TERM PROJECT – FINAL DELIVERABLE
SELLING ON AMAZON

Prepared by: Alena Raupova

1. STRUCTURAL BUSINESS RULES

- Each seller has one or more inventories.
- Each inventory belongs to only one seller.
- Each seller has first and last name.
- Each inventory has information about the inventory id, what the product is, how many units there are, the condition, and which seller it belongs to.

- Each inventory stores only one product.
- Each product can be stored in one or more inventories.
- Each product has information about the product id, the product's name, description, price, to which category it belongs.

- Each product belongs to only one category.
- Each category has one or more products.
- Each category has the category id, the category name.

- Each consumer selection contains products from only one inventory.
- Each inventory can be selected in zero or more consumer selections.
- Each consumer selection has information about the quantity of selected products, from which inventory the products were selected and to which order it belongs.

- Each consumer selection is displayed in only one order.
- Each order contains one or more consumer selections.
- Each order has information about the order id, the shipment speed, the price of the order, the consumer of that order, which package is related to the order, payment for the order, to which address send the order.

- Each consumer places zero or more orders.
- Each order is placed by only one consumer.
- Each consumer has consumer id, first and last name, phone number, email.
- Each consumer has one or more addresses.
- Each address belongs to only one consumer.
- Each address has address id, address line 1, address line 2, city, state, zip code, address type, information about the consumer to which the address belongs.

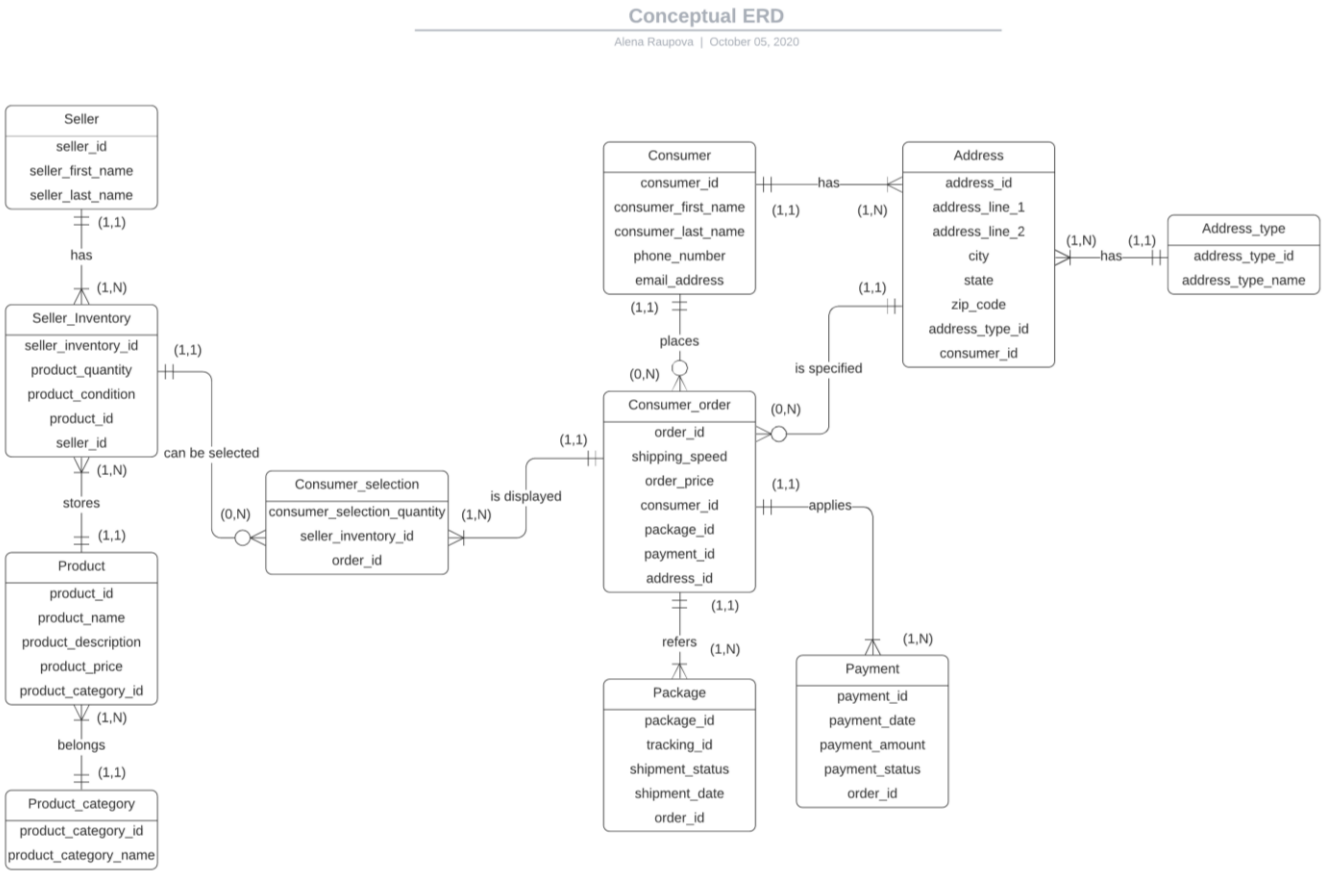
- Each address has only one address type.
- Each address type has one or more addresses.
- Each address type has address type id, address type name.

- Each address is specified in zero or more orders.
- Each order refers to only one address, which type is the default.

- Each order can be sent in one or more packages.
- Each package refers to only one order.
- Each package has package id, tracking id, shipment status, shipment date, which order it belongs to.

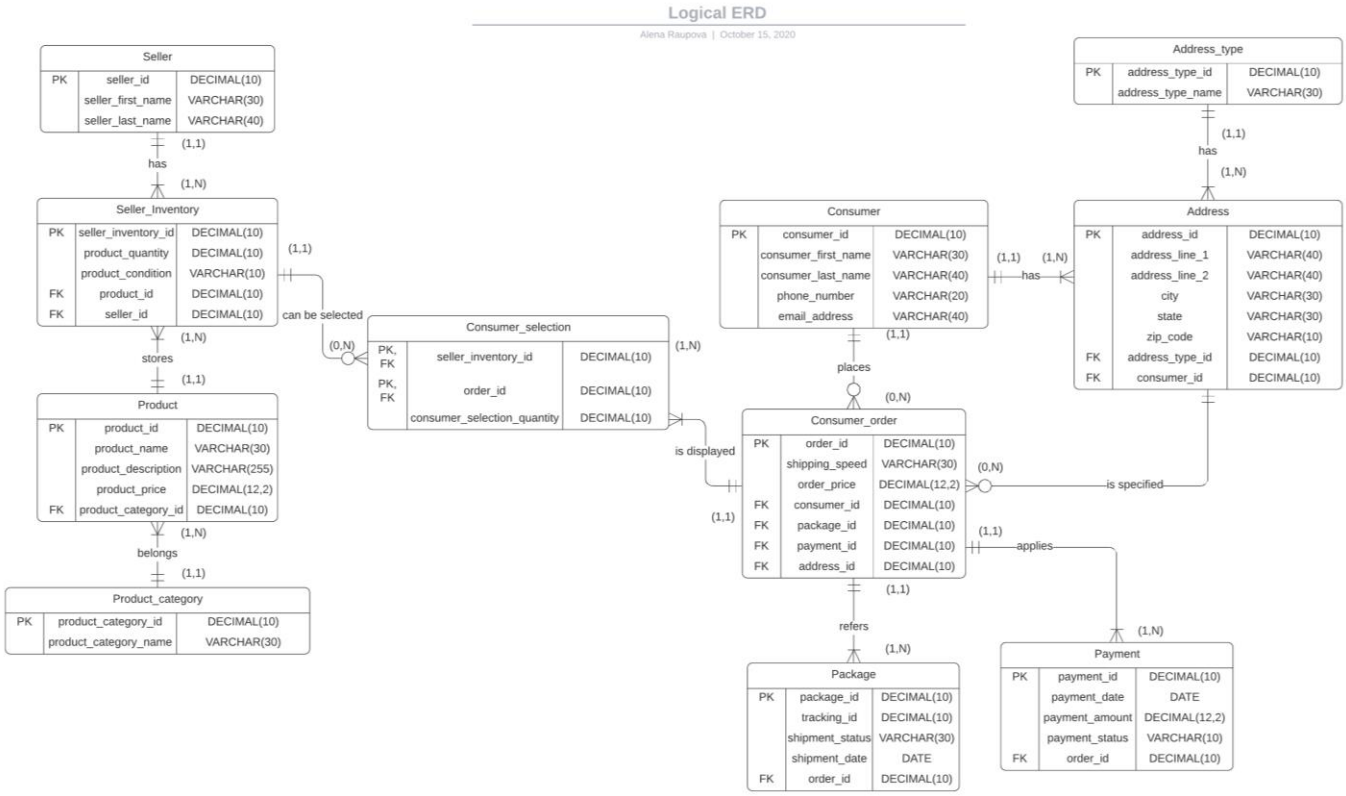
- Each order is paid one or more payments.
- Each payment applies to only one order.
- Each payment has payment id, payment date, payment amount, payment status, which order it belongs to.

