Group 15

CS5322 Project 1 Report

Members:

Name	Matric
Chun Hong Wei	A0167886E
Chew Zheng Xiong	A0167003R
Ng Jong Ray, Edward	A0216695U
Cheng Xianhao	A0167000X
Li YingHui	A0112832R

Contribution statement: Everyone had an equal and fair contribution

Introduction

Ecommerce platforms play a pivotal role in our lives today. As such, we have decided to implement our project in Oracle based on an ecommerce use case, by implementing Oracle's Virtual Private Database (VPD) to demonstrate access controls that were previously taught in lecture. This seeks to demonstrate the importance of access and information control to different roles in the ecommerce use case.

Application

This ecommerce platform database prototype is designed to serve as a holistic digital marketplace which is capable of catering to a broad spectrum of customers, vendors, and administrative personnel.

During registration, users can sign up their role as Customer, Seller or Admin.

As a customer, he/she is allowed to browse products, make purchases, track order status, read all reviews and share corresponding feedback once the product arrives. Moreover, the customer data remains confidential as the application only grants customers to view and update their respective user information, order details, payment information.

Sellers are the cornerstone of the ecommerce platform, they are granted admin rights to manage their products, i.e., adjust prices, update stock. In addition, sellers are able to track order status associated with their own products, and access transactional specifics excluding sensitive payment method details.

Additionally, the application also furnishes sellers with insights from aggregated customer feedback. It also embeds a robust support infrastructure via the Support Cases table. When users encounter any issues regarding their experience on the ecommerce platform, they have the ability to initiate a support request for the issue they are facing. This will subsequently be delegated to the support cases team which will manage the assigned support cases promptly.

ER and Tables

At the core of the e-commerce database design is the USERS table, compartmentalised by the attribute user_type into distinct roles of CUSTOMER, SELLER, and ADMIN. Each user, irrespective of their role, is uniquely identified by their id and has associated credentials like username and their hashed password for authentication. The username here corresponds to the database username that is assigned to each user and is unique. The team also understands that the current industry standard is to save the user's passwords in the database by hashing the salted password. However, as the focus of this prototype is to implement Oracle VPD, our team has just hashed the password with SHA256 to showcase how it could be done in our database implementation. Lastly, the account_standing attribute ascertains the current status of the user, indicating whether they are actively using the platform or have been marked for deletion.

Next, the PRODUCTS table allows the sellers to list their items for purchase on the platform. The table consists of a unique id, detailed descriptions, pricing, amount of stocks, and a product category. The seller_id enables a direct link to the USERS table, this ensures the seller's product ownership and management of their product sales on the ecommerce platform.

Following which, the ORDERS table, which has a pivotal role on the ecommerce platform. Each order has a unique id, linking the order to the associated user_id. The ORDERS table also encompasses timestamps and status indicators to illustrate the order's progress from initiation to completion, or even potential cancellation.

The ORDER_DETAILS table shows past and current orders. The table lists the products purchased, quantities, and the purchase price. This table shares a many-to-one relationship with the ORDERS table

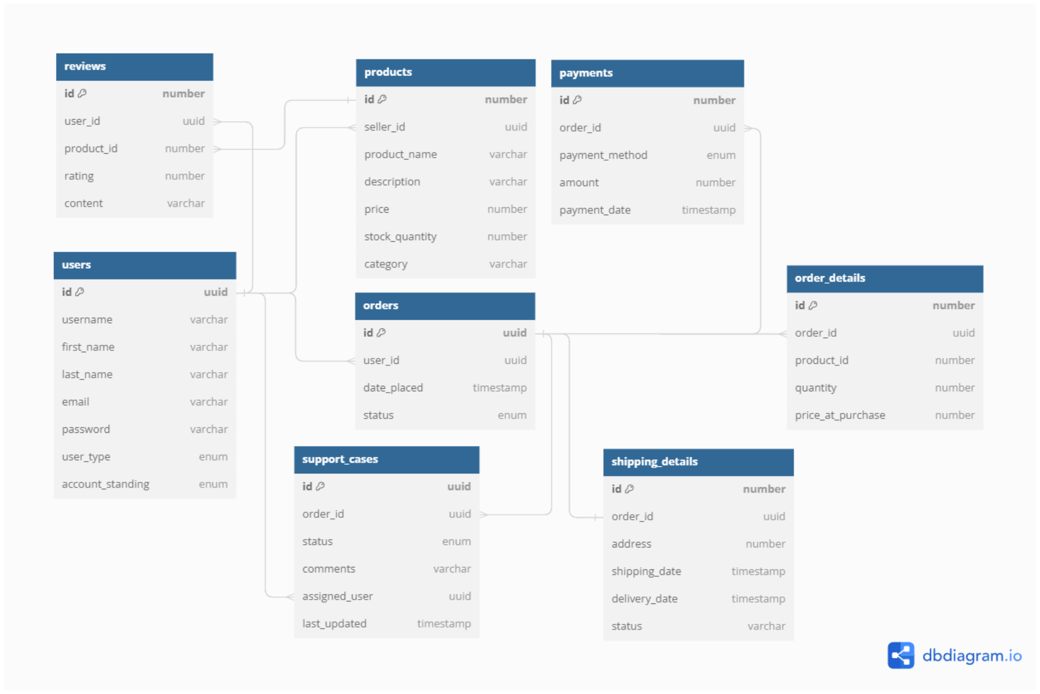


Figure 1: ER Table Design for Ecommerce Database

Customer feedback is an integral part of the e-commerce platform. The REVIEWS table captures the product reviews that are shared by the users on the platform. This also enables sellers to locate feedback relating to their products to act on it if required.

All the financial transactions on the ecommerce platform are immediately documented in the PAYMENTS table. The PAYMENTS table captures the payment method, amount transacted, and timestamps of each purchase. The table is also directly associated with the order that has been placed by the user.

SHIPPING_DETAILS table ensures that the products ordered by customers safely reach their doorstep. Every order has an associated shipping address, estimated shipping time delivery dates and the delivery status. All this vital information is captured in the table to ensure that the logistical component of the ecommerce platform is efficiently carried out.

Lastly, in order to provide a seamless user experience on the ecommerce platform, the SUPPORT_CASES table was developed to support user queries and issues faced on their transactions. Customers are able to raise support issue tickets that are then assigned to the support team of the ecommerce platform for prompt follow-up and subsequently, resolution.

Access to the tables mentioned above are governed by discretionary access control (DAC). DAC policies applied to each table can be found in table 1.

Table	**User Type**		
	Admin	*Seller*	*Customer*
USERS	S, U, I	S, U, I	S, U, I
PRODUCTS	S, U, I	S, U, I	S
ORDERS	S, U, I	S	S, I

ORDER_ DETAILS	S, U, I	S	S, I
PAYMENTS	S, U, I	S	S, I
REVIEWS	S, U, I	S	S, U, I
SHIPPING_ DETAILS	S, U, I	S, U	S, I
SUPPORT_ CASES	S, U, I	S	S
S = SELECT, U = UPDATE, I = INSERT			

Table 1: DAC policies applied to each table

Notice that DELETE operations are not part of any DAC policies – this is because the team has adopted a soft-delete approach where deleted entries are not physically removed from the database store, but merely indicated as such in the table. For instance, a deleted user in the USER table will have its account_standing attribute set to 'DELETED'.

A more detailed ER model representing the above entities and relationships can be found in Figure 1 and Appendix A of this report.

Security Policies

Although discretionary access policies are able to determine the access of users on each table, they are not able to efficiently restrict access to each entry of a table, unless a view is created for each user for each table. Given the sensitive nature of the data present in our ecommerce application, a stricter governance around access to the data is needed – it will be a clear violation to many privacy regulations if users are able to access each other's account information in the USERS table. Thus, a fine-grained access mechanism is required where access to each entry of a table is provisioned based on each user's role and identity. For Oracle databases, such policies can be achieved through VPD. An advantage of utilising VPD in our current implementation is that most of the access control can occur transparently without the user's knowledge. This allows users to simply query the database without needing extra knowledge, thus enhancing ease of use.

Database Context

Before delving deeper into the VPD policies implemented for each table, we will first discuss the properties of users that are applied in the VPD policies to restrict a user's access to an entry in each table.

First is the username of the user. This is both the username that is captured in the USERS table, and the database account's username as discussed in the previous section. This property can be retrieved by database context through the default USERENV namespace with the SESSION_USER parameter.

Next, is the role of the user. This is captured in the USER_TYPE column of the USERS table. A context is created to store this property and can be retrieved through the ECOMMERCE_ USER_TYPE namespace with the USER_TYPE parameter.

Finally, the user ID which is referenced in the majority of our VPD policies. This is the primary key of the USER table, and like the role property, a context is created to store this property. This can be retrieved through the same ECOMMERCE_USER_TYPE namespace with the USER_ID parameter.

Both the user ID and role properties are loaded automatically into the context through the use of database logon triggers. A copy of the SQL statements detailing the creation of contexts, packages and triggers can be found in Appendix B of this report.

VPD Implementation

Every user should be in-charge of and privy to their and only their user profiles. This means that regular users should only be able to access and modify their respective account details. Furthermore, as the access controls differ between a seller and a customer, there needs to be tight governance over changing an account type. Such operations should only be carried out by an application admin where it is expected that privileges are elevated and commands executed are deliberate. These requirements result in the following VPD policies being applied on the USERS table:

1. Customer and seller users can only view and update their own user entry.
2. Admin users can view but not update all user entries.
3. USER_TYPE column of each entry can only be updated by admin users.

To elaborate on our design choice, the above point #1 is designed with two policies instead of one to make the policies more modular. If they are combined within one single policy, modification of only one of either customer or seller rules will involve the other to be affected during tests.

Furthermore, the policies were created with scalability in mind. Recognising that multiple policies can append their predicates to one query, our team was careful to ensure that conditions that did not meet the criteria of its specific policy would return predicates that do not add value. An example would be returning *`ELSE return '1=1',`* continuing the where clause to allow for additional predicates from other policies to be added.

Implementation of the above VPD policies can be found in Appendix C.

User activities on an ecommerce platform can be used to profile the users by tracking their order history which provides a combination of data points, such as product category, purchase frequency etc that may be unique to the user. As such, this information should be treated with a need-to-know basis – customers should only be able to view their own orders, while sellers should only be able to view orders containing their products. Application admins however should have access to all data to provide support if there are any discrepancies in the order details. Both ORDER and ORDER_DETAILS tables contain information related to a user's purchases on the platform, resulting in the following VPD policies being applied:

1. Customer users can only view orders that have been placed by themselves.
2. Customer users can only view order details belonging to orders that have been placed by themselves.
3. Seller users can only view orders that contain their products.
4. Seller users can only view order details of their own products.
5. Admin users can view all orders and order details.

