Oracle Virtual Private Database for Property & Casualty Insurance - Auto Claims

Ajay Batra A0298397B
Liu Yun A0268280Y
Elamparithi Kannan A0298319M
Pavin Rex Baskar A0298567A
Indrayani Wanjari A0298405W
Rishikesh Kadiyala A0298806L

***Abstract*—This report details the design and implementation of an Oracle Virtual Private Database (VPD) for our auto claims application in the domain of Property & Casualty Insurance. The VPD ensures that sensitive data access to related tables is restricted based on roles through different steps of the business process to enhance database security.**

***Index Terms*—Oracle VPD, Database Security, Property & Casualty Insurance - Auto Claim, Fine Grained Access Control(FGAC), Security Policies, ER Model.**

I Introduction

Virtual Private Database (VPD) is a feature in Oracle that enables row and column-level security, allowing us to restrict data access based policies that are mostly related to roles. In this report, we explore the application of VPD in our auto claims insurance application context to ensure that different roles (e.g., policy holder, adjuster, auditor, investigator, manager, vendor) have limited access to certain database tables, rows, and columns adhering to security policies.

II Application Overview

P&C Insurance domain is an insurance domain which provides protection against potential risks associated with ownership and liability.

- It encompasses a broad range of insurance products that cover both damage to property and legal responsibility for injuries or damage caused to others.
- Unlike life or health insurance, P&C insurance focuses on safeguarding assets and mitigating liability risks, both for individuals and businesses.

II-A Types of P&C Insurance

- Homeowner's insurance
- Auto insurance
- Workers Compensation
- Marine insurance

For our project purpose, we will be considering an auto insurance (vehicle insurance) application implementation. Regarding auto insurance, (or any other P&C insurance types), there are 2 main processes involved:

1. Policy creation and maintenance:

- A user/client takes his/her auto insurance with the company and the company creates an auto insurance policy for the client, based on various factors such as premium plan chosen, coverage and sub coverage added for the plan, validity etc.,
- The insurance company is responsible for notifying the policy holder on renewals and claim payments.

2. Claims:

- Process of filing and settling claims when policyholders experience losses covered under their insurance policies.
- Claims are typically filed after events such as accidents, property damage, or liability-related incidents. (We here focus on only accidents, vehicle theft, vandalism and auto insurance liability (where the policy holder is at fault))
- Efficient claims handling is crucial for both insurers and policyholders, as it determines the financial compensation policyholders receive and plays a critical role in customer satisfaction and retention.
- The process of handling P&C insurance claims can be complex, involving claim initiation, investigation, assessment of damage or liability, settlement negotiation, and payment.

II-B Coverages available for auto policy

1. Collision coverage:

- pays for the repair or replacement of the insured's vehicle after an accident involving another car or a stationary object (like a tree or guardrail), regardless of who is at fault.
- In short words, covers expenses after colliding with any car/object.

2. Comprehensive coverage:

- Theft
- Vandalism
- Natural disasters (floods, hurricanes, earthquakes)
- Fire
- Falling objects (e.g., tree branches)
- Animal collisions (e.g., hitting a deer)
- Glass damage (e.g., broken windshields)

3. Uninsured Motorist coverage: Protects the policyholder if they are hit by a driver who does not have insurance. It typically covers medical expenses, lost wages, and other injury-related costs.

4. Medical Payments coverage (MedPay): For the health costs of client and the passengers of the client's vehicle, injured in an accident, regardless of whose fault.

5. Liability coverage:

- Covers medical expenses, lost wages, pain and suffering, and legal fees when the insured driver causes injury or death to others in an accident.

- Covers the cost of repairing or replacing another person's property (typically their vehicle) that is damaged in an accident caused by the insured driver.

II-C Key steps involved in auto claim process

1. Claim filing:

1.a. Initial Notification: The policyholder contacts the insurer to report the loss or incident. This can be done through various channels, including phone, online portals, or mobile apps. The policyholder provides essential details, including the nature of the loss, the date, location, and any other relevant information.

1.b. Documentation: The policyholder is usually required to provide documentation to support the claim. This might include photos of the damage, repair estimates, police reports (for theft or accidents) and medical records (for injury claims).

2. Claim acknowledgement: The insurer/insurance company acknowledges receipt of the claim and assigns a claim adjuster to investigate. A claim number is typically generated, and the policyholder is informed about the next steps in the process.

3. Claim investigation:

3.a. Assessing the Damage and validating the claim: The adjuster assigns an auditor to check the authenticity of the claim. The auditor may interview the parties involved in an accident. He/she will investigate to determine who is legally responsible for the damage or injury. This may involve reviewing police reports, witness statements, and other evidence.