2. CONCEPTUAL ERD



In this deliverable, I changed the "Package" entity to add a "shipment_date" field.

3. LOGICAL ERD



4. ASPECT 1

4.1. Creation of reusable stored procedure

Clark_AlenaRaupova

Worksheet | Query Builder

```
CREATE OR REPLACE PROCEDURE ADD_NEW_PRODUCT (
product_id_arg IN DECIMAL,
product_name_arg IN VARCHAR,
product_description_arg IN VARCHAR,
product_price_arg IN DECIMAL,
product_category_id_arg IN DECIMAL,
seller_inventory_id_arg IN DECIMAL,
seller_id_arg IN DECIMAL)
IS
BEGIN
INSERT INTO Product (product_id, product_name, product_description, product_price, product_category_id)
VALUES (product_id_arg, product_name_arg, product_description_arg, product_price_arg, product_category_id_arg);
INSERT INTO Seller_inventory (seller_inventory_id, product_quantity, product_condition, product_id, seller_id)
VALUES (seller_inventory_id_arg, 0, NULL, product_id_arg, seller_id_arg);
END;
```

Script Output

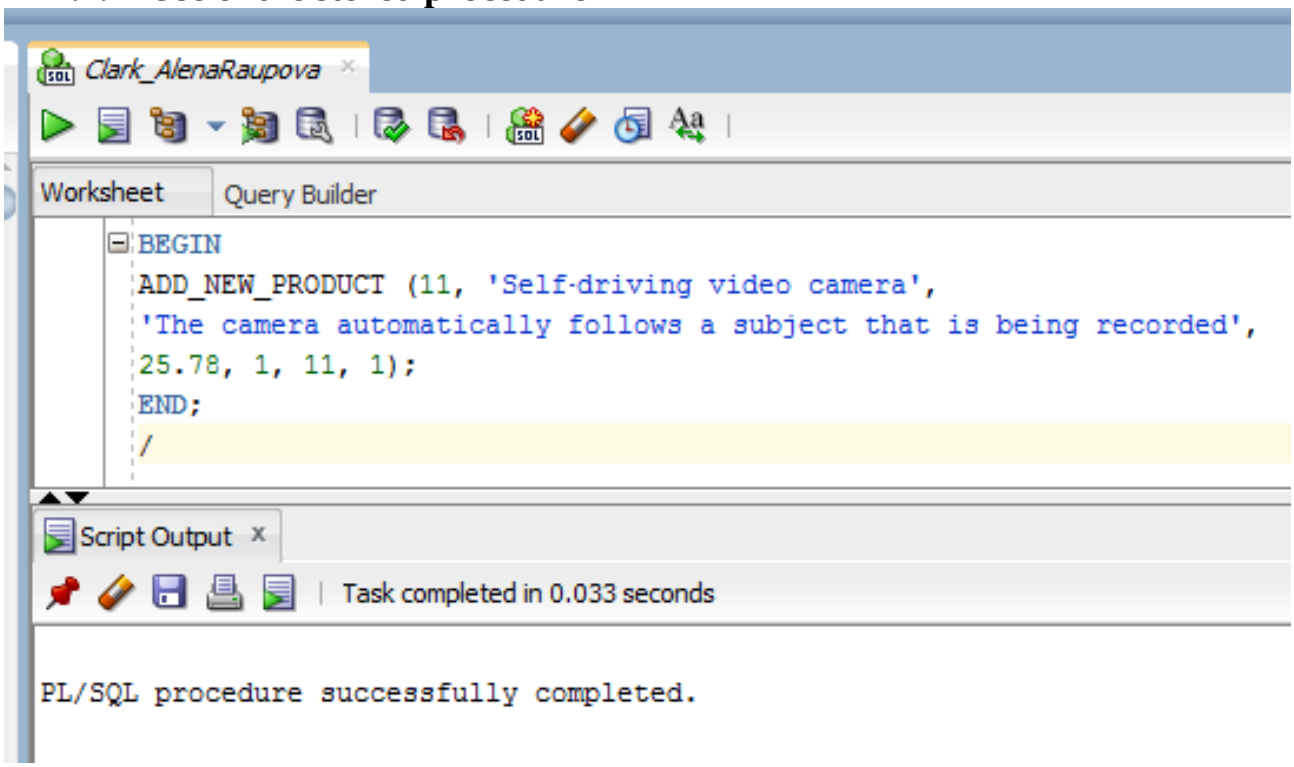
Task completed in 2.109 seconds

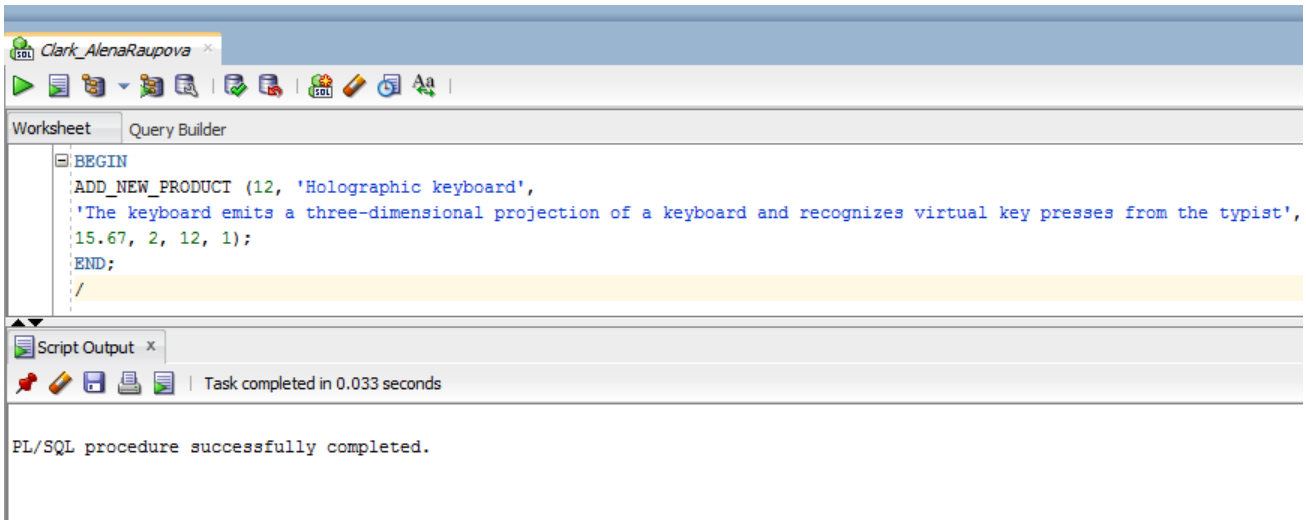
```
Procedure ADD_NEW_PRODUCT compiled
```

The procedure "ADD_NEW_PRODUCT" creates a new record in the "Product" table. Simultaneously with the creation of the record in the "Product" table, a

new record will be created in the "Seller_inventory" table. This procedure is necessary in order for the seller to add the product to the Amazon site. However, this assumes that the seller has not yet sent any units of product to the Amazon warehouse. Therefore, a new record in the inventory table will have null values in the product_quantity and product_condition fields.