It is worth highlighting that the difference between the VPDs applied for ORDERS and ORDER_DETAILS tables lies in the complexity of the policy. For the ORDERS table, as the user ID is already embedded as part of the table, it is relatively easy to construct a VPD by directly referencing the user ID column, as in the case of the table's VPD policy for Customer users. In the ORDER_DETAILS table however, as user ID is not captured as part of the tuples, joins with other tables are required to determine the associated user ID of each tuple. For instance, to determine a customer user's access, the VPD policy performs a join between the ORDER_DETAILS and ORDERS tables to determine the associated ORDERS entry for each ORDER_DETAILS entry, and since the ORDERS entry contain the user ID of the customer, access to the ORDER_DETAILS entry can then be determined accordingly. As we will see in subsequent examples, most VPD policies involve performing some form of join given the prevalence of normalisation in relational databases.

We also understand that additional conditions that allow the ecommerce app to function such as allowing sellers to change the status to 'REFUNDED' on the orders table can have additional constraints, however, as this does not affect the security of the data, thus we decided against adding these into the order and order detail policies.

Implementation of the above VPD policies can be found in Appendix D and Appendix E.

While shipping details and payment information are not directly related to the user's activities on the platform, they often contain sensitive information, such as the address of the receiver and the payment method. To protect the privacy of our users, this information should also be kept on a need-to-know basis. In other words, customers' payment and shipping details are obscured from other customers, and sellers are not provided insights into specific payment methods, yet they can view transactions and shipping details related to their product listings. The following VPD policies are thus applied to SHIPPING_DETAILS and PAYMENTS table:

1. Customer users can only view payment details for orders that have been placed by them.
2. Seller users can only view payment details for orders that contain their products.
3. Seller cannot view the payment methods
4. Customer users can only view shipping details for orders that have been placed by them.
5. Seller users can only view shipping details for orders that contain their products.
6. Admin users can view all payment and shipping details.

In order to implement a precise access control mechanism to the ecommerce database, i.e., shield a specific sensitive attribute of a table, column-level VPD is applied to PAYMENTS table so that PAYMENT_METHOD column containing sensitive information is obscured for sellers. While point #2 above grants permission for sellers to view payment details only related to their own products, the attribute payment method belongs to confidential information and it is an inappropriate exposure to sellers. Hence, point #3 is enforced to the PAYMENT_METHOD column when it is referenced in a query from a seller.

Implementation of the above VPD policies can be found in Appendix G and Appendix H.

Product listings are the cornerstone of an ecommerce platform – they are the reason for the majority of the activities happening on the platform. It is thus important to ensure their data integrity as the opposite of that will lead to conflicts between sellers and customers, undermining the reliability and credibility of the platform. As such, sellers should only be able to manage their and only their own products. This gives rise to the following VPD policy on the PRODUCTS table:

1. Seller users can only update their own product.

This policy highlights the advantage of using VPD as a form of access control. Although the implementation of this policy involves a straightforward check: if the current user_type trying to update the products table is a 'SELLER' type, the predicate *SELLER_ID = SYS_CONTEXT('ecommerce_user_type','user_id')* is appended to the query. This predicate enables sellers to efficiently update their product details en masse, without affecting the products belonging to other sellers. For instance, a seller might wish to run a 30% discount on all their products can execute a query such as:

```
UPDATE ECOMMERCE.PRODUCTS
SET PRICE=PRICE*0.7;
```

With the VPD in place, the predicate will be transparently appended to the query, allowing the seller to easily update the prices of all their products without the need to list out the product_ids corresponding to their products in a WHERE clause. This significantly streamlines the seller's experience on the platform, especially when dealing with a large inventory of products.

Implementation of this VPD policy can be found in Appendix J.

Reviews serve an important role in rewarding good actors on the platform – sellers who provide quality products and services are likely to be given better reviews, garnering more support from other customers. In order to create an environment to ensure that genuine feedback is provided by customers, customers are only able to edit their own product reviews and are only allowed to submit their reviews after the delivery is completed. Sellers are only able to view the reviews relating to their own products while customer usernames or related information are not available. This is to preserve the anonymity and privacy of the customer to prevent the seller from performing any malicious acts in the event of poor reviews. The following VPD policies are applied to the REVIEWS table:

1. Customer user can only create reviews for products after they have been delivered
2. Customer users can only update their own reviews.
3. Seller users can only see reviews for their own products.

Point #1 above is done by checking INSERT queries into the REVIEWS table, to ensure that the current user has an order status which is 'COMPLETED' and that the product which they are going to review is in their order_details. Due to how other policies on the ORDERS table restrict customers to only view their own orders, the VPD for this point #1 can be simple to implement as the orders are already filtered to those belonging to the specific customer. This shows how existing policies can affect and hence streamline new policies to make them easy to create as well.

Moreover, this shows how VPDs can help enhance our ecommerce application. By allowing customers to INSERT a new review only after their order has been completed and received, we ensure that the feedback provided is based on their actual experience with the product. Preventing customers from adding a review before receiving the product improves the experience of other users who are viewing the product's reviews, as the reviews are now significantly more informative and helpful.

Implementation of these VPD policies can be found in Appendix F.

Expanding on the support received on the ecommerce platform, VPD policies have also been implemented to better support any issues that the customers face. The admin users are the only user group to make changes to support entries. This prevents the users or sellers from changing the reported support cases or interfering with the process. In addition, the admin users can assist in reassigning support personnel to ensure each support case is handled promptly and effectively. Lastly, the ecommerce platform also provides aggregated views for the support personnel to review their assigned support cases and their statuses. The following VPD policies are thus, applied to the SUPPORT_CASES table:

1. Admin users can view, update and create support cases.
2. Admin users can also re-assign support cases to ensure that sellers are informed.

Implementation for the above VPD policies can be found in Appendix I.

Wrapping up on the policy rationale behind the various VPDs implemented, the team has meticulously assessed the functional and unique VPDs to ensure that the ecommerce platform will not only serve its basic purpose, but to retain users and promote usage growth. In comparison against access control managed by DAC, VPDs will ensure that the ecommerce platform continues to remain both secure and efficient in its usage.

Results & Discussion

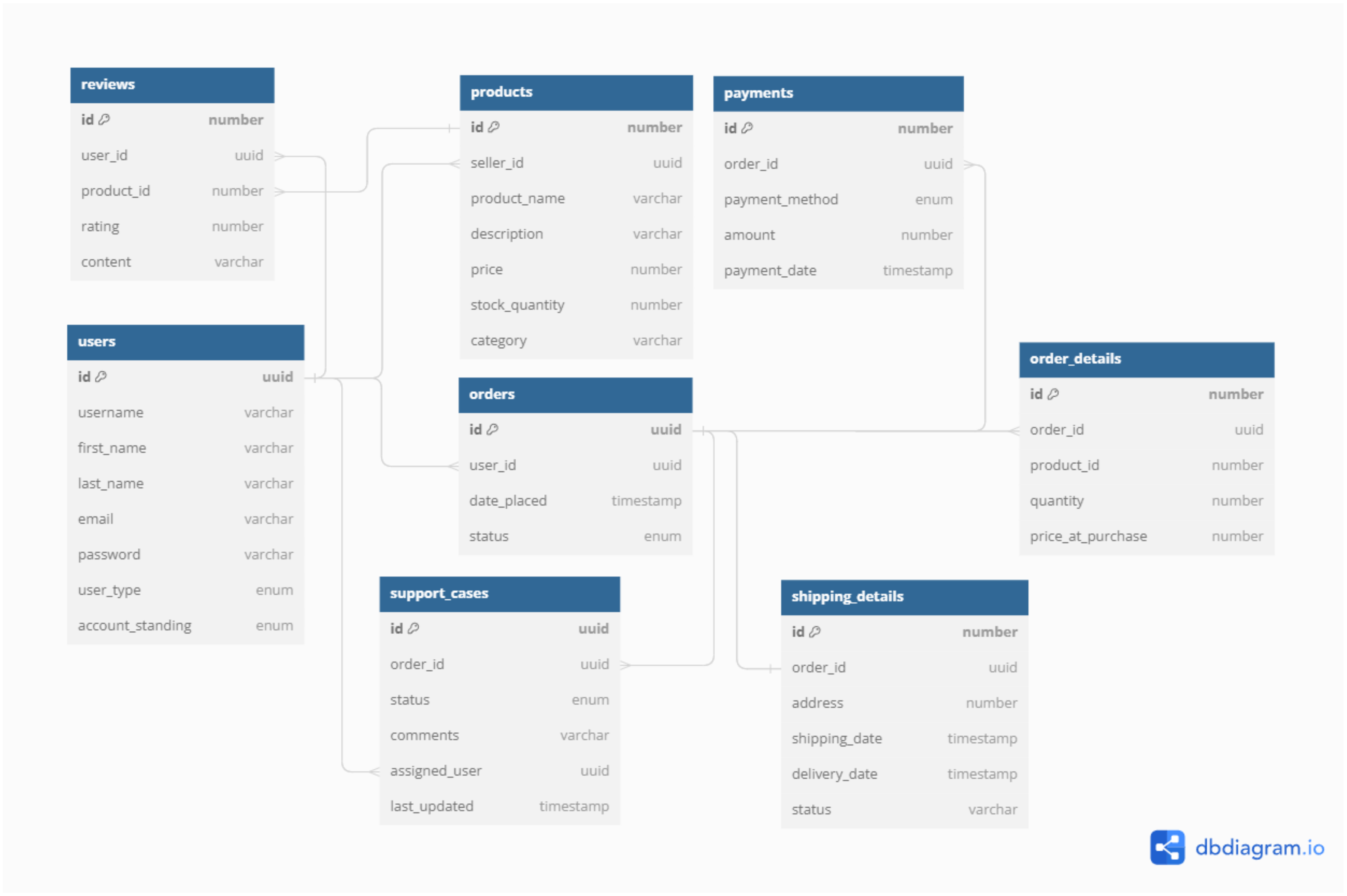
After integrating the VPD into the prototype ecommerce database, the team observed the following: Firstly, the VPDs were able to achieve their purpose in preventing exploits or unauthorised access from various user groups.

This ensured that access attempts are safe and secure for the ecommerce platform to continue its operations without compromising on its database security. Secondly, the performance of the database did not experience any slowness despite the myriad of VPD implemented. Although some query execution times were relatively slower than others, the execution speeds are still a testament of the security and operational needs are fulfilled successfully. Lastly, there is a need to consider the growth of the ecommerce platform. The scalability and complexity of the ecommerce platform's database must evolve in tandem with the implemented VPDs. Perhaps, new VPDs will have to be incorporated to consider the evolving structure and threats that loom ahead for ecommerce platforms.