3.b. Checking Policy Coverage: Once the validation by the auditor is done, the adjuster notifies the policy investigator to review the insurance policy and to verify whether the policy is in place or expired.

3.c. Estimating Costs: The adjuster estimates the repair or replacement costs for vehicle damage or calculates medical and legal expenses for liability claims and verifies it with the deductible limits available for the claimant/policy holder's auto policy. The adjuster also notifies the manager to assign vendors for any additional services required:

- Towing and labor (Roadside assistance)
- Salvage vendors (Scrap dealers)
  - When the repair cost exceeds the current value of the car, the insurance company declares it as a total loss and compensates the current value of the car to the client. The damaged car is given out to salvage vendors, from whom the insurance company will earn by selling the damaged vehicle.

4. Claim settlement and payment:

- The adjuster sends the claim validation and investigation results to the manager. He/she is responsible for processing full payments (if the limits are within range) or partial payments. The manager on behalf of our company issues a payment for the agreed-upon amount. Payment may be made to the policyholder directly or to the representative of the policyholder (on policy holder's death during the accident).
- After the payment, the claim is closed, and the record is maintained by our insurance company.

Figure 1 presents an overview of the property and casualty insurance - auto claim process, with the detailed steps varying according to different roles, as described in the following sections.

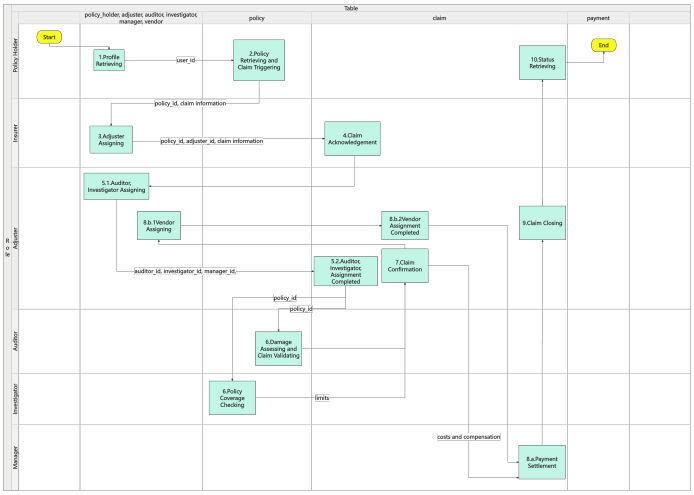

Fig. 1: Application Architecture Overview

II-D Roles involved in auto claim process

1) **Policy holder**
   a) The person who holds an auto insurance policy in our company and who is capable of raising claims when a loss incident occurs.
2) **Adjuster**
   a) Representative of our company assigned to a particular claim to manage and process the full flow.
   b) Is responsible of assigning auditor, policy investigator and vendor depending on the claim's needs.
   c) Works under a manager.
3) **Auditor**
   a) Assigned by the adjuster
   b) Is responsible for checking if the raised claim is valid or not.
   c) Has the privilege to reject a claim if found fraudulent.
4) **Policy investigator**
   a) Assigned by the adjuster
   b) Is responsible for checking if the policy of the claimant/policyholder is valid or expired.
   c) Has the privilege to reject a claim if the policy is found invalid or expired.
5) **Vendor**
   a) Assigned by the adjuster.
   b) Is responsible of carrying out third party services like towing and labor, salvage etc.,
6) **Manager**
   a) Heads a number of adjusters.
   b) Overseas the claim process and issues payments if required.

III ER Model and Tables

Our auto insurance claim application uses multiple tables to store different types of data. In this section, we provide an overview

of the key dynamic and static tables used in the process (see Table I for a summary of the main tables). Figure 2 presents the ER diagram for the database structure.

III-A Static Tables

- **coverage_cd**: Lists coverage and sub-coverage types for auto policies.
- **payment_type_cd**: Specifies the type of payments that can be made for auto claims.
- **injury_cd**: Lists possible injuries related auto claims.
- **vendor_cd**: Stores information on vendors providing services like towing/labor, salvage, glass repair.
- **auditor_cd:** Lists our company's available auditor details.
- **policy_investigator_cd:** Lists our company's available policy investigator details.
- **manager_cd:** List our company's managers details.
- **adjuster_cd:** List our company's adjusters details.