4.2. Use of the stored procedure





4.3. SQL query

Clark_AlenaRaupova

Worksheet | Query Builder

```
SELECT Product.product_name, Product.product_price, Product_category.product_category_name
FROM Product
JOIN Product_category ON Product.product_category_id = Product_category.product_category_id
WHERE Product_category.product_category_name = 'Computers' OR Product_category.product_category_name = 'Electronics'
AND product_price <= 30;
```

Query Result | SQL | All Rows Fetched: 4 in 0.012 seconds

	PRODUCT_NAME	PRODUCT_PRICE	PRODUCT_CATEGORY_NAME
1	Portable Charger	19.99	Computers
2	Webcam	30	Electronics
3	Self-driving video camera	25.78	Computers
4	Holographic keyboard	15.67	Electronics

5. Aspect 2

5.1. Creation of reusable stored procedure

Clark_AlenaRaupova

Worksheet | Query Builder

```
CREATE OR REPLACE PROCEDURE SELLER_DELIVERS_PRODUCT (
seller_inventory_id_arg IN DECIMAL,
product_quantity_arg IN DECIMAL,
product_condition_arg IN VARCHAR,
product_id_arg IN DECIMAL,
seller_id_arg IN DECIMAL)
IS
BEGIN
UPDATE Seller_inventory
SET product_quantity = product_quantity + product_quantity_arg,
product_condition = product_condition_arg
WHERE seller_inventory_id = seller_inventory_id_arg
AND product_id = product_id_arg
AND seller_id = seller_id;
END;
```

Query Result | Script Output

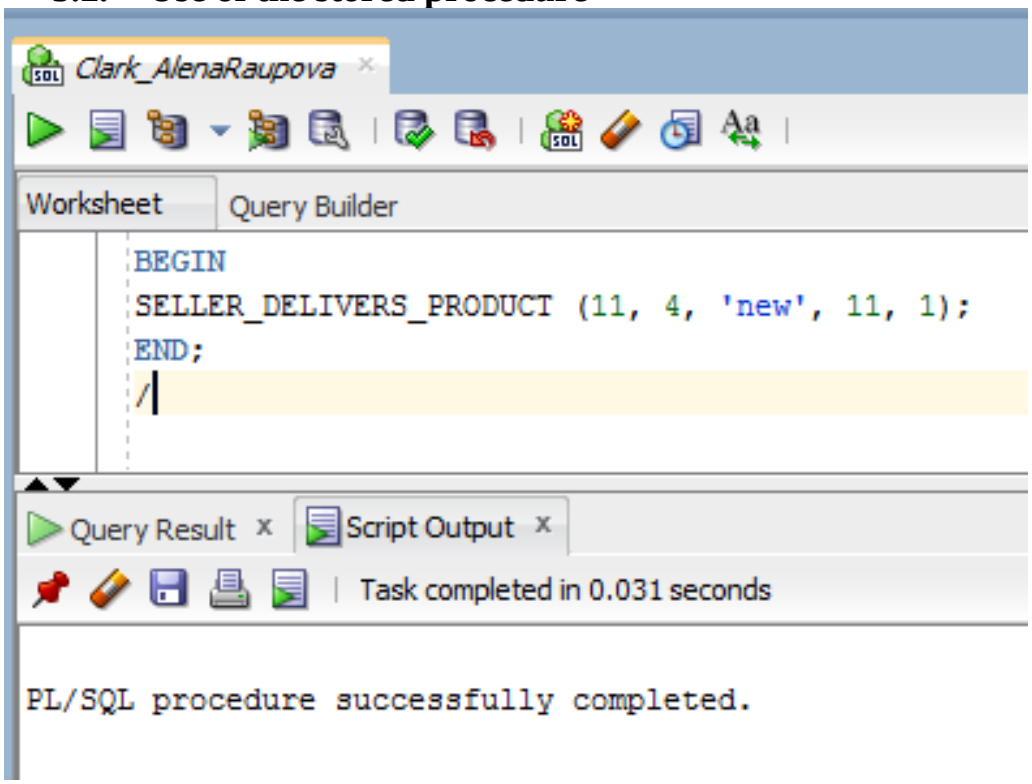
Task completed in 0.627 seconds

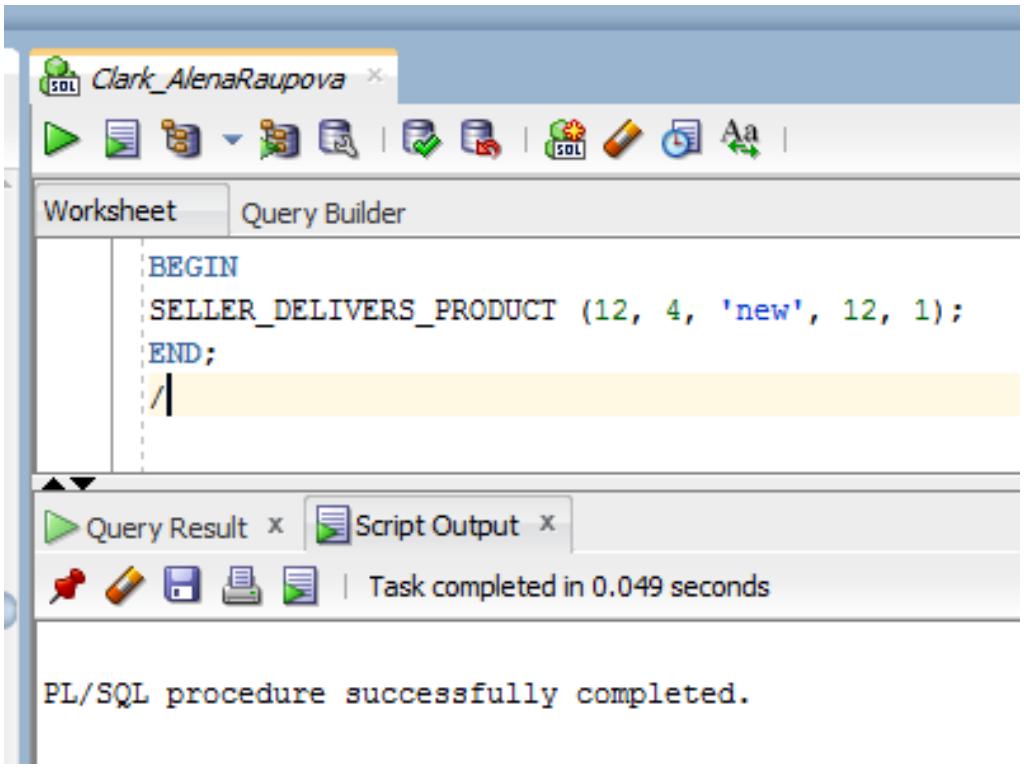
```
Procedure SELLER_DELIVERS_PRODUCT compiled
```

After the seller sends the product and information about it to the Amazon warehouse, we need to update the record in the "Inventory" table, which refers to the received product and the seller. In order to get a new value for the

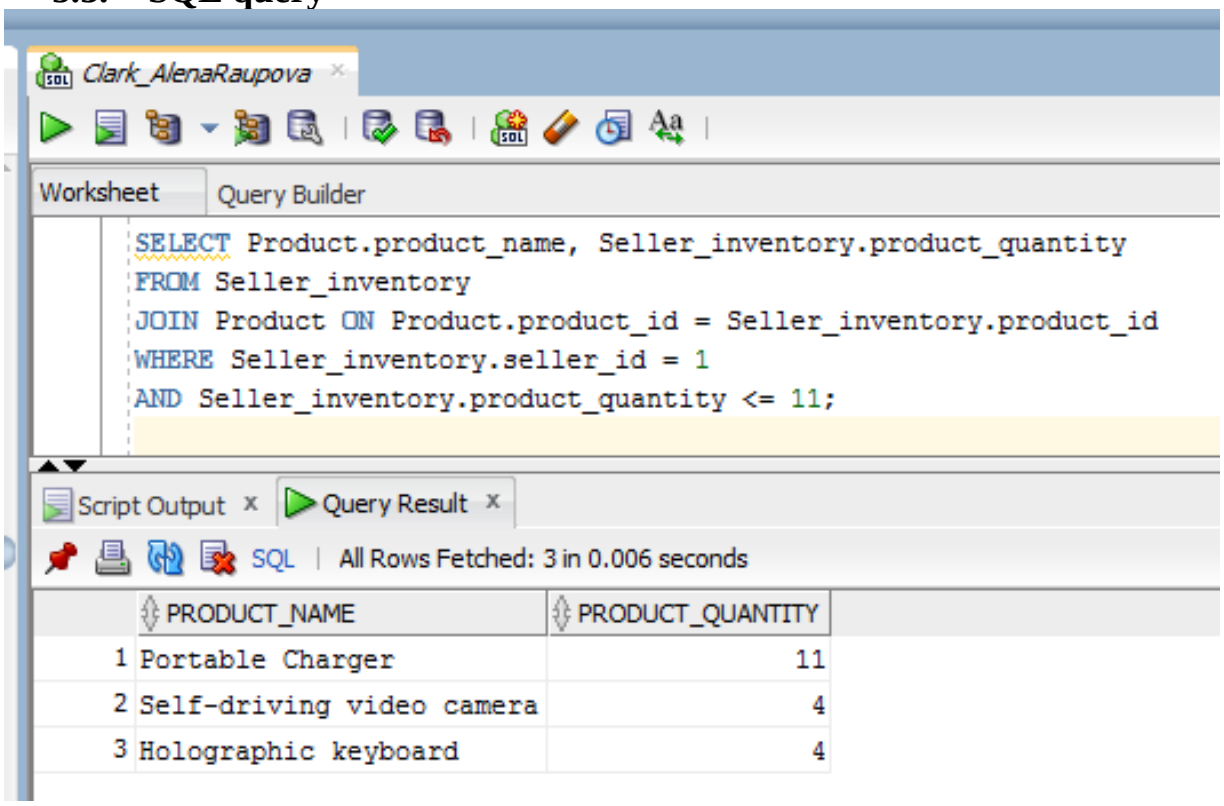
quantity of the product, we need to add the quantity of the product received from the seller to the already available quantity of this product from this seller in his inventory.

5.2. Use of the stored procedure





5.3. SQL query


Clark_AlenaRaupova
Worksheet
Query Builder
SELECT Product.product_name, Seller_inventory.product_quantity
FROM Seller_inventory
JOIN Product ON Product.product_id = Seller_inventory.product_id
WHERE Seller_inventory.seller_id = 1
AND Seller_inventory.product_quantity <= 11;
Script Output
Query Result
SQL
All Rows Fetched: 3 in 0.006 seconds
PRODUCT_NAME
PRODUCT_QUANTITY
1 Portable Charger 11
2 Self-driving video camera 4
3 Holographic keyboard 4

6. Aspect 3

6.1. Creation of reusable stored procedure

Clark_AlenaRaupova

Worksheet | Query Builder

```
CREATE OR REPLACE PROCEDURE ADD_NEW_CONSUMER (
consumer_id_arg IN DECIMAL,
consumer_first_name_arg IN VARCHAR,
consumer_last_name_arg IN VARCHAR,
phone_number_arg IN VARCHAR,
email_address_arg IN VARCHAR,
address_id_arg IN DECIMAL,
address_line_1_arg IN VARCHAR,
address_line_2_arg IN VARCHAR,
city_arg IN VARCHAR,
state_arg IN VARCHAR,
zip_code_arg IN VARCHAR,
address_type_id_arg IN DECIMAL
)
IS
BEGIN
INSERT INTO Consumer (consumer_id, consumer_first_name,
consumer_last_name, phone_number, email_address)
VALUES (consumer_id_arg, consumer_first_name_arg,
consumer_last_name_arg, phone_number_arg, email_address_arg);
INSERT INTO Address (address_id, address_line_1, address_line_2,
city, state, zip_code, address_type_id, consumer_id)
VALUES (address_id_arg, address_line_1_arg, address_line_2_arg,
city_arg, state_arg, zip_code_arg, address_type_id_arg, consumer_id_arg);
END;
```

Script Output

Task completed in 0.038 seconds

```
Procedure ADD_NEW_CONSUMER compiled
```

The procedure "ADD_NEW_CONSUMER" creates a new record in the "Consumer" table and a new record in the "Address" table. The record in the "Address" table includes a consumer_id field that will associate this record with the consumer.

6.2. Use of the stored procedure

Clark_AlenaRaupova

Worksheet | Query Builder

```
BEGIN
ADD_NEW_CONSUMER (9, 'Alena', 'Raupova', '605-414-2147', 'araupova@gmail.com', 17,
'37275 St Rt 17m M', NULL, 'Middle Island', 'NY', '11953', 1);
END;
/
```

Script Output | Task completed in 0.202 seconds

```
PL/SQL procedure successfully completed.
```

Clark_AlenaRaupova

Worksheet | Query Builder

```
BEGIN
ADD_NEW_CONSUMER (10, 'Konnor', 'Moore', '410-655-8723', 'kmoore@gmail.com',
18, '74 S Westgate St', '#123', 'Albany', 'NY', '12204', 1);
END;
/
```

Script Output | Task completed in 0.056 seconds

```
PL/SQL procedure successfully completed.
```

6.3. SQL query

Clark_AlenaRaupova

Worksheet | Query Builder

```
SELECT consumer_last_name, count(consumer_last_name)
FROM Consumer
GROUP BY consumer_last_name
HAVING count(consumer_last_name) > 4;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.088 seconds

	CONSUMER_LAST_NAME	COUNT(CONSUMER_LAST_NAME)
1	Moore	5

7. Aspect 4

7.1. Creation of tables and constraints

7.1.1. Tables

Clark_AlenaRaupova

Worksheet | Query Builder

```
CREATE TABLE Consumer_selection (
seller_inventory_id DECIMAL(10) NOT NULL,
order_id DECIMAL(10) NOT NULL,
consumer_selection_quantity DECIMAL(10) NOT NULL,
PRIMARY KEY(seller_inventory_id, order_id)
);

CREATE TABLE Consumer_order (
order_id DECIMAL(10) PRIMARY KEY,
shipping_speed VARCHAR(30) NOT NULL,
order_price DECIMAL(12,2) NOT NULL,
consumer_id DECIMAL(10) NOT NULL,
package_id DECIMAL(10),
payment_id DECIMAL(10),
address_id DECIMAL(10)
);

CREATE TABLE Payment (
payment_id DECIMAL(10) PRIMARY KEY,
payment_date DATE,
payment_amount DECIMAL(12,2),
payment_status VARCHAR (10),
order_id DECIMAL(10) NOT NULL
);
```

Script Output

Task completed in 0.065 seconds

```
Table CONSUMER_SELECTION created.

Table CONSUMER_ORDER created.