Conclusion

In conclusion, the implementation of an ecommerce platform is supported by the robust database security administered through VPDs. The utilisation of Oracle VPDs enhances the user, seller and admin experience of the ecommerce platform. Effectively retaining these users to propel business growth. However, with cybersecurity threats consistently relooking at vulnerabilities in the systems, VPDs will also have to take into account the security of user operations beyond the data stored in the database itself.

Appendix A - ER Model



reviews	
id	number
user_id	uuid
product_id	number
rating	number
content	varchar

users	
id	uuid
username	varchar
first_name	varchar
last_name	varchar
email	varchar
password	varchar
user_type	enum
account_standing	enum

products	
id	number
seller_id	uuid
product_name	varchar
description	varchar
price	number
stock_quantity	number
category	varchar

orders	
id	uuid
user_id	uuid
date_placed	timestamp
status	enum

payments	
id	number
order_id	uuid
payment_method	enum
amount	number
payment_date	timestamp

order_details	
id	number
order_id	uuid
product_id	number
quantity	number
price_at_purchase	number

support_cases	
id	uuid
order_id	uuid
status	enum
comments	varchar
assigned_user	uuid
last_updated	timestamp

shipping_details	
id	number
order_id	uuid
address	number
shipping_date	timestamp
delivery_date	timestamp
status	varchar

Appendix B - Context Creation

```
CREATE OR REPLACE PACKAGE ecommerce.context_user_type_package AS
    PROCEDURE set_user_type_context;
    PROCEDURE set_user_id_context;
END;

CREATE OR REPLACE PACKAGE BODY ecommerce.context_user_type_package IS
    PROCEDURE set_user_type_context IS
        v_username  VARCHAR2(40);
        v_user_type VARCHAR2(12);
    BEGIN
        v_username := SYS_CONTEXT('USERENV','SESSION_USER');
        BEGIN
            SELECT user_type
            INTO   v_user_type
            FROM   ECOMMERCE.Users
            WHERE  UPPER(username) = v_username;

            DBMS_OUTPUT.PUT_LINE('set_user_type_context user_type: ' ||
v_user_type);

            DBMS_SESSION.set_context('ecommerce_user_type','user_type',
v_user_type);
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                DBMS_SESSION.set_context('ecommerce_user_type','user_type',
NULL);
        END;
    END set_user_type_context;

    PROCEDURE set_user_id_context IS
        v_username  VARCHAR2(40);
        v_user_id VARCHAR2(36);
    BEGIN
        v_username := SYS_CONTEXT('USERENV','SESSION_USER');
        BEGIN
            SELECT ID
            INTO   v_user_id
            FROM   ECOMMERCE.Users
            WHERE  UPPER(username) = v_username;

            DBMS_SESSION.set_context('ecommerce_user_type','user_id',
v_user_id);
```

```
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                DBMS_SESSION.set_context('ecommerce_user_type','user_id',
NULL);
        END;
    END set_user_id_context;
END context_user_type_package;

GRANT EXECUTE ON ecommerce.context_user_type_package TO PUBLIC;
CREATE PUBLIC SYNONYM context_user_type_package FOR
ecommerce.context_user_type_package;

CREATE OR REPLACE TRIGGER ecommerce.set_user_type_trigger AFTER LOGON ON
DATABASE
BEGIN
    ecommerce.context_user_type_package.set_user_type_context;
    ecommerce.context_user_type_package.set_user_id_context;
END;
```

Appendix C - VPD Policies on USERS Table

1. Customers can only view their own passwords and other information
2. Seller can only view their own passwords and other information
3. Admin can view the whole USERS table
4. Customers can only update their own passwords and other information excluding user roles
5. Seller can only update their own passwords and other information excluding user roles
6. Admin can only update their own passwords and other information
7. Only Admin can change user roles (e.g., promote a customer to a seller or vice-versa).

```
CREATE OR REPLACE FUNCTION ecommerce.customer_info_view (
      p_schema IN VARCHAR2, p_object IN VARCHAR2)
RETURN VARCHAR2 AS condition VARCHAR2(200);
BEGIN
    IF SYS_CONTEXT('ecommerce_user_type','user_type') = 'ADMIN' THEN
        RETURN '1=1';
    ELSE
        condition := 'UPPER(username) =
SYS_CONTEXT(''USERENV'',''SESSION_USER'')';
    END IF;
    DBMS_OUTPUT.PUT_LINE('customer_info_update return condition: ' ||
condition);
    RETURN condition;
END customer_info_view;

/

CREATE OR REPLACE FUNCTION ecommerce.customer_info_update (
      schema_var IN VARCHAR2, table_var IN VARCHAR2)
RETURN VARCHAR2 AS condition VARCHAR2(200);
BEGIN
    condition := 'UPPER(username) =
SYS_CONTEXT(''USERENV'',''SESSION_USER'')';
    DBMS_OUTPUT.PUT_LINE('customer_info_update return condition: ' ||
condition);
    RETURN condition;
END customer_info_update;

/

CREATE OR REPLACE FUNCTION ecommerce.role_update(
      p_schema IN VARCHAR2, p_object IN VARCHAR2)
RETURN VARCHAR2 AS
```

```
BEGIN
    IF SYS_CONTEXT('ecommerce_user_type','user_type') = 'ADMIN' THEN
        RETURN '1=1';
    ELSE
        RETURN '1=0';
    END IF;
END role_update;
/

BEGIN
    DBMS_RLS.ADD_POLICY(
        object_schema   => 'ECOMMERCE',
        object_name     => 'Users',
        policy_name     => 'customer_info_view_policy',
        function_schema => 'ECOMMERCE',
        policy_function => 'customer_info_view',
        statement_types => 'SELECT'
    );
END;

/

BEGIN
    DBMS_RLS.ADD_POLICY(
        object_schema   => 'ECOMMERCE',
        object_name     => 'Users',
        policy_name     => 'customer_info_update_policy',
        function_schema => 'ECOMMERCE',
        policy_function => 'customer_info_update',
        statement_types => 'UPDATE, DELETE',
        sec_relevant_cols => 'username, first_name, last_name, email,
password, account_standing',
        update_check    => true
    );
END;

/

BEGIN
    DBMS_RLS.ADD_POLICY(
        object_schema   => 'ECOMMERCE',
        object_name     => 'Users',
            policy_name     => 'role_update_policy',
            function_schema => 'ECOMMERCE',
            policy_function => 'role_update',
            statement_types => 'UPDATE',
        sec_relevant_cols => 'id, user_type',
        update_check => TRUE
       );
END;
```

Appendix D - VPD Policies on ORDERS Table

1. Admins have full access to the orders table
2. Customers can view only their orders
3. Customer can insert only orders tagged to their uuid (Add new order)
4. Sellers can view only the orders containing their products
5. Sellers can only update the orders containing their products (Change status)

```
CREATE OR REPLACE FUNCTION ecommerce.customer_order_policy (
    p_schema IN VARCHAR2,
    p_object IN VARCHAR2)
RETURN VARCHAR2 AS
    v_user_type VARCHAR2(12);
    v_user_id VARCHAR2(36);
BEGIN
    -- Identify user
v_user_id := SYS_CONTEXT('ecommerce_user_type', 'user_id');
v_user_type := SYS_CONTEXT('ecommerce_user_type', 'user_type');

    -- Return predicate to filter the orders by their ID
    IF v_user_type IN ('CUSTOMER') THEN
        RETURN 'USER_ID = ''' || v_user_id || '''';
    ELSIF v_user_type = 'ADMIN' THEN
        RETURN '1=1';
    ELSE
        RETURN '1=1';
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN '1=1';
    WHEN OTHERS THEN
        RETURN '1=1';
END customer_order_policy;

CREATE OR REPLACE FUNCTION ecommerce.seller_order_policy (
    p_schema IN VARCHAR2,
    p_object IN VARCHAR2)
RETURN VARCHAR2 AS
    v_user_type VARCHAR2(12);
    v_user_id VARCHAR2(36);
BEGIN
    -- Identify user
v_user_id := SYS_CONTEXT('ecommerce_user_type', 'user_id');
v_user_type := SYS_CONTEXT('ecommerce_user_type', 'user_type');

    -- If user is cust or seller, return predicate to filter the orders by
their ID
    IF v_user_type IN ('SELLER') THEN
```

```
        RETURN 'ID IN (SELECT od.order_id FROM ecommerce.order_details od
                             JOIN ecommerce.products p ON od.product_id =
p.ID
                             WHERE p.seller_id = ''' || v_user_id || ''')';
    ELSIF v_user_type = 'ADMIN' THEN
        RETURN '1=1';
    ELSE
        RETURN '1=1';
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN '1=1';  -- Return if user not found
    WHEN OTHERS THEN
        RETURN '1=1';
END seller_order_policy;



BEGIN
    DBMS_RLS.ADD_POLICY (
        object_schema   => 'ECOMMERCE',
        object_name     => 'ORDERS',
        policy_name     => 'CUSTOMER_ORDER_POLICY',
        function_schema => 'ECOMMERCE',
        policy_function => 'CUSTOMER_ORDER_POLICY',
        statement_types => 'SELECT',
        update_check    => TRUE
    );
END;

BEGIN
    DBMS_RLS.ADD_POLICY (
        object_schema   => 'ECOMMERCE',
        object_name     => 'ORDERS',
        policy_name     => 'SELLER_ORDER_POLICY',
        function_schema => 'ECOMMERCE',
        policy_function => 'SELLER_ORDER_POLICY',
        statement_types => 'SELECT, UPDATE',
        update_check    => TRUE
    );
END;
```

Appendix E - VPD Policies on ORDER_DETAILS Table

1. Customers can only view order details for orders that are placed by themselves
2. Customer can only create order detail entries that are associated to orders they have placed
3. Sellers can only view order details for orders that contain their products

```
CREATE OR REPLACE FUNCTION check_order_details_access(
    schema_name IN VARCHAR2,
    table_name IN VARCHAR2
) RETURN VARCHAR2
AS
    v_username  VARCHAR2(40);
    v_user_type VARCHAR2(12);
    v_user_id   VARCHAR2(36);
    v_condition VARCHAR2(4000);
BEGIN
    v_username := LOWER(SYS_CONTEXT('USERENV', 'SESSION_USER'));
    v_user_type := LOWER(SYS_CONTEXT('ecommerce_user_type', 'user_type'));
    v_user_id := SYS_CONTEXT('ecommerce_user_type', 'user_id');
    IF (v_username = 'ecommerce' OR v_username = 'system' OR v_user_type =
'admin') THEN
        v_condition := '1=1';
    ELSIF (v_user_type = 'customer') THEN
        v_condition := 'EXISTS (
                            SELECT 1 FROM orders o
                            WHERE order_id = o.id
                            AND o.user_id = ''' || v_user_id || '''
                        )';
    ELSE
        v_condition := 'EXISTS (
                            SELECT 1 FROM products p
                            WHERE p.seller_id = ''' || v_user_id || '''
                            AND product_id = p.id
                        )';
    END IF;
    RETURN v_condition;
END;
BEGIN
    DBMS_RLS.ADD_POLICY(
            object_schema => 'ECOMMERCE',
            object_name => 'order_details',
            policy_name => 'order_details_access_policy',
            function_schema => 'ECOMMERCE',
            policy_function => 'check_order_details_access',
            statement_types => 'SELECT, INSERT',
            update_check => true,
            enable => true
        );
END;
```

Appendix F - VPD Policies on REVIEWS Table

1. Sellers are only allowed view ratings for their own products but not individual review details
2. Customers can only insert a review after the delivery date
3. Customers can only update their reviews, admin can update all reviews

```
CREATE OR REPLACE FUNCTION ECOMMERCE.seller_review_policy (
    p_schema IN VARCHAR2,
    p_object IN VARCHAR2)
RETURN VARCHAR2 AS
    condition VARCHAR2(200);
BEGIN
    IF (SYS_CONTEXT('ecommerce_user_type','user_type') = 'SELLER') THEN
        condition := 'product_id IN (SELECT id FROM ECOMMERCE.products WHERE
seller_id = SYS_CONTEXT(''ECOMMERCE_USER_TYPE'', ''USER_ID''))';
    ELSE
        condition := '1=1';
    END IF;
    RETURN condition;
END seller_review_policy;

CREATE OR REPLACE FUNCTION review_after_delivery_check(
      ECOMMERCE IN VARCHAR2, REVIEWS IN VARCHAR2)
RETURN VARCHAR2 AS condition VARCHAR2 (200);
BEGIN
    condition:='PRODUCT_ID IN
    (SELECT o.product_id FROM orderandproductview o
       WHERE ''COMPLETED'' = o.status)';
      RETURN condition;
END review_after_delivery_check;

CREATE OR REPLACE FUNCTION update_review(
      ECOMMERCE IN VARCHAR2, REVIEWS IN VARCHAR2)
RETURN VARCHAR2 AS condition VARCHAR2 (200);
BEGIN
      IF SYS_CONTEXT('ecommerce_user_type','user_type') = 'ADMIN' THEN
       RETURN '1=1';
      ELSE
       condition := 'SYS_CONTEXT(''ecommerce_user_type'',''user_id'') =
USER_ID';
    END IF;
    RETURN condition;
END update_review;
```

```
/

BEGIN
    DBMS_RLS.ADD_POLICY(
        object_schema   => 'ECOMMERCE',
        object_name     => 'reviews',
        policy_name     => 'seller_review_policy',
        function_schema => 'ECOMMERCE',
        policy_function => 'seller_review_policy',
        statement_types => 'SELECT',
        update_check    => true
    );
END;

BEGIN
      DBMS_RLS.ADD_POLICY(
      object_schema => 'ECOMMERCE',
      object_name => 'REVIEWS',
      policy_name => 'review_after_delivery_check_policy',
      function_schema => 'ECOMMERCE',
      policy_function => 'review_after_delivery_check',
      statement_types => 'INSERT',
      update_check => TRUE);
END;

BEGIN
      DBMS_RLS.ADD_POLICY(
      object_schema => 'ECOMMERCE',
      object_name => 'REVIEWS',
      policy_name => 'update_review_policy',
      function_schema => 'ECOMMERCE',
      policy_function => 'update_review',
      statement_types => 'UPDATE');
END;
/
```

Appendix G - VPD Policies on PAYMENTS Table

1. Sellers cannot see paymentMethod but can see other attributes of PAYMENTS table
2. Customers can't view the payment information of other customers. For sellers, only view payments of orders on their products. Admin can view the whole payments table.

```
CREATE OR REPLACE FUNCTION ecommerce.seller_view_payment_info(
      p_schema IN VARCHAR2, p_object IN VARCHAR2)
RETURN VARCHAR2 AS
BEGIN
    IF (SYS_CONTEXT('ecommerce_user_type','user_type') = 'SELLER') THEN
        RETURN '1=0';
    ELSE
        RETURN '1=1';
    END IF;
END seller_view_payment_info;


CREATE OR REPLACE FUNCTION ecommerce.payments_policy (
    p_schema IN VARCHAR2,
    p_object IN VARCHAR2)
RETURN VARCHAR2 AS
    v_user_type VARCHAR2(12);
    v_user_id VARCHAR2(36);
BEGIN
    -- Identify the user type and ID based on the username
    v_user_id := SYS_CONTEXT('ecommerce_user_type', 'user_id');
    v_user_type := SYS_CONTEXT('ecommerce_user_type', 'user_type');
    -- If the user is a customer, return a predicate to filter the payments
by their ID
    IF v_user_type = 'CUSTOMER' THEN
        RETURN 'ORDER_ID IN (SELECT ID FROM ecommerce.orders WHERE USER_ID =
''' || v_user_id || ''')';
    ELSIF v_user_type = 'SELLER' THEN
        -- For sellers, return a predicate to filter payments of orders on
their products
        RETURN 'ORDER_ID IN (SELECT o.ID FROM ecommerce.orders o
                             JOIN ecommerce.order_details od ON o.ID =
od.order_id
                             JOIN ecommerce.products p ON od.product_id =
p.ID
                             WHERE p.seller_id = ''' || v_user_id || ''')';
    ELSE
        -- return no restrictions for other users.
        RETURN '1=1';
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
```

```
        RETURN '1=0';  -- Return if the user is not found, denying access
    WHEN OTHERS THEN
        RETURN '1=0';  -- Deny access in case of other exceptions
END payments_policy;
/

BEGIN
      DBMS_RLS.ADD_POLICY (
            object_schema    => 'ECOMMERCE',
            object_name      => 'Payments',
            policy_name      => 'seller_view_payment_info_policy',
            function_schema  => 'ECOMMERCE',
            policy_function  => 'seller_view_payment_info',
            statement_types  => 'SELECT',
            update_check     => false,
            sec_relevant_cols => 'payment_method',
            sec_relevant_cols_opt => dbms_rls.ALL_ROWS
       );
END;

BEGIN
    DBMS_RLS.ADD_POLICY (
        object_schema   => 'ECOMMERCE',
        object_name     => 'PAYMENTS',
        policy_name     => 'PAYMENTS_POLICY',
        function_schema => 'ECOMMERCE',
        policy_function => 'PAYMENTS_POLICY',
        statement_types => 'SELECT'
    );
END;
/
```

Appendix H - VPD Policies on SHIPPING_DETAILS Table

1. Customers can only see shipping details associated to the orders they have placed
2. Customer can only create shipping detail entries that are associated to the orders they have placed
3. Sellers can only see shipping details associated to the orders that contain their products

```
CREATE OR REPLACE FUNCTION check_shipping_details_access (
    schema_name IN VARCHAR2,
    table_name IN VARCHAR2
) RETURN VARCHAR2
AS
    v_username VARCHAR2(40);
    v_user_type VARCHAR2(12);
    v_user_id VARCHAR2(36);
    v_condition VARCHAR2(4000);
BEGIN
    v_username := LOWER(SYS_CONTEXT('USERENV','SESSION_USER'));
    v_user_type := LOWER(SYS_CONTEXT('ecommerce_user_type','user_type'));
    v_user_id := SYS_CONTEXT('ecommerce_user_type','user_id');
    IF (v_username = 'ecommerce' OR v_username = 'system' OR v_user_type =
'admin') THEN
        v_condition := '1=1';
    ELSIF (v_user_type = 'customer') THEN
        v_condition := 'EXISTS (
                            SELECT 1 FROM orders o
                            WHERE order_id = o.id
                            AND o.user_id = '''|| v_user_id ||'''
                        )';
    ELSE
        v_condition := 'order_id IN (
                            SELECT o.id FROM products p, orders o,
order_details od
                            WHERE p.seller_id = '''|| v_user_id ||'''
                            AND od.product_id = p.id
                            AND od.order_id = o.id
                        )';
    END IF;
    RETURN v_condition;
END;

BEGIN
```

```
    DBMS_RLS.ADD_POLICY(
            object_schema    => 'ECOMMERCE',
            object_name      => 'shipping_details',
            policy_name      => 'shipping_details_access_policy',
            function_schema  => 'ECOMMERCE',
            policy_function  => 'check_shipping_details_access',
            statement_types  => 'SELECT, INSERT',
            update_check     => true,
            enable           => true
        );
END;
```

Appendix I - VPD Policies on SUPPORT_CASES Table

1. Admin users can view, update and delete support cases.
2. Admin users can also re-assign support cases to ensure that sellers are informed.

```
CREATE OR REPLACE FUNCTION ECOMMERCE.support_cases_update_policy (
    p_schema IN VARCHAR2,
    p_object IN VARCHAR2)
RETURN VARCHAR2 AS
    condition VARCHAR2(200);
BEGIN
    IF SYS_CONTEXT('ecommerce_user_type', 'user_type') = 'ADMIN' THEN
        RETURN '1=1'; -- Admin has full access
    ELSE
        -- Non-admin users can't update or delete records
        condition := '1=0';
    END IF;

    DBMS_OUTPUT.PUT_LINE('support_cases_update_policy return condition: ' ||
condition);
    RETURN condition;
END support_cases_update_policy;
/

CREATE OR REPLACE FUNCTION ECOMMERCE.support_staff_case_policy (
    p_schema IN VARCHAR2,
    p_object IN VARCHAR2)
RETURN VARCHAR2 AS
    condition VARCHAR2(200);
BEGIN
    condition := 'assigned_user = SYS_CONTEXT(''ecommerce_user_type'',
''user_id'') OR SYS_CONTEXT(''ecommerce_user_type'', ''user_type'') =
''ADMIN''';
    RETURN condition;
END support_staff_case_policy;
/
```

Appendix J - VPD Policies on PRODUCTS table

1. Seller users can only update their own product.

```
CREATE OR REPLACE FUNCTION update_product(
      ECOMMERCE IN VARCHAR2, PRODUCTS IN VARCHAR2)
RETURN VARCHAR2 AS condition VARCHAR2 (200);
BEGIN
    IF SYS_CONTEXT('ecommerce_user_type','user_type') = 'ADMIN' THEN
      RETURN '1=1';
      ELSIF SYS_CONTEXT('ecommerce_user_type','user_type') = 'SELLER' THEN
      condition := 'SELLER_ID =
SYS_CONTEXT(''ecommerce_user_type'',''user_id'')';
      ELSE
      RETURN '1=0';
      END IF;
      RETURN condition;
END update_product;

BEGIN
    DBMS_RLS.ADD_POLICY(
    object_schema => 'ECOMMERCE',
    object_name => 'PRODUCTS',
    policy_name => 'update_product_policy',
       function_schema => 'ECOMMERCE',
    policy_function => 'update_product',
    statement_types => 'INSERT, UPDATE, DELETE',
    update_check => TRUE); --prevent others from also inserting into products
table
END;
```

Questions to ask prof:

1. **Efficiency and optimization of tables (eg. requiring a join every single product query)**
2. **VPD policies for operations that require DAC**
3. **Do we include the policies SQL Statement in the project, or just statements to describe the policy? (with the 10 page limit)**

Timeline:
Table creation by 27 Sep
27 Sep - 4 October is Generate data, everything else
Decide whether to do 9/10/11 tables on 4 October
11 October onwards 100% on report
8 October talk about research paper for initial questions
15 October meet for project 2 and research paper

To-do list:

1. **Table Creation, Entity Relationship Diagram, Database account creation (Customer seller admin) (XianHao)**
2. **Policy 1-10 excl 7/8 Policy 11-14 (Edward, Hong Wei, Ying Hui)**
3. **Data Generation (Lawrence)**
4. **Tell the story of our database (No kick)**

Tables:

1. Users:
   a. Fields: **userID**, username, firstName, lastName, email, password, userType, account_standing (customer, seller, admin)
2. Products:
   a. Fields: **productID**, sellerID, productName, description, price, stockQuantity, category
3. Orders:
   a. Fields: **orderID**, userID, datePlaced, status
4. OrderDetails:
   a. Fields: **orderDetailID**, orderID, productID, quantity, priceAtPurchase
5. Reviews:
   a. Fields: **reviewID**, productID, userID, rating, content
6. Payments:
   a. Fields: **paymentID**, orderID, paymentMethod, amount, paymentDate
7. Shipping:
   a. Fields: **shippingID**, orderID, address, shippingDate, deliveryDate, status, trackingID
8. Support cases:

a. Fields: **supportID**, orderID, ~~*reviewID*~~, status, comments, assignedTo, lastUpdated
9. Address book:
10. Stored payment methods:
11. Discount coupon for any product (with a time limit, and also with a limited number of coupons)

Please color code your stuff to track

Done and tested, Done not tested or got probs, Not Done,

Policy done by:

1	2	3	4/1a	5	6	7	8	9	10	11	12	13	14	~~16~~	~~17~~	18	~~19~~	20
Hong Wei	ed	YH	changed	ed	ed	Law	Law	Hong Wei	YH	XH	YH	XH	Hong Wei	Law/YH Covered in 3	Duplicate with 3	Law	~~Law~~	Law

Policies:
1. Customers can view only their orders.
2. Sellers can only manage their products.
3. Customers can only update their own passwords and other information
   a. Seller can only update their own passwords and other information
   b. Admin can only update their own passwords and other information
   c. Customers can only view their own passwords and other information
   d. Seller can only view their own passwords and other information
   e. Admin can view the whole table
4. Sellers can view only their orders containing their products
5. Customer can only edit their own reviews
6. Customers can only review after delivery date (implication is that already bought)
7. Only Admin can update cases
8. Only Admin can do delete
   a. Should we use VPD for operations that can be done with DAC?
9. Customers can't view the payment information of other customers.
10. Sellers cannot see paymentMethod but can see other things. (Column level VPD lecture 2.1 slide 25)
11. Sellers can only view order and shipping details related to their own products.
12. Only Admin can change user roles (e.g., promote a customer to a seller or vice-versa).

13. Sellers can only access financial reports related to their own sales. (Aggregation VPD), how much earn total or how many they sell this month
14. ~~Customers can only initiate returns within a specified return period after the delivery date.~~
15. ~~Only Admin can ban or suspend user accounts. (Don't do first)~~
16. ~~Allow sellers and customers to see their own account details (User table)~~
    Note: Policy 16 seems has been covered by above policy 3 with
    `statement_types => 'SELECT, UPDATE, DELETE'`

17. ~~Customers can only update their own passwords and other information. (User table)~~
    Note: Policy 17 is duplicate with above policy 3
18. Sellers might be allowed to view aggregated ratings (like average rating, number of reviews) for their own products but not individual review details (Aggregation VPD for product review)
19. ~~Customers can see the number of their orders in each status (e.g., "PENDING", "SHIPPED") without accessing the detailed order information of other customers. (Aggregation VPD for shipping table)~~
20. Support staff might want to see the number of cases they have in each status (e.g., "OPEN", "IN_PROGRESS") without accessing the details of cases assigned to others. (Aggregation VPD for support cases)

Entity Linkages:

1. Users:
    a. userID is the primary key.
    b. userType determines the role (customer, seller, admin).
2. Products:
    a. productID is the primary key.
    b. sellerID is a foreign key linking to Users.userID, where userType = 'seller'.
3. Orders:
    a. orderID is the primary key.
    b. customerID is a foreign key linking to Users.userID, where userType = 'customer'.
4. OrderDetails:
    a. orderDetailID is the primary key.
    b. orderID is a foreign key linking to Orders.orderID.
    c. productID is a foreign key linking to Products.productID.
5. Reviews:
    a. reviewID is the primary key.
    b. productID is a foreign key linking to Products.productID.
    c. customerID is a foreign key linking to Users.userID, where userType = 'customer'.
6. Payments:
    a. paymentID is the primary key.
    b. orderID is a foreign key linking to Orders.orderID.

7. Shipping:
    a. shippingID is the primary key.
    b. orderID is a foreign key linking to Orders.orderID.

Relationships Summary:

- Users to Products: One-to-Many (One seller has many products).
- Users to Orders: One-to-Many (One customer places many orders).
- Orders to OrderDetails: One-to-Many (One order can have multiple products/items).
- Products to OrderDetails: One-to-Many (One product can be in many orders, but each order detail references one product).
- Products to Reviews: One-to-Many (One product can have many reviews).
- Users to Reviews: One-to-Many (One customer can leave many reviews).
- Orders to Payments: One-to-Many (Each order has one or more payment detail).
- Orders to Shipping: One-to-One (Each order has one shipping detail).

Customers can view only their orders.

```
BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema    => 'ECOMMERCE',
    object_name      => 'Orders',
    policy_name      => 'Customer_Order_Access',
    function_schema  => 'ECOMMERCE',
    policy_function  => 'Customer_Access_Function',
    statement_types  => 'SELECT',
    update_check     => TRUE,
    enable           => TRUE,
    policy_type      => DBMS_RLS.SHARED_STATIC
  );
END;
/
```

Sellers can only manage their products.

```
BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema    => 'ECOMMERCE',
    object_name      => 'Products',
    policy_name      => 'Seller_Product_Access',
    function_schema  => 'ECOMMERCE',
    policy_function  => 'Seller_Access_Function',
    statement_types  => 'SELECT, UPDATE',
    update_check     => TRUE,
    enable           => TRUE,
    policy_type      => DBMS_RLS.SHARED_STATIC
  );
END;
/
```

Inserting Data:
Before data is inserted, permission has to be granted to the ECOMMERCE user which contains the schema to allow insert to be done. This is done with: ALTER USER ECOMMERCE QUOTA UNLIMITED ON <TABLESPACE>;

Context Setup

```
CREATE OR REPLACE PACKAGE ecommerce.context_user_type_package AS
    PROCEDURE set_user_type_context;
    PROCEDURE set_user_id_context;
END;

CREATE OR REPLACE PACKAGE BODY ecommerce.context_user_type_package IS
    PROCEDURE set_user_type_context IS
        v_username  VARCHAR2(40);
        v_user_type VARCHAR2(12);
    BEGIN
        v_username := SYS_CONTEXT('USERENV','SESSION_USER');
        BEGIN
            SELECT user_type
            INTO   v_user_type
            FROM   ECOMMERCE.Users
            WHERE  UPPER(username) = v_username;

            DBMS_OUTPUT.PUT_LINE('set_user_type_context user_type: ' ||
v_user_type);

            DBMS_SESSION.set_context('ecommerce_user_type','user_type',
v_user_type);
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                DBMS_SESSION.set_context('ecommerce_user_type','user_type',
NULL);
        END;
    END set_user_type_context;
    PROCEDURE set_user_id_context IS
        v_username  VARCHAR2(40);
        v_user_id VARCHAR2(36);
    BEGIN
        v_username := SYS_CONTEXT('USERENV','SESSION_USER');
        BEGIN
            SELECT ID
            INTO   v_user_id
            FROM   ECOMMERCE.Users
            WHERE  UPPER(username) = v_username;

            DBMS_SESSION.set_context('ecommerce_user_type','user_id',
v_user_id);
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                DBMS_SESSION.set_context('ecommerce_user_type','user_id',
NULL);
        END;
    END set_user_id_context;
END context_user_type_package;

GRANT EXECUTE ON ecommerce.context_user_type_package TO PUBLIC;
CREATE PUBLIC SYNONYM context_user_type_package FOR
ecommerce.context_user_type_package;

CREATE OR REPLACE TRIGGER ecommerce.set_user_type_trigger AFTER LOGON ON
DATABASE
BEGIN
    ecommerce.context_user_type_package.set_user_type_context;
    ecommerce.context_user_type_package.set_user_id_context;
END;
```

Policy 1:

Policy Function
 	
Order table restricted to customer see their
	`CREATE OR REPLACE FUNCTION ecommerce.customer_order_policy (`
	`    p_schema IN VARCHAR2,`
	`    p_object IN VARCHAR2)`
	`RETURN VARCHAR2 AS`
	`    v_user_type VARCHAR2(12);`
	`    v_user_id VARCHAR2(36);`
	`BEGIN`
	`    -- Identify user`
	`v_user_id := SYS_CONTEXT('ecommerce_user_type', 'user_id');`

own order	`v_user_type := SYS_CONTEXT('ecommerce_user_type', 'user_type');`
	
	`    -- If user is cust or seller, return predicate to filter the orders by their ID`
	`    IF v_user_type IN ('CUSTOMER') THEN`
	`        RETURN 'USER_ID = ''' || v_user_id || '''';`
	`    ELSIF v_user_type = 'ADMIN' THEN`
	`        RETURN '1=1';`
	`    ELSE`
	`        RETURN '1=1';`
	`    END IF;`
	
	`EXCEPTION`
	`    WHEN NO_DATA_FOUND THEN`
	`        RETURN '1=1';  -- Return if user not found`
	`    WHEN OTHERS THEN`
	`        RETURN '1=1';`
	`END customer_order_policy;`

Enable Policy	`BEGIN`
	`    DBMS_RLS.ADD_POLICY (`
	`        object_schema   => 'ECOMMERCE',`
	`        object_name     => 'ORDERS',`
	`        policy_name     => 'CUSTOMER_ORDER_POLICY',`
	`        function_schema => 'ECOMMERCE',`
	`        policy_function => 'CUSTOMER_ORDER_POLICY',`
	`        statement_types => 'SELECT',`
	`        update_check    => TRUE`
	`    );`
	`END;`