III-B Dynamic Tables

- **policy_holder**: Holds policyholder information including personal details.
- **auto_policy**: Stores data about the auto insurance policies owned by policyholders, including coverage, premium, and status.
- **claim**: Manages information on filed claims such as type, amount, and status.
- **payment**: Contains details of payments made for settled/ongoing claims.
- **injury**: Logs injuries related to specific claims, with injury codes, severity and medical expenses.
- **status:** Contains processing statuses of employee operations(eg. auditor's, policy investigator's, vendor's)

TABLE I: Available tables in our application - Overview

Table Name	Description	Primary Keys
policy_holder	Stores customer information	policy_holder_id
auto_policy	Stores auto insurance policy details	policy_id
claim	Stores claims filed by policyholders	claim_id
payment	Contains payment details for claims	payment_id, claim_id
injury	Logs injuries related to specific claims	claim_id, injury_code
status	Contains processing status of employee operations	claim_id
coverage_cd	Defines coverage types for policies	coverage_code, sub_coverage_code
payment_type_cd	Specifies types of payments	payment_type_code
injury_cd	Lists possible injuries related to claims	injury_code
vendor_cd	Stores information on vendors providing services	vendor_id
auditor_cd	Lists our company's available auditor details	auditor_id
policy_investigator_cd	Lists our company's available policy investigator details	investigator_id
adjuster_cd	List our company's adjusters details	adjuster_id
manager_cd	List our company's managers details	manager_id

IV Security Policies

This section defines the security policies enforced for each role within the system using Oracle VPD's fine-grained access control. Each policy ensures that users can only access data relevant to their role. All policies are implemented within the `insurance` schema.

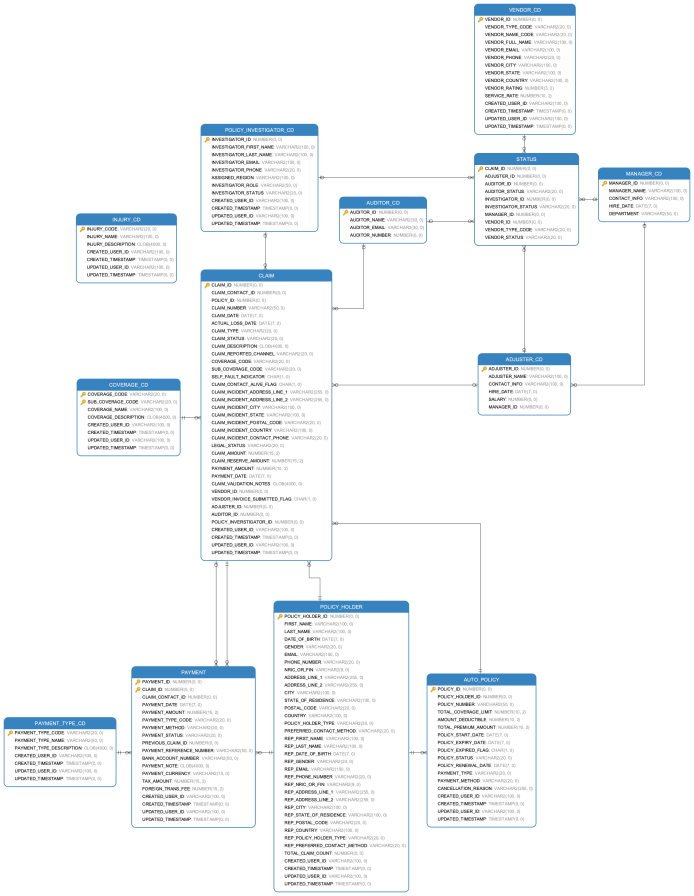

Fig. 2: ER Diagram for the Insurance Application

IV-A Environment Context for Insert and Update

For all 'insert' and 'update' operations, the current user's ID is automatically assigned to the `created_user_id` and `updated_user_id` fields.

```
BEGIN
  FOR t IN (SELECT table_name FROM all_tables WHERE
      owner = 'INSURANCE') LOOP
    EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER
        trg_set_user_ids_' || t.table_name ||
      ' BEFORE INSERT OR UPDATE ON ' || t.table_name
          ||
      ' FOR EACH ROW
       BEGIN
         IF INSERTING THEN
           :NEW.created_user_id := SYS_CONTEXT('
               USERENV', ''SESSION_USER'');
         END IF;
         IF UPDATING THEN
           :NEW.updated_user_id := SYS_CONTEXT('
               USERENV', ''SESSION_USER'');
         END IF;
       END;';
  END LOOP;
END;
/
```

IV-B Policy Holder

The following Oracle Virtual Private Database (VPD) policies restrict policy holders to view and update their own data.