Table PAYMENT created.
```

7.1.2. Constraints

Clark_AlenaRaupova

Worksheet | Query Builder

```
ALTER TABLE Consumer_selection
ADD CONSTRAINT consumer_selection_seller_inventory_fk
FOREIGN KEY(seller_inventory_id)
REFERENCES Seller_inventory(seller_inventory_id);
```

Script Output

Task completed in 0.068 seconds

```
Table CONSUMER_SELECTION altered.
```

Clark_AlenaRaupova

Worksheet | Query Builder

```
ALTER TABLE Consumer_selection
ADD CONSTRAINT consumer_selection_consumer_order_fk
FOREIGN KEY(order_id)
REFERENCES Consumer_order(order_id);

ALTER TABLE Consumer_order
ADD CONSTRAINT consumer_order_consumer_fk
FOREIGN KEY(consumer_id)
REFERENCES Consumer(consumer_id);

ALTER TABLE Consumer_order
ADD CONSTRAINT consumer_order_address_fk
FOREIGN KEY(address_id)
REFERENCES Address(address_id);
```

Script Output

Task completed in 0.058 seconds

```
Table CONSUMER_SELECTION altered.

Table CONSUMER_ORDER altered.

Table CONSUMER_ORDER altered.
```

Clark_AlenaRaupova

Worksheet | Query Builder

```
ALTER TABLE Payment
ADD CONSTRAINT payment_consumer_order_fk
FOREIGN KEY(order_id)
REFERENCES Consumer_order(order_id);
```

Script Output

Task completed in 0.077 seconds

```
Table PAYMENT altered.
```

7.2. Creation of reusable stored procedure

Clark_AlenaRaupova

Worksheet | Query Builder

```
CREATE OR REPLACE PROCEDURE CONSUMER_PURCHASES_PRODUCT (
order_id_arg IN DECIMAL,
consumer_id_arg IN DECIMAL,
seller_inventory_id_arg IN DECIMAL,
product_price_arg IN DECIMAL,
consumer_selection_quantity_arg IN DECIMAL,
shipping_speed_arg IN VARCHAR,
payment_id_arg IN DECIMAL
)
IS
BEGIN
INSERT INTO Consumer_order (order_id, shipping_speed, order_price,
consumer_id, package_id, payment_id, address_id)
VALUES (order_id_arg, shipping_speed_arg, (product_price_arg*consumer_selection_quantity_arg),
consumer_id_arg, NULL, payment_id_arg, NULL);
INSERT INTO Consumer_selection (seller_inventory_id, order_id, consumer_selection_quantity)
VALUES (seller_inventory_id_arg, order_id_arg, consumer_selection_quantity_arg);
INSERT INTO Payment (payment_id, payment_date, payment_amount, payment_status, order_id)
VALUES (payment_id_arg, NULL, NULL, 'unpaid', order_id_arg);
UPDATE Seller_inventory SET product_quantity = (product_quantity - consumer_selection_quantity_arg)
WHERE seller_inventory_id = seller_inventory_id_arg;
END;
```

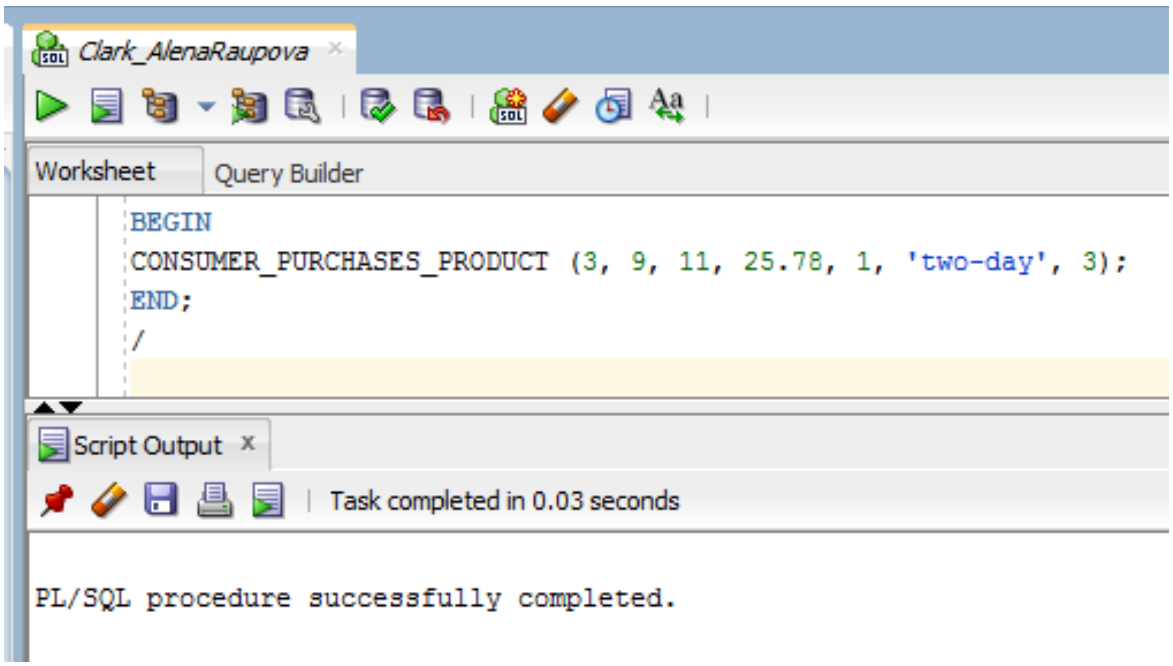
Script Output

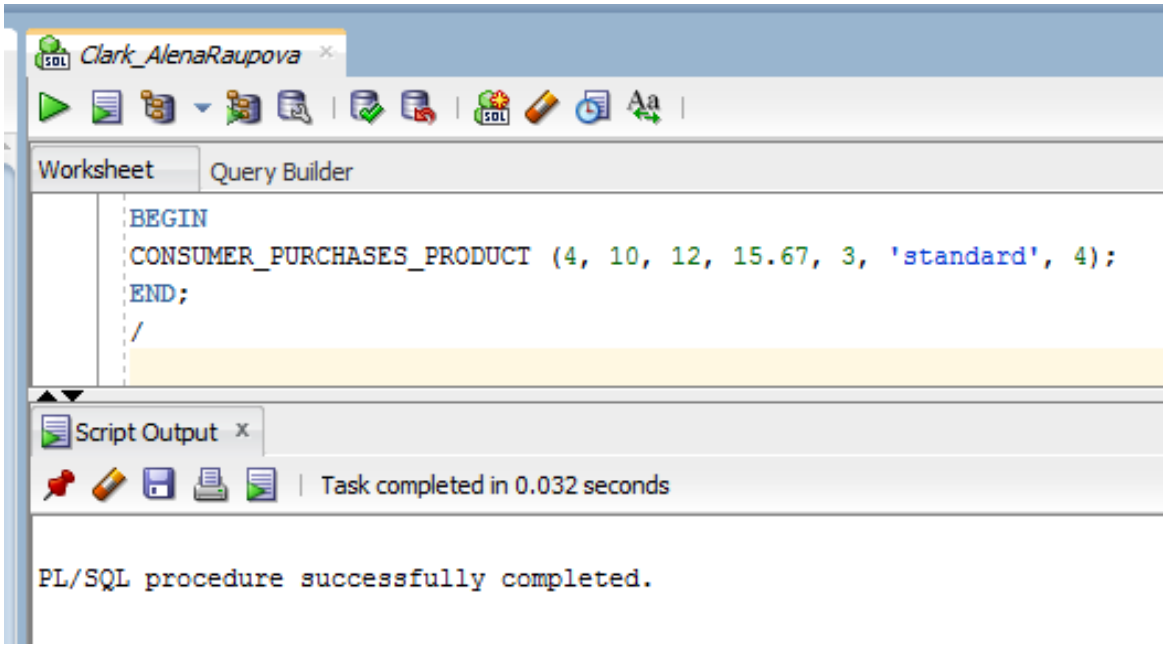
Task completed in 0.433 seconds

```
Procedure CONSUMER_PURCHASES_PRODUCT compiled
```

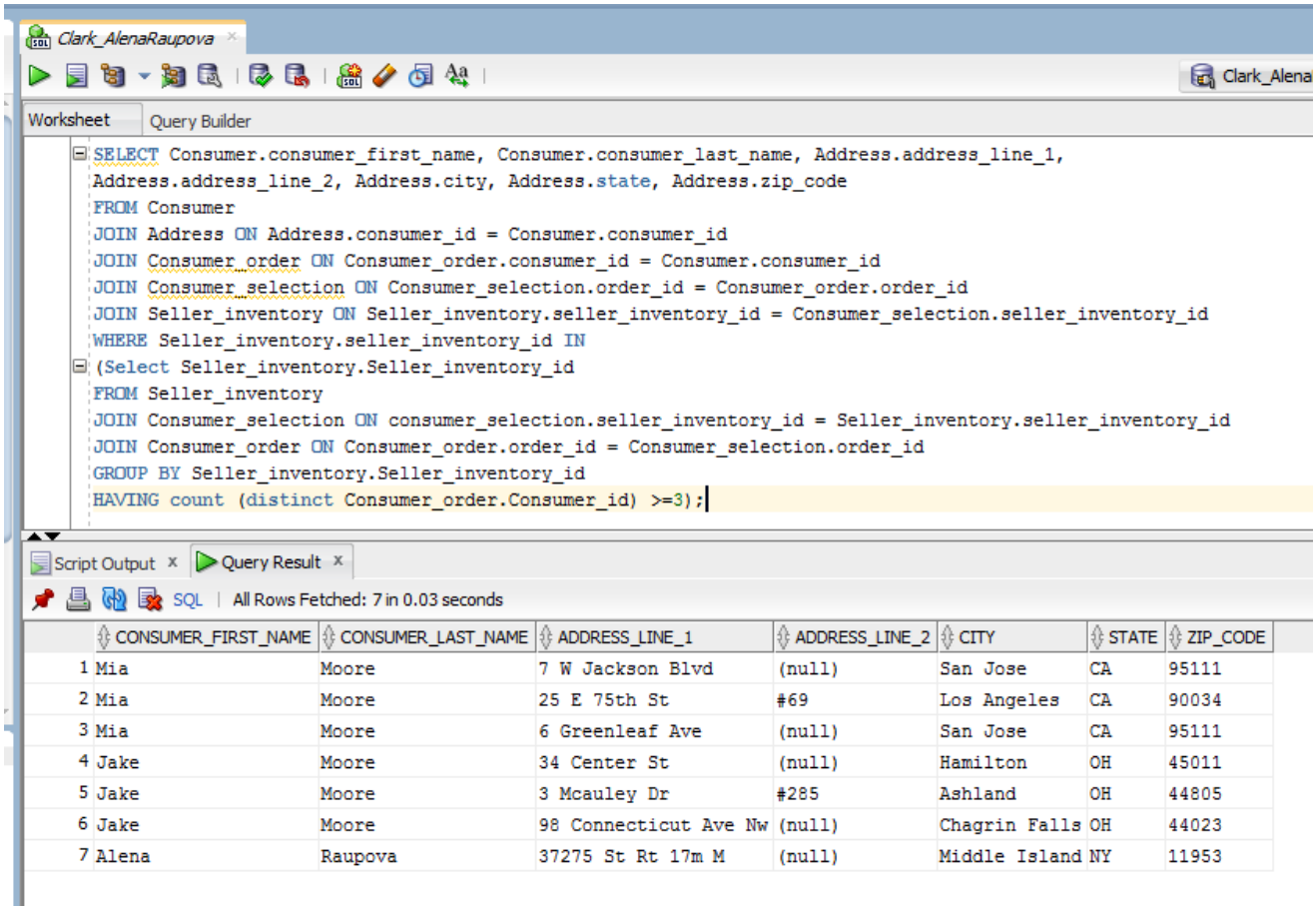
The procedure "CONSUMER_PURCHASES_PRODUCT" creates a record in the "Consumer_order" table, which has fields: order_id, shipping_speed (which the user selects according to his/her preferences), order_price (which will be calculated by multiplying the price of the product by the selected quantity of the product), consumer_id (which places the order), package_id (initially this field will be empty because the package has not yet been collected and packed), payment_id (a new record will be created in the "Payment" table, the consumer should already be able to pay for the order, and also this record is required in order to see if the order has been paid), address_id (initially this field will be empty in order to be able to select the desired type of address).
In parallel, an entry will be created in the "Consumer_selection" table, in which the following fields will be written: order_id (information about the order), seller_inventory_id (this is the inventory from which the product will be purchased), consumer_selection_quantity (the quantity of the selected product from the inventory).
Earlier I said that the procedure creates a new record in the "Payment" table. The payment_date and payment_amount fields will initially be null, since initially the consumer only places the order, not pay for it.
The record in the "Seller_inventory" table is also updated. This is necessary in order to reduce the quantity of the product in the seller's inventory by the quantity of the product that the consumer has chosen to order.

7.3. Use of the stored procedure





7.4. SQL query



	CONSUMER_FIRST_NAME	CONSUMER_LAST_NAME	ADDRESS_LINE_1	ADDRESS_LINE_2	CITY	STATE	ZIP_CODE
1	Mia	Moore	7 W Jackson Blvd	(null)	San Jose	CA	95111
2	Mia	Moore	25 E 75th St	#69	Los Angeles	CA	90034
3	Mia	Moore	6 Greenleaf Ave	(null)	San Jose	CA	95111
4	Jake	Moore	34 Center St	(null)	Hamilton	OH	45011
5	Jake	Moore	3 Mcauley Dr	#285	Ashland	OH	44805
6	Jake	Moore	98 Connecticut Ave Nw	(null)	Chagrin Falls	OH	44023
7	Alena	Raupova	37275 St Rt 17m M	(null)	Middle Island	NY	11953

This request has two parts.
The nested subquery gives information about the inventories that were selected to order by three or more different users.

The external query displays the first names, last names, addresses of these users. As a result of the query, some users have multiple addresses, because these users have several different types of addresses in the "Address" table.

8. Aspect 5

8.1. Creation of reusable stored procedure

Clark_AlenaRaupova

Worksheet | Query Builder

```
CREATE OR REPLACE PROCEDURE PRODUCT_SHIPMENT (
order_id_arg IN DECIMAL,
payment_id_arg IN DECIMAL,
payment_date_arg IN DATE,
payment_amount_arg IN DECIMAL,
payment_status_arg IN VARCHAR,
package_id_arg IN DECIMAL,
tracking_id_arg IN DECIMAL,
shipment_status_arg IN VARCHAR,
shipment_date_arg IN DATE,
consumer_id_arg IN DECIMAL,
address_type_name_arg IN VARCHAR
)
IS
BEGIN
UPDATE Payment
SET payment_date = payment_date_arg,
payment_amount = payment_amount_arg,
payment_status = payment_status_arg
WHERE payment_id = payment_id_arg
AND order_id = order_id_arg;
UPDATE Consumer_order
SET address_id = (Select Address.address_id
FROM Address
JOIN Address_type ON Address_type.address_type_id=Address.address_type_id
WHERE consumer_id = consumer_id_arg
AND Address_type.address_type_name = address_type_name_arg)
WHERE order_id = order_id_arg;
INSERT INTO Package (package_id, tracking_id, shipment_status, shipment_date, order_id)
VALUES (package_id_arg, tracking_id_arg, shipment_status_arg, shipment_date_arg, order_id_arg);
UPDATE Consumer_order
SET package_id = package_id_arg
WHERE order_id = order_id_arg;
END;
```

Script Output

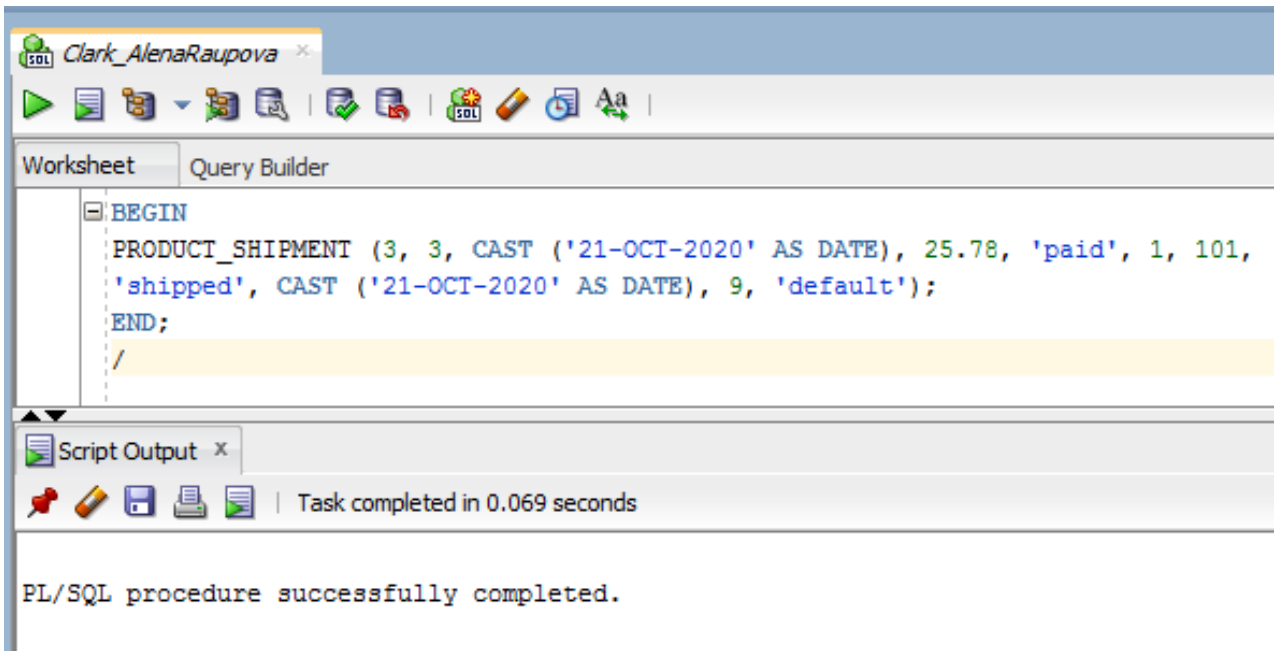
Task completed in 0.457 seconds

```
Procedure PRODUCT_SHIPMENT compiled
```

First of all, it should be clarified that the package should be sent only after the consumer pays for the order. Therefore, the "Product_shipment" procedure first updates the date, amount, and payment status in the "Payment" table.

Then the procedure updates the record in the "Consumer_order" table. First, the address_id field is updated in the record. Aspect 5 requires that the package should be sent to an address of type "default". But if we need to send a package to an address with a different type of address, then this procedure will allow us to specify an argument with a different type of address.
The procedure also creates a record in the "Package" table, which stores important information about the shipment (tracking_id, shipment_status, shippment_date, order_id, to which the package belongs).
After the record is created in the "Package" table, the record in the "Consumer_order" table is updated, in which the package_id field is updated, so that we can track all the necessary information about the delivery of this order (for example, the tracking number).

8.2. Use of the stored procedure



Clark_AlenaRaupova

Worksheet | Query Builder

```
BEGIN
PRODUCT_SHIPMENT (4, 4, CAST ('21-OCT-2020' AS DATE), 47.01, 'paid', 2, 201,
'shipped',CAST ('21-OCT-2020' AS DATE), 10, 'default');
END;
/
```

Script Output

Task completed in 0.061 seconds

```
PL/SQL procedure successfully completed.
```

8.3. SQL query

Clark_AlenaRaupova

Worksheet | Query Builder

```
SELECT Product.product_name, sum(Consumer_order.order_price) as sum_order_price
FROM Product
JOIN Seller_inventory ON Seller_inventory.product_id = Product.product_id
JOIN Consumer_selection ON Consumer_selection.seller_inventory_id = Seller_inventory.seller_inventory_id
JOIN Consumer_order ON Consumer_order.order_id = Consumer_selection.order_id
JOIN Package ON Package.order_id = Consumer_order.order_id
WHERE Package.shipment_date BETWEEN CAST('21-OCT-2020' AS DATE) AND CAST('25-OCT-2020' AS DATE)
AND address_id IN (SELECT address_id
FROM Address
WHERE state IN ('NY','CA'))
GROUP BY Product.product_name;
```

Script Output | Query Result

SQL | All Rows Fetched: 2 in 0.154 seconds

	PRODUCT_NAME	SUM_ORDER_PRICE
1	Self-driving video camera	25.78
2	Holographic keyboard	47.01

This query finds the names of products that were shipped from October 21, 2020 to October 25, 2020 and the state of dispatch was NY or CA. In addition to the name of such products, the query result displays the summed amounts of all orders for these products that meet the specified conditions.

9. The creation of the index

First of all, all primary key columns already have been indexed by the database. Here is the list of them:

Primary key column	**Description**
Seller.seller_id	This is the primary key of the Seller table.
Seller_inventory.seller_inventory_id	This is the primary key of the Seller_inventory table.
Product.product_id	This is the primary key of the Product table.
Product_category.product_category_id	This is the primary key of the Product_category table.
Consumer_selection.seller_inventory_id, Consumer_selection.order_id	This is the primary key of the Consumer_selection table.
Consumer.consumer_id	This is the primary key of the Consumer table.
Address.address_id	This is the primary key of the Address tabe.
Address_type.address_type_id	This is the primary key of the Address_type table.
Consumer_order.order_id	This is the primary key of the Consumer_order table.
Package.package_id	This is the primary key of the Package table.
Payment.payment_id	This is the primary key of the Payment table.

Second, foreign key columns should always be indexed. Here is the list of them:

Foreign key column	**Description**
Seller_inventory.product_id	This foreign key in the Seller_inventory table references the Product table. The index is non-unique since many inventories can store the same product.
Seller_inventory.seller_id	This foreign key in the Seller_inventory table references the Seller table. The index is non-unique since many inventories can belong to the same seller.
Product.product_category_id	This foreign key in the Product table references the Product_category

	table. The index is non-unique since many products can belong to the same category.
Consumer_selection.seller_inventory_id	This foreign key in the Consumer_selection table references the Seller_inventory table. The index is non-unique since many consumer selections can have the same seller inventory (many consumers can select the product from the same inventory).
Consumer_selection.order_id	This foreign key in the Consumer_selection table references Consumer_order table. The index is non-unique since many consumer_selection can refer to the same order (order can include many consumer_selections).
Address.address_type_id	This foreign key in the Address table references the Address_type table. The index is non-unique since many addresses can have the same address type.
Address.consumer_id	This foreign key in the Address table references the Consumer table. The index is non-unique since one consumer can have many addresses.
Consumer_order.consumer_id	This foreign key in the Consumer_order table references the Consumer table. The index is non-unique since one consumer can place many orders.
Consumer_order.package_id	This foreign key in the Consumer_order table references the Package table. The index is unique since many different consumer orders cannot be sent in one package.
Consumer_order.payment_id	This foreign key in the Consumer_order table references the Payment table. The index is unique since many different consumer orders cannot be paid by the same payment.

Consumer_order.address_id	This foreign key in the Consumer_order table references the Address table. The index is non-unique since many consumer orders can have the same address.
Package.order_id	This foreign key in the Package table references the Consumer_order table. The index is non-unique since more than 1 package can refer to the same order (for example, if some part of the order was collected later).
Payment.order_id	This foreign key in the Payment table references the Consumer_order table. The index is non-unique since many payments can apply to the same order (for example, if consumer decided to pay in installments).

Third, columns in WHERE clauses and JOIN conditions are usually indexed in order to speed up the result. Let's consider them in more detail.

Index column	Description
Product_category.product_category_name	This column is in the WHERE clause (Aspect 1 query) in order to find the required category id. The database can contain many products that belong to different categories, so the search can be delayed if each product is checked. Product_category.product_category_name index will help speed up our search. Moreover, this index can be used in other queries as well. For example, in order to determine in which states products of a certain category are most often ordered. It is a unique index because categories cannot repeat.
Product.product_price	This column is in the WHERE clause (Aspect 1 query) in order to find which products have a price <= of the required value. The use of this index is justified, because we can quickly select from the database those

	products that have the price we need. It is a non-unique index because many products can have the same price.
seller_inventory.product_quantity	This column is in the WHERE clause (Aspect 2 query) in order to find those inventories that have in stock the quantity of goods that <= the required value. Therefore, we need to index this column in order to simplify and speed up our work. It is a non-unique index because many inventories can have the same product quantity.
Package.shipment_date	This column is in the WHERE clause (Aspect 5 query) in order to find packages that were sent on the requested dates. This index will be used frequently in various queries for analytical purposes. For example, find out the status of sending a package on a specific date. It is a non-unique index because many packages can be sent on the same date.
Address.state	This column is in the WHERE clause (Aspect 5 query) in order to find packages that have been sent to the requested states. Moreover, this index can be used to analyze payments by state, find out which category products are most often shipped in a certain state, how many customers live in a certain state, etc. It is a non-unique index because many addresses can have the same state.

Below I have created these indexes.

Clark_AlenaRaupova

Worksheet | Query Builder