Test VPD	Log into **CUSTOMER_1** `SELECT * FROM ECOMMERCE.ORDERS;` Expected result: Only can view UUID9 with username CUSTOMER_1 orders Log into **APP_ADMIN** SELECT * FROM ECOMMERCE.ORDERS; Expected result: Can view all orders

Policy 1a:

Policy	`CREATE OR REPLACE FUNCTION ecommerce.seller_order_policy (`

Function
 	
Order table restricted to seller see the orders of customers of only their products
	```
	    p_schema IN VARCHAR2,
	    p_object IN VARCHAR2)
	RETURN VARCHAR2 AS
	    v_user_type VARCHAR2(12);
	    v_user_id VARCHAR2(36);
	BEGIN
	    -- Identify user
	v_user_id := SYS_CONTEXT('ecommerce_user_type', 'user_id');
	v_user_type := SYS_CONTEXT('ecommerce_user_type', 'user_type');

	    -- If user is cust or seller, return predicate to filter the orders by their ID
	    IF v_user_type IN ('SELLER') THEN
	        RETURN 'ID IN (SELECT od.order_id FROM ecommerce.order_details od
	                             JOIN ecommerce.products p ON od.product_id = p.ID
	                             WHERE p.seller_id = ''' || v_user_id || ''')';
	    ELSIF v_user_type = 'ADMIN' THEN
	        RETURN '1=1';
	    ELSE
	        RETURN '1=1';
	    END IF;

	EXCEPTION
	    WHEN NO_DATA_FOUND THEN
	        RETURN '1=1';  -- Return if user not found
	    WHEN OTHERS THEN
	        RETURN '1=1';
	END seller_order_policy;
	```

Enable Policy	
	```
	BEGIN
	    DBMS_RLS.ADD_POLICY (
	        object_schema   => 'ECOMMERCE',
	        object_name     => 'ORDERS',
	        policy_name     => 'SELLER_ORDER_POLICY',
	        function_schema => 'ECOMMERCE',
	        policy_function => 'SELLER_ORDER_POLICY',
	        statement_types => 'SELECT, UPDATE',
	        update_check    => TRUE
	    );
	END;

	BEGIN
	    DBMS_RLS.ENABLE_POLICY(
	```

	```
	        object_schema    => 'ECOMMERCE',
	        object_name      => 'orders',
	        policy_name      => 'SELLER_ORDER_POLICY',
	        enable           => true
	    );
	END;
	```
Test VPD	Log into **CUSTOMER_1** `SELECT * FROM ECOMMERCE.ORDERS;` Expected result: Only can view UUID9 with username CUSTOMER_1 orders Log into **APP_ADMIN** SELECT * FROM ECOMMERCE.ORDERS; Expected result: Can view all orders

Policy 2 draft: Sellers can only manage their products (what about admin)

Policy Function	CREATE OR REPLACE FUNCTION update_product(
	        ECOMMERCE IN VARCHAR2, PRODUCTS IN VARCHAR2)
	RETURN VARCHAR2 AS condition VARCHAR2 (200);
	BEGIN
	   IF SYS_CONTEXT('ecommerce_user_type','user_type') = 'ADMIN' THEN
	        RETURN '1=1';
	        ELSIF SYS_CONTEXT('ecommerce_user_type','user_type') = 'SELLER' THEN
	        condition := 'SELLER_ID = SYS_CONTEXT(''ecommerce_user_type'',''user_id'')';
	        ELSE
	        RETURN '1=0';
	        END IF;
	        RETURN condition;
	END update_product;

Enable Policy	BEGIN
	        DBMS_RLS.ADD_POLICY(
	        object_schema => 'ECOMMERCE',
	        object_name => 'PRODUCTS',
	        policy_name => 'update_product_policy',
	        function_schema => 'ECOMMERCE',

	policy_function => 'update_product',
	statement_types => 'INSERT, UPDATE, DELETE',
	update_check => TRUE); --prevent others from also inserting into products table
	END;

Test VPD	BEGIN
	ecommerce.context_user_type_package.set_user_type_context;
	END;
	
	Log into **CUSTOMER_1**
	UPDATE ECOMMERCE.PRODUCTS
	SET STOCK_QUANTITY=100;
	Expected result: denied
	DELETE FROM PRODUCTS;
	Expected result: denied
	
	Log into **SELLER_1**
	UPDATE ECOMMERCE.PRODUCTS
	SET STOCK_QUANTITY=100;
	Expected result: only uuid10 products updated
	
	DELETE FROM PRODUCTS;
	Expected result: denied because cannot delete
	
	Log into **APP_ADMIN**
	UPDATE ECOMMERCE.PRODUCTS
	SET STOCK_QUANTITY=100;
	Expected result: can
	DELETE FROM PRODUCTS;
	Expected result: can

Policy 3 draft: Customers/Seller/Admin can only update their own passwords and other information
Customers/Seller/ can only view their own passwords and other information
Admin can view the whole users table

Policy Function	`CREATE OR REPLACE FUNCTION ecommerce.customer_info_update (`
	`      schema_var IN VARCHAR2,`
	`    table_var IN VARCHAR2)`
	`RETURN VARCHAR2 AS condition VARCHAR2(200);`
	`BEGIN`
	`    condition := 'UPPER(username) = SYS_CONTEXT(''USERENV'',''SESSION_USER'')';`

```
    DBMS_OUTPUT.PUT_LINE('customer_info_update return condition: ' 
|| condition);
    RETURN condition;
END customer_info_update;
```

Enable Policy

```
BEGIN
    DBMS_RLS.DROP_POLICY(
      object_schema    => 'ECOMMERCE',
      object_name      => 'Users',
      policy_name      => 'customer_info_update_policy'
       );
    DBMS_RLS.ADD_POLICY(
        object_schema   => 'ECOMMERCE',
        object_name     => 'Users',
        policy_name     => 'customer_info_update_policy',
        function_schema => 'ECOMMERCE',
        policy_function => 'customer_info_update',
        statement_types => 'UPDATE, DELETE',
        sec_relevant_cols => 'username, first_name, last_name, 
email, password, account_standing',
        update_check    => true
    );
END;
```

Test VPD

```
Log into CUSTOMER_1

SELECT * FROM ECOMMERCE.Users;
Expected result: can only view information of current login user

UPDATE ECOMMERCE.Users SET last_name = 'Update';
Expected result: update successfully

UPDATE ECOMMERCE.Users SET last_name = 'Update' WHERE username = 
'seller_1';
Expected result: denied

UPDATE ECOMMERCE.USERS SET password = STANDARD_HASH('pAssw0rd', 
'SHA256');
Expected result: update successfully

UPDATE ECOMMERCE.Users SET user_type = 'SELLER';
Expected result: update fail due to policy 12
```

	Same behaviour as above if Log into **Seller_1** if Log into **APP_ADMIN** `SELECT * FROM ECOMMERCE.Users;` Expected result: can view the whole table Update behaviour should be same as customer and seller

Policy 4 draft: Sellers can view only their orders containing their products

Policy Function	
	```
	CREATE OR REPLACE FUNCTION ecommerce.customer_order_policy (
	    p_schema IN VARCHAR2,
	    p_object IN VARCHAR2)
	RETURN VARCHAR2 AS
	    v_user_type VARCHAR2(12);
	    v_user_id VARCHAR2(36);
	BEGIN
	    -- Identify user
	v_user_id := SYS_CONTEXT('ecommerce_user_type', 'user_id');
	v_user_type := SYS_CONTEXT('ecommerce_user_type', 'user_type');

	    -- If user is cust or seller, return predicate to filter the
	orders by their ID
	    IF v_user_type IN ('SELLER') THEN
	        RETURN 'USER_ID = ''' || v_user_id || '''';
	    ELSIF v_user_type = 'ADMIN' THEN
	        RETURN '1=1';
	    ELSE
	        RETURN '1=1';
	    END IF;

	EXCEPTION
	    WHEN NO_DATA_FOUND THEN
	        RETURN '1=1';  -- Return if user not found
	    WHEN OTHERS THEN
	        RETURN '1=1';
	END customer_order_policy;
	```

Enable Policy	
	```
	BEGIN
	    DBMS_RLS.ADD_POLICY (
	        object_schema   => 'ECOMMERCE',
	        object_name     => 'ORDERS',
	        policy_name     => 'CUSTOMER_ORDER_POLICY',
	        function_schema => 'ECOMMERCE',
	```

	`        policy_function => 'CUSTOMER_ORDER_POLICY',`
	`        statement_types => 'SELECT',`
	`        update_check    => TRUE`
	`    );`
	`END;`

Test VPD	Log into **CUSTOMER_1** `SELECT * FROM ECOMMERCE.ORDERS;` Expected result: Only can view UUID9 with username CUSTOMER_1 orders Log into **APP_ADMIN** SELECT * FROM ECOMMERCE.ORDERS; Expected result: Can view all orders

Policy 5 draft: Customer can only edit their own reviews, admin can edit all reviews

Policy Function	CREATE OR REPLACE FUNCTION update_review(
	 ECOMMERCE IN VARCHAR2, REVIEWS IN VARCHAR2)
	RETURN VARCHAR2 AS condition VARCHAR2 (200);
	BEGIN
	 IF SYS_CONTEXT('ecommerce_user_type','user_type') = 'ADMIN' THEN
	  RETURN '1=1';
	 ELSE
	  condition := 'SYS_CONTEXT(''ecommerce_user_type'',''user_id'') = USER_ID';
	  END IF;
	  RETURN condition;
	END update_review;

Enable Policy	BEGIN DBMS_RLS.ADD_POLICY(object_schema => 'ECOMMERCE', object_name => 'REVIEWS', policy_name => 'update_review_policy', function_schema => 'ECOMMERCE', policy_function => 'update_review', statement_types => 'UPDATE'); END;

Test VPD	BEGIN
	  ecommerce.context_user_type_package.set_user_type_context;
	END;
	
	Log into **CUSTOMER_1**
	UPDATE ECOMMERCE.REVIEWS
	SET RATING=1;
	Expected result: denied not own rating
	UPDATE ECOMMERCE.REVIEWS
	SET RATING=1 WHERE ID=6;
	Expected result: denied not own rating
	
	Log into **SELLER_1**
	UPDATE ECOMMERCE.REVIEWS
	SET RATING=1;
	Expected result: denied not own rating
	UPDATE ECOMMERCE.REVIEWS
	SET RATING=1 WHERE ID=6;
	Expected result: denied not own rating
	
	Log into **APP_ADMIN**
	UPDATE ECOMMERCE.REVIEWS
	SET RATING=1;
	Expected result: allowed
	UPDATE ECOMMERCE.REVIEWS
	SET CONTENT=” WHERE ID=10;
	Expected result: allowed

Policy 6 draft: Customers can only review after delivery date

Policy Function	CREATE OR REPLACE FUNCTION review_after_delivery_check(
	          ECOMMERCE IN VARCHAR2, REVIEWS IN VARCHAR2)
	RETURN VARCHAR2 AS condition VARCHAR2 (200);
	BEGIN
	    condition:='PRODUCT_ID IN
	    (SELECT o.product_id FROM orderandproductview o
	           WHERE ''COMPLETED'' = o.status)';
	          RETURN condition;
	END review_after_delivery_check;

Enable Policy	BEGIN
	          DBMS_RLS.ADD_POLICY(
	          object_schema => 'ECOMMERCE',
	          object_name => 'REVIEWS',
	          policy_name => 'review_after_delivery_check_policy',

	function_schema => 'ECOMMERCE',
	policy_function => 'review_after_delivery_check',
	statement_types => 'INSERT',
	update_check => TRUE);
	END;

Test VPD	Log into **CUSTOMER_1**
	INSERT into ECOMMERCE.REVIEWS
	values (11, 'UUID9', 8, 5, 'good');
	Expected result: denied, no such product tied to user order
	
	INSERT into ECOMMERCE.REVIEWS
	values (11, ‘UUID9’, 9, 5, ‘good’);
	Expected result: accepted
	
	INSERT into ECOMMERCE.REVIEWS
	values (11, ‘UUID9’, 22, 5, ‘good’);
	Expected result: denied, order not completed
	
	Log into **SELLER_1**
	INSERT into ECOMMERCE.REVIEWS (id, user_id, product_id, rating, content)
	values (11, ‘UUID10’, 9, 5, ‘good’);
	Expected result: denied not own rating
	
	Log into **APP_ADMIN**
	INSERT into ECOMMERCE.REVIEWS (id, user_id, product_id, rating, content)
	values (11, ‘UUID10’, 9, 5, ‘good’);
	Expected result: denied because admin doesnt buy

Policy 7&8 Draft:

Policy Function	CREATE OR REPLACE FUNCTION
	ECOMMERCE.support_cases_update_policy (
	  p_schema IN VARCHAR2,
	  p_object IN VARCHAR2)
	RETURN VARCHAR2 AS
	  condition VARCHAR2(200);
	BEGIN
	  IF SYS_CONTEXT('ecommerce_user_type', 'user_type') = 'ADMIN' THEN
	    RETURN '1=1'; -- Admin has full access
	  ELSE
	    -- Non-admin users can't update or delete records
	    condition := '1=0';
	  END IF;

	` DBMS_OUTPUT.PUT_LINE('support_cases_update_policy return condition: ' || condition);`
	` RETURN condition;`
	`END support_cases_update_policy;`
	`/`

Enable Policy	`BEGIN`
	` DBMS_RLS.ADD_POLICY(`
	`  object_schema => 'ECOMMERCE',`
	`  object_name => 'support_cases',`
	`  policy_name => 'support_cases_update_policy',`
	`  function_schema => 'ECOMMERCE',`
	`  policy_function => 'support_cases_update_policy',`
	`  statement_types => 'UPDATE, DELETE',`
	`  update_check => true`
	` );`
	`END;`
	`/`

Test VPD	**Login as app_admin**
	
	INSERT INTO ECOMMERCE.support_cases (id, order_id, status, comments, assigned_user) VALUES ('SUPPORT_UUID12', 'ORDER_UUID11', 'OPEN', 'Issue with payment processing', 'UUID9');
	
	SELECT * FROM ECOMMERCE.support_cases
	
	DELETE FROM ECOMMERCE.support_cases WHERE id = 'SUPPORT_UUID12';

Policy 9 draft: Customers can't view the payment information of other customers. For sellers, only view payments of orders on their products

Policy Function	`CREATE OR REPLACE FUNCTION ecommerce.payments_policy (`
	`    p_schema IN VARCHAR2,`
	`    p_object IN VARCHAR2)`
	`RETURN VARCHAR2 AS`
	`    v_user_type VARCHAR2(12);`
	`    v_user_id VARCHAR2(36);`
	`BEGIN`

```
    -- Identify the user type and ID based on the username
    v_user_id := SYS_CONTEXT('ecommerce_user_type', 'user_id');
    v_user_type := SYS_CONTEXT('ecommerce_user_type',
'user_type');
    -- If the user is a customer, return a predicate to filter
the payments by their ID
    IF v_user_type = 'CUSTOMER' THEN
        RETURN 'ORDER_ID IN (SELECT ID FROM ecommerce.orders
WHERE USER_ID = ''' || v_user_id || ''')';
    ELSIF v_user_type = 'SELLER' THEN
        -- For sellers, return a predicate to filter payments of
orders on their products
        RETURN 'ORDER_ID IN (SELECT o.ID FROM ecommerce.orders o
                             JOIN ecommerce.order_details od ON
o.ID = od.order_id
                             JOIN ecommerce.products p ON
od.product_id = p.ID
                             WHERE p.seller_id = ''' ||
v_user_id || ''')';
    ELSE
        -- For now, we'll return no restrictions for other
users.
        RETURN '1=1';
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN '1=0';  -- Return a false predicate if the user
is not found, denying access
    WHEN OTHERS THEN
        RETURN '1=0';  -- Deny access in case of other
exceptions
END payments_policy;
/
```

Enable Policy

```
BEGIN
    DBMS_RLS.ADD_POLICY (
        object_schema   => 'ECOMMERCE',
        object_name     => 'PAYMENTS',
        policy_name     => 'PAYMENTS_POLICY',
        function_schema => 'ECOMMERCE',
        policy_function => 'PAYMENTS_POLICY',
        statement_types => 'SELECT'
    );
END;
/
```

	```
	BEGIN
	    DBMS_RLS.ENABLE_POLICY(
	        object_schema    => 'ECOMMERCE',
	        object_name      => 'PAYMENTS',
	        policy_name      => 'PAYMENTS_POLICY',
	        enable           => True
	    );
	END;
	```
Test VPD	Log into **CUSTOMER_1** `SELECT * FROM ECOMMERCE.PAYMENTS;` Expected result: Only can view their own payments Log into **SELLER_1** SELECT * FROM ECOMMERCE.PAYMENTS; Expected result: Can view all payments to their products Log into **APP_ADMIN** SELECT * FROM ECOMMERCE.PAYMENTS; Expected result: Can view all payments

Policy 10 draft: Sellers cannot see paymentMethod but can see other things.

Policy Function	
	```
	CREATE CONTEXT ecommerce_user_type USING
	ecommerce.context_user_type_package;

	CREATE OR REPLACE PACKAGE ecommerce.context_user_type_package AS
	  PROCEDURE set_user_type_context;
	END;
	/

	CREATE OR REPLACE PACKAGE BODY
	ecommerce.context_user_type_package IS
	  PROCEDURE set_user_type_context IS
	    v_username  VARCHAR2(40);
	    v_user_type VARCHAR2(12);
	  BEGIN
	    v_username := SYS_CONTEXT('USERENV','SESSION_USER');
	    BEGIN
	```

	```
	      SELECT user_type
	      INTO   v_user_type
	      FROM   ECOMMERCE.Users
	      WHERE  UPPER(username) = v_username;

	      DBMS_OUTPUT.PUT_LINE('set_user_type_context user_type: ' ||
	v_user_type);

	      DBMS_SESSION.set_context('ecommerce_user_type','user_type',
	v_user_type);
	    EXCEPTION
	      WHEN NO_DATA_FOUND THEN

	DBMS_SESSION.set_context('ecommerce_user_type','user_type',
	NULL);
	    END;
	  END set_user_type_context;
	END context_user_type_package;
	/

	GRANT EXECUTE ON ecommerce.context_user_type_package TO PUBLIC;
	CREATE PUBLIC SYNONYM context_user_type_package FOR
	ecommerce.context_user_type_package;

	--CREATE OR REPLACE TRIGGER ecommerce.set_security_context AFTER
	LOGON ON DATABASE
	BEGIN
	  ecommerce.context_user_type_package.set_user_type_context;
	END;
	/
	```

Enable Policy	
	```
	CREATE OR REPLACE FUNCTION ecommerce.seller_view_payment_info(
	      p_schema IN VARCHAR2, p_object IN VARCHAR2)
	RETURN VARCHAR2 AS
	BEGIN
	    IF (SYS_CONTEXT('ecommerce_user_type','user_type') =
	'SELLER') THEN
	        RETURN '1=0';
	    ELSE
	        RETURN '1=1';
	    END IF;
	END seller_view_payment_info;
	```

	```
	BEGIN
	  ecommerce.context_user_type_package.set_user_type_context;
	END;
	```
Test VPD	Log into **SELLER_1** `SELECT * FROM ecommerce.payments;` `Expected Result:` Sellers cannot see paymentMethod which is NULL but can see other things Log into **APP_ADMIN** `SELECT * FROM ecommerce.payments;` `Expected Result:` Admin can see paymentMethod Log into **CUSTOMER_1** `SELECT * FROM ecommerce.payments;` `Expected Result:` Customer can see paymentMethod

Policy 11/13 Draft:

Policy Function	
	```
	CREATE OR REPLACE FUNCTION check_order_details_access(
	    schema_name IN VARCHAR2,
	    table_name IN VARCHAR2
	) RETURN VARCHAR2
	AS
	    v_username  VARCHAR2(40);
	    v_user_type VARCHAR2(12);
	    v_user_id   VARCHAR2(36);
	    v_condition VARCHAR2(4000);
	BEGIN
	    v_username := LOWER(SYS_CONTEXT('USERENV',
	'SESSION_USER'));
	    v_user_type := LOWER(SYS_CONTEXT('ecommerce_user_type',
	```

```
'user_type'));
    v_user_id := SYS_CONTEXT('ecommerce_user_type',
'user_id');
    IF (v_username = 'ecommerce' OR v_username = 'system' OR
v_user_type = 'admin') THEN
        v_condition := '1=1';
    ELSIF (v_user_type = 'customer') THEN
        v_condition := 'EXISTS (
                            SELECT 1 FROM orders o
                            WHERE order_id = o.id
                            AND o.user_id = ''' || v_user_id
|| '''
                        )';
    ELSE
        v_condition := 'EXISTS (
                            SELECT 1 FROM products p
                            WHERE p.seller_id = ''' ||
v_user_id || '''
                            AND product_id = p.id
                        )';
    END IF;
    RETURN v_condition;
END;

CREATE OR REPLACE FUNCTION check_shipping_details_access (
    schema_name IN VARCHAR2,
    table_name IN VARCHAR2
) RETURN VARCHAR2
AS
    v_username VARCHAR2(40);
    v_user_type VARCHAR2(12);
    v_user_id VARCHAR2(36);
    v_condition VARCHAR2(4000);
BEGIN
    v_username :=
LOWER(SYS_CONTEXT('USERENV','SESSION_USER'));
    v_user_type :=
LOWER(SYS_CONTEXT('ecommerce_user_type','user_type'));
    v_user_id := SYS_CONTEXT('ecommerce_user_type','user_id');
    IF (v_username = 'ecommerce' OR v_username = 'system' OR
v_user_type = 'admin') THEN
        v_condition := '1=1';
    ELSIF (v_user_type = 'customer') THEN
        v_condition := 'EXISTS (
                            SELECT 1 FROM orders o
                            WHERE order_id = o.id
                            AND o.user_id = '''|| v_user_id
||'''
                        )';
```

	```
	    ELSE
	        v_condition := 'order_id IN (
	                                SELECT o.id FROM products p, 
	orders o, order_details od
	                                WHERE p.seller_id = '''||
	v_user_id ||'''
	                                AND od.product_id = p.id
	                                AND od.order_id = o.id
	                        )';
	    END IF;
	    RETURN v_condition;
	END;
	```