- **Policy 1: Access own profile on the `policy_holder` table.** Policy holders can view and update only their own records, where the `policy_holder_id` matches their own ID. However, they are not permitted to insert a new record, which is actually operated be the admin.
- **Policy 2: View own auto-policies on the `auto_policy` table.** Policy holders can view records in the `auto_policy` table where the `policy_holder_id` matches their own ID. However, they are not permitted to update records in this table.
- **Policy 3: View own claims on the `claim` table.** Policy holders can view records in the `claim` table where the `policy_holder_id` matches their own ID. Similar to the auto-policies, they are not permitted to update records in this table.

IV-B1 VPD Function Definition

The function checks the current `policy_holder_id` from the environment context and restricts access to rows based on this value.

```
CREATE OR REPLACE FUNCTION policy_holder_vpd_function
    (p_schema VARCHAR2, p_object VARCHAR2)
RETURN VARCHAR2 AS
BEGIN
  RETURN 'policy_holder_id = SYS_CONTEXT(''USERENV'',
      ''SESSION_USER_ID'')';
END policy_holder_vpd_function;
```

IV-B2 VPD Policy Application

This policy is applied to the `policy_holder`, `auto_policy`, and `claim` tables, restricting `select`, `insert`, and `update` operations.

```
BEGIN
  DBMS_RLS.ADD_POLICY(
    object_schema   => 'INSURANCE',
    object_name     => 'policy_holder',
    policy_name     => 'policy_holder_select_policy',
    function_schema=> 'INSURANCE',
    policy_function=> 'policy_holder_vpd_function',
    statement_types=> 'SELECT, UPDATE'
  );

  DBMS_RLS.ADD_POLICY(
    object_schema   => 'INSURANCE',
    object_name     => 'auto_policy',
    policy_name     => 'auto_policy_select_policy',
    function_schema=> 'INSURANCE',
    policy_function=> 'policy_holder_vpd_function',
    statement_types=> 'SELECT'
  );

  DBMS_RLS.ADD_POLICY(
    object_schema   => 'INSURANCE',
    object_name     => 'claim',
    policy_name     => 'claim_select_policy',
    function_schema=> 'INSURANCE',
    policy_function=> 'policy_holder_vpd_function',
    statement_types=> 'SELECT'
  );
END;
```

IV-C Adjuster

Adjusters handle the overall claims process and are responsible for assigning an auditor, policy investigator, and requesting vendors depending on the claim's needs:

- **Policy 1: View and Update Assigned Claims.** Adjusters can view and update claims assigned to them via 'adjuster_id' in the 'claim' table.
- **Policy 2: Update Assigned Claim Status.** Adjusters can update the status of the claim based on the validation from the auditor and policy investigator.
- **Policy 3: View Restricted Info.** Adjusters can view restricted information of the auditor, policy investigator, and vendors.

IV-C1 VPD Function Definition

The function checks the role of the user using username from the environment context and restricts access to rows based on this value.

```
CREATE OR REPLACE FUNCTION insurance.
    adjuster_on_claim_and_status (
    schema_name IN VARCHAR2,
    table_name IN VARCHAR2
) RETURN VARCHAR2 AS
    condition VARCHAR2(100);
BEGIN
    IF insurance.has_adjuster_role = 1 THEN
        condition := 'ADJUSTER_ID = ''' || insurance.
            get_adjuster_id || '''';
    END IF;

    RETURN condition;
END adjuster_on_claim_and_status;


CREATE OR REPLACE FUNCTION insurance.
    adjuster_select_on_cd (
    schema_name IN VARCHAR2,
    table_name IN VARCHAR2
) RETURN VARCHAR2 AS
    condition VARCHAR2(100);
BEGIN
    IF insurance.has_adjuster_role = 1 THEN
        condition := '1=0';
    END IF;

    RETURN condition;
END adjuster_select_on_cd;
```

IV-C2 VPD Policy Definition

This policy is applied to the `claim`, `status`, `auditor_cd`, `policy_investigator_cd`, `vendor_cd`, and `payment` tables, restricting `select`, `insert`, and `update` operations.

```
BEGIN
    DBMS_RLS.ADD_POLICY(
        object_schema => 'INSURANCE',
        object_name => 'claim',
        policy_name => 'policy_adjuster_on_claim',
        function_schema => 'INSURANCE',
        policy_function => '
            adjuster_on_claim_and_status',
        statement_types => 'SELECT, UPDATE'
    );
    DBMS_RLS.ADD_POLICY(
        object_schema => 'INSURANCE',
        object_name => 'status',
        policy_name => 'policy_adjuster_on_status',
        function_schema => 'INSURANCE',
        policy_function => '
            adjuster_on_claim_and_status',
        statement_types => 'SