Key (FK) referencing project(project_id)
score : int	Not Null, Default 0	

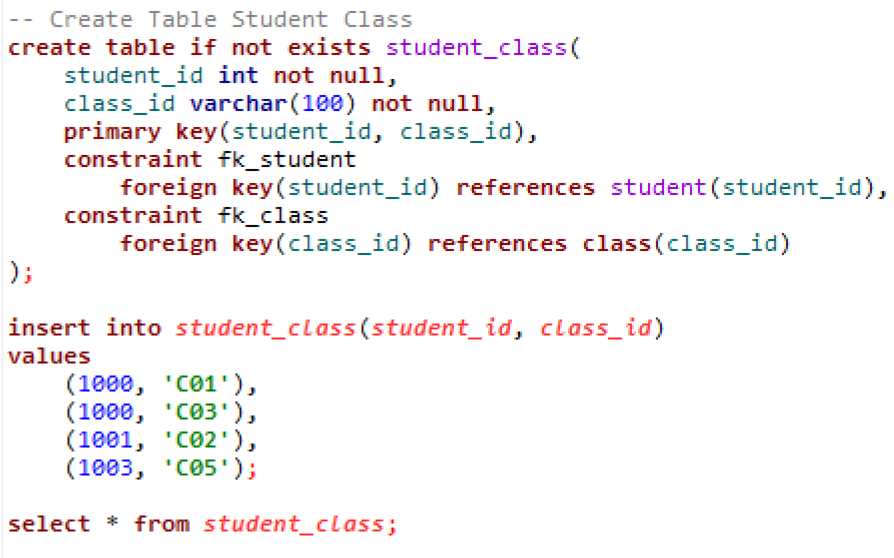
- **DDL**



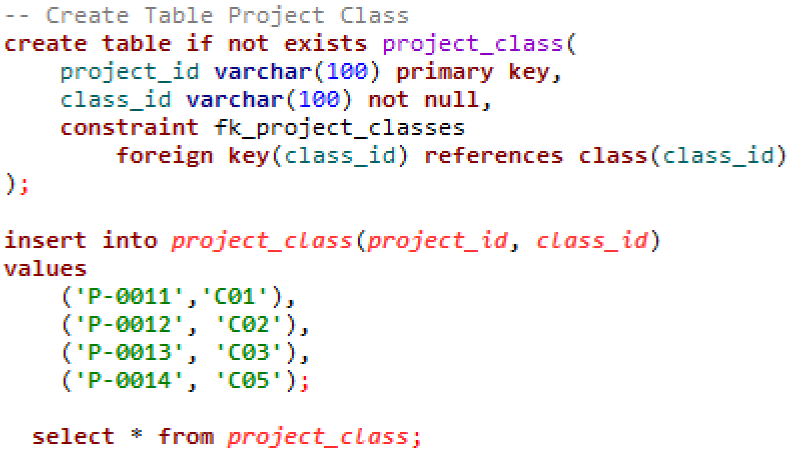
```
-- Create Table Student
create table if not exists student(
    student_id int primary key, -- PK
    first_name varchar(100) not null,
    last_name varchar(100) not null
);

insert into student(student_id, first_name, last_name)
values
    (1000, 'Kira', 'Granger'),
    (1001, 'Katherine', 'Erlich'),
    (1003, 'Shannon','Black');

select * from student;

-- Create Table Project
create table if not exists project(
    project_id varchar(100) primary key, -- PK
    project varchar(100) not null
);

insert into project(project_id, project)
values
    ('P-0011', 'E-commerce Website'),
    ('P-0012', 'IoT Program'),
    ('P-0013', 'Book Catalog Website'),
    ('P-0014', 'VR Game');

select * from project;

-- Create Table Student Project List
create table if not exists student_and_project_list(
    student_id int not null, -- CPK / FK
    project_id varchar(100) not null, --CPK /FK
    primary key(student_id, project_id),
    constraint fk_project
        foreign key(project_id) references project(project_id),
    constraint fk_student
        foreign key(student_id) references student(student_id)
);

insert into student_and_project_list(student_id, project_id)
values
    (1000, 'P-0011'),
    (1000, 'P-0013'),
    (1001, 'P-0012'),
    (1003, 'P-0014');

select * from student_and_project_list;
```