```
CREATE UNIQUE INDEX product_category_name_index
ON Product_category (product_category_name);
```

Script Output

Task completed in 0.999 seconds

```
INDEX PRODUCT_CATEGORY_NAME_INDEX created.
```

Clark_AlenaRaupova

Worksheet | Query Builder

```
CREATE INDEX product_price_index
ON Product (product_price);
```

Script Output

Task completed in 0.27 seconds

```
Index PRODUCT_PRICE_INDEX created.
```

Clark_AlenaRaupova

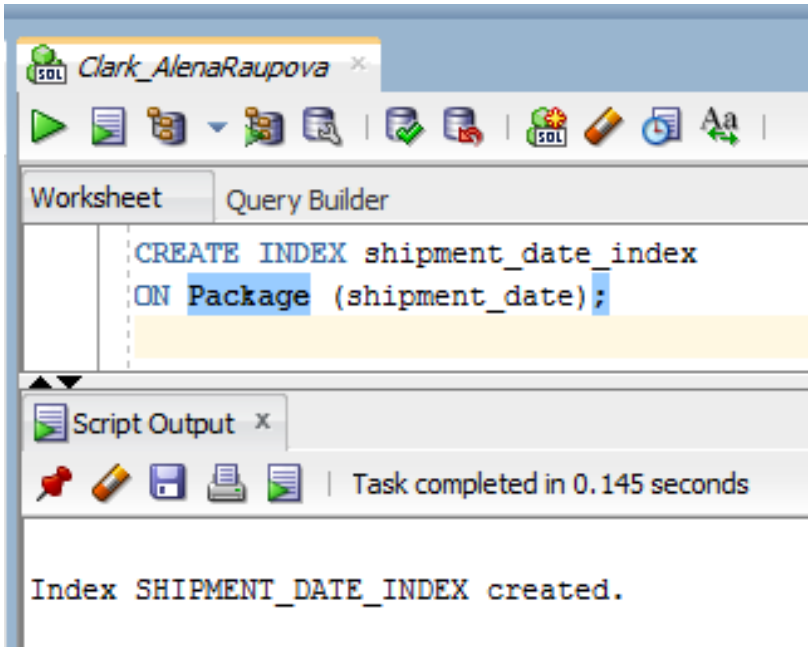
Worksheet | Query Builder

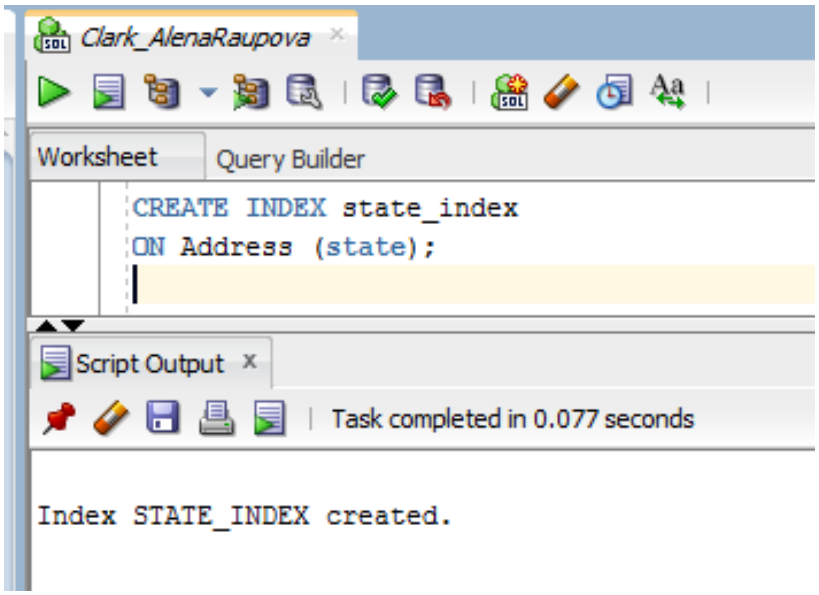
```
CREATE INDEX product_quantity_index
ON seller_inventory (product_quantity);
```

Script Output

Task completed in 0.168 seconds

```
Index PRODUCT_QUANTITY_INDEX created.
```





In addition, I also chose two more columns that should have an index, in my opinion. I do not use these columns in my queries. But such indexes can be useful in the future.

Index column	Assumption	Description
Payment.payment_date	This column can be used in queries in order to analyze how much money was received in a certain day / month / year. This is required for	I would create this index as a non-unique index since many payments can have the same payment date (many consumers can pay for

	financial analytics purposes. Also, this column can be used to view payment statuses for a specific date. Therefore, this column needs to be assigned an index so that important analytics are carried out much faster.	their orders at the same time).
Seller.seller_last_name, Seller.seller_first_name	Suppose there is a seller who has performed poorly and has been banned from selling products on the site. In this case, if someone tries to create a new account with the first name and last name of such a seller, then this will be prohibited. There are also many queries that will use the seller's first and last name for analytical purposes. For example, in order to calculate the total sales of a seller on the site or find information about which cities his products are most often ordered to. Such an index will speed up queries.	This is Multiple-column index. I use it to speed up lookups on both Seller.last_name and Seller.first_name. I would create this index as a unique index since usually sellers have a unique combination of first name and last name.

Below I have created these indexes.

Clark_AlenaRaupova

Worksheet | Query Builder

```
CREATE INDEX payment_date_index
ON Payment (payment_date);
```

Script Output

Task completed in 0.276 seconds

```
Index PAYMENT_DATE_INDEX created.
```

Clark_AlenaRaupova

Worksheet | Query Builder

```
CREATE UNIQUE INDEX seller_name_index
ON Seller (seller_last_name, seller_first_name);
```

Script Output

Task completed in 0.345 seconds

```
INDEX SELLER_NAME_INDEX created.
```