Enable Policy	
	```
	BEGIN
	    DBMS_RLS.ADD_POLICY(
	            object_schema => 'ECOMMERCE',
	            object_name => 'order_details',
	            policy_name => 'order_details_access_policy',
	            function_schema => 'ECOMMERCE',
	            policy_function => 'check_order_details_access',
	            statement_types => 'SELECT',
	            update_check => false,
	            enable => true
	        );
	END;

	BEGIN
	    DBMS_RLS.ADD_POLICY(
	            object_schema    => 'ECOMMERCE',
	            object_name      => 'shipping_details',
	            policy_name      =>
	'shipping_details_access_policy',
	            function_schema  => 'ECOMMERCE',
	            policy_function  =>
	'check_shipping_details_access',
	            statement_types  => 'SELECT',
	            update_check     => false,
	            enable           => true
	        );
	END;
	```

Test VPD	

Policy 12 draft:

Policy Function

```
CREATE OR REPLACE FUNCTION ecommerce.role_update(
      p_schema IN VARCHAR2, p_object IN VARCHAR2)
RETURN VARCHAR2 AS
BEGIN
    IF SYS_CONTEXT('ecommerce_user_type','user_type') = 'ADMIN'
THEN
        RETURN '1=1';
    ELSE
        RETURN '1=0';
    END IF;
END role_update;
```

Enable Policy

```
BEGIN
   DBMS_RLS.DROP_POLICY(
         object_schema    => 'ECOMMERCE',
         object_name      => 'Users',
         policy_name      => 'role_update_policy'
          );
   DBMS_RLS.ADD_POLICY(
      object_schema   => 'ECOMMERCE',
      object_name     => 'Users',
         policy_name     => 'role_update_policy',
         function_schema => 'ECOMMERCE',
         policy_function => 'role_update',
         statement_types => 'UPDATE',
      sec_relevant_cols => 'id, user_type',
      update_check => TRUE
          );
END;
```

Test VPD

Log into **APP_ADMIN**

UPDATE ECOMMERCE.users SET user_type = 'CUSTOMER' WHERE id = 'UUID2';
Expected Result: Admin can change user roles to 'SELLER'

Log into **SELLER_1**

UPDATE ECOMMERCE.Users SET user_type = 'CUSTOMER';
Expected result: update fail due to policy 12

Expected Result: Seller cannot change user roles to 'CUSTOMER'

	Log into **CUSTOMER_1** UPDATE ECOMMERCE.Users SET user_type = 'SELLER'; Expected Result: Customer cannot change user roles to 'SELLER'

Policy 14: Seller and admin can allow customer to refund within specified period of 7 days

<table>
<tr><td>Policy Function</td><td>

Need to insert some data into shipping and see how to update both shipping status and order status to return/refund

```
CREATE OR REPLACE FUNCTION ecommerce.refund_policy (
    p_schema IN VARCHAR2,
    p_object IN VARCHAR2)
RETURN VARCHAR2 AS
    v_user_type VARCHAR2(12);
    v_user_id VARCHAR2(36);
    v_refund_period NUMBER := 7;  -- Default 7
BEGIN
    v_user_id := SYS_CONTEXT('ecommerce_user_type', 'user_id');
    v_user_type := SYS_CONTEXT('ecommerce_user_type',
'user_type');
    -- If user is cust, return predicate to check refund
eligibility
    IF v_user_type = 'CUSTOMER' THEN
        RETURN 'USER_ID = ''' || v_user_id || ''' AND
                status = ''COMPLETED'' AND
                delivery_date + ' || v_refund_period || ' >=
SYSDATE';
    ELSE
        RETURN NULL;
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN '1=0';
    WHEN OTHERS THEN
        RETURN '1=0';
```

</td></tr>
</table>

	`END refund_policy;`
	`/`

Enable Policy	
	```
	BEGIN
	    DBMS_RLS.ADD_POLICY (
	        object_schema   => 'ECOMMERCE',
	        object_name     => 'ORDERS',
	        policy_name     => 'REFUND_POLICY',
	        function_schema => 'ECOMMERCE',
	        policy_function => 'REFUND_POLICY',
	        statement_types => 'UPDATE'
	    );
	END;
	/

	BEGIN
	    DBMS_RLS.ENABLE_POLICY(
	        object_schema    => 'ECOMMERCE',
	        object_name      => 'ORDERS',
	        policy_name      => 'REFUND_POLICY',
	        enable           => true
	    );
	END;
	```

Test VPD	-- Insert test orders
	-- Order delivered 7 days ago
	INSERT INTO ecommerce.orders (id, user_id, status, delivery_date)
	VALUES ('ORDER_UUID12', 'UUID9', 'COMPLETED', SYSDATE - 7);
	
	-- Order delivered 8 days ago
	INSERT INTO ecommerce.orders (id, user_id, status, delivery_date)
	VALUES ('ORDER_UUID13', 'UUID9', 'COMPLETED', SYSDATE - 8);
	
	-- Order delivered recently
	INSERT INTO ecommerce.orders (id, user_id, status, delivery_date)
	VALUES ('ORDER_UUID14', 'UUID9', 'COMPLETED', SYSDATE - 1);
	
	—---

	UPDATE ecommerce.orders
	SET status = 'REFUNDED'
	WHERE id = 'ORDER_UUID12'; -- Should succeed
	
	UPDATE ecommerce.orders
	SET status = 'REFUNDED'
	WHERE id = 'ORDER_UUID13'; -- Should fail
	
	UPDATE ecommerce.orders
	SET status = 'REFUNDED'
	WHERE id = 'ORDER_UUID14'; -- Should succeed

Policy 16 Draft:

Policy Function	CREATE OR REPLACE FUNCTION ecommerce.customer_info_view (
	          p_schema IN VARCHAR2, p_object IN VARCHAR2)
	RETURN VARCHAR2 AS condition VARCHAR2(200);
	BEGIN
	   IF SYS_CONTEXT('ecommerce_user_type','user_type') = 'ADMIN' THEN
	      RETURN '1=1';
	   ELSE
	      condition := 'UPPER(username) =
	SYS_CONTEXT(''USERENV'',''SESSION_USER'')';
	   END IF;
	   DBMS_OUTPUT.PUT_LINE('customer_info_update return condition: ' ||
	condition);
	   RETURN condition;
	END customer_info_view;
	
	/

Enable Policy	BEGIN
	   DBMS_RLS.ADD_POLICY(
	      object_schema => 'ECOMMERCE',
	      object_name => 'Users',
	      policy_name => 'customer_info_view_policy',
	      function_schema => 'ECOMMERCE',
	      policy_function => 'customer_info_view',
	      statement_types => 'SELECT'
	   );
	END;

	/

Test VPD	Context set to self
	
	UPDATE ECOMMERCE.users SET email = 'new_email@example.com' WHERE id = 'UUID1'; (Pass for this)
	
	UPDATE ECOMMERCE.users SET email = 'hacked_email@example.com' WHERE id = 'UUID2'; (Fail for this)

Policy 18 Draft:

Policy Function	
	```
	CREATE OR REPLACE FUNCTION ECOMMERCE.seller_review_policy (
	   p_schema IN VARCHAR2,
	   p_object IN VARCHAR2)
	RETURN VARCHAR2 AS
	   condition VARCHAR2(200);
	BEGIN
	   IF (SYS_CONTEXT('ecommerce_user_type','user_type') = 'SELLER') THEN
	      condition := 'product_id IN (SELECT id FROM ECOMMERCE.products
	WHERE seller_id = SYS_CONTEXT(''ECOMMERCE_USER_TYPE'',
	''USER_ID''))';
	   ELSE
	      condition := '1=1';
	   END IF;
	   RETURN condition;
	END seller_review_policy;
	/
	```

Enable Policy	
	```
	BEGIN
	   DBMS_RLS.ADD_POLICY(
	      object_schema   => 'ECOMMERCE',
	      object_name     => 'reviews',
	      policy_name     => 'seller_review_policy',
	      function_schema => 'ECOMMERCE',
	      policy_function => 'seller_review_policy',
	      statement_types => 'SELECT',
	      update_check    => true
	   );
	END;
	/
	```

Test VPD	Log into SELLER_1
	
	SELECT product_id, AVG(rating) as average_rating, COUNT(*) as review_count
	FROM ECOMMERCE.reviews
	GROUP BY product_id;
	
	Expected Result: Seller can view aggregated ratings and count of reviews for their own products only, but not individual review details.

Policy 20 Draft:

Policy Function	
	```
	CREATE OR REPLACE FUNCTION ECOMMERCE.support_staff_case_policy (
	    p_schema IN VARCHAR2,
	    p_object IN VARCHAR2)
	RETURN VARCHAR2 AS
	    condition VARCHAR2(200);
	BEGIN
	    condition := 'assigned_user = SYS_CONTEXT(''ecommerce_user_type'', ''user_id'') OR SYS_CONTEXT(''ecommerce_user_type'', ''user_type'') = ''ADMIN''';
	    RETURN condition;
	END support_staff_case_policy;
	/
	```

Enable Policy	
	```
	BEGIN
	    DBMS_RLS.ADD_POLICY(
	        object_schema   => 'ECOMMERCE',
	        object_name     => 'support_cases',
	        policy_name     => 'support_staff_case_policy',
	        function_schema => 'ECOMMERCE',
	        policy_function => 'support_staff_case_policy',
	        statement_types => 'SELECT',
	        update_check    => true
	    );
	END;
	/
	```

Test VPD	Log into APP_ADMIN
	
	SELECT status, COUNT(*) as cases_count
	FROM ECOMMERCE.support_cases

	GROUP BY status; Expected Result: Support staff can view the number of cases they are assigned based on each status but cannot view details of cases assigned to other staff members.