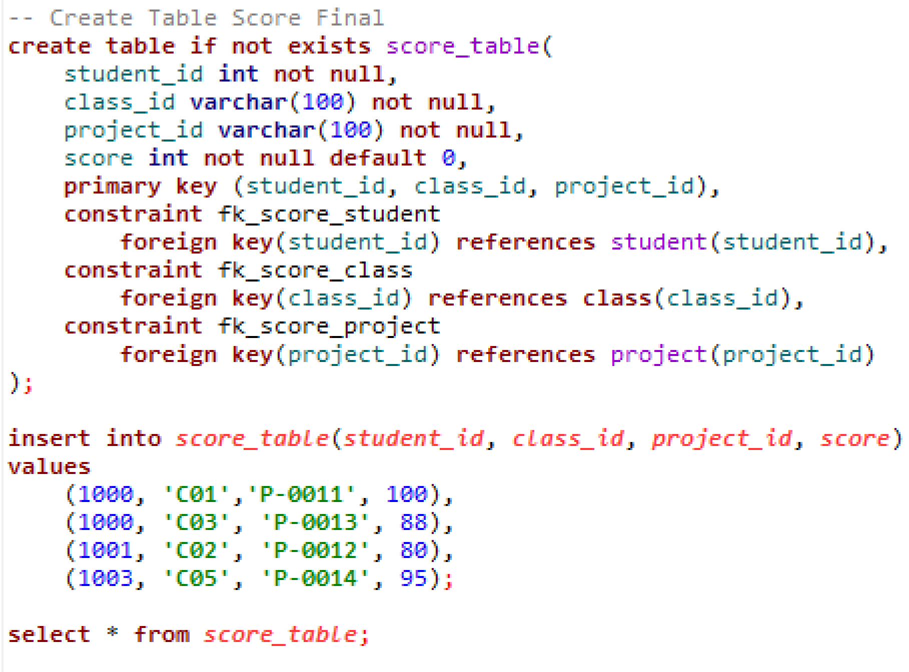
```
-- Create Table Class
create table if not exists class (
    class_id varchar(10) primary key,
    class_name varchar(50) not null
);
-- Insert data into Classes table
insert into class(class_id, class_name) values
    ('C01', 'Class A'),
    ('C02', 'Class B'),
    ('C03', 'Class C'),
    ('C05', 'Class D');

select * from class;
```



```sql
-- Create Table Student Class
create table if not exists student_class(
    student_id int not null,
    class_id varchar(100) not null,
    primary key(student_id, class_id),
    constraint fk_student
        foreign key(student_id) references student(student_id),
    constraint fk_class
        foreign key(class_id) references class(class_id)
);

insert into student_class(student_id, class_id)
values
    (1000, 'C01'),
    (1000, 'C03'),
    (1001, 'C02'),
    (1003, 'C05');

select * from student_class;
```



```sql
-- Create Table Project Class
create table if not exists project_class(
    project_id varchar(100) primary key,
    class_id varchar(100) not null,
    constraint fk_project_classes
        foreign key(class_id) references class(class_id)
);

insert into project_class(project_id, class_id)
values
    ('P-0011','C01'),
    ('P-0012', 'C02'),
    ('P-0013', 'C03'),
    ('P-0014', 'C05');

  select * from project_class;
```



```sql
-- Create Table Score Final
create table if not exists score_table(
    student_id int not null,
    class_id varchar(100) not null,
    project_id varchar(100) not null,
    score int not null default 0,
    primary key (student_id, class_id, project_id),
    constraint fk_score_student
        foreign key(student_id) references student(student_id),
    constraint fk_score_class
        foreign key(class_id) references class(class_id),
    constraint fk_score_project
        foreign key(project_id) references project(project_id)
);

insert into score_table(student_id, class_id, project_id, score)
values
    (1000, 'C01','P-0011', 100),
    (1000, 'C03', 'P-0013', 88),
    (1001, 'C02', 'P-0012', 80),
    (1003, 'C05', 'P-0014', 95);

select * from score_table;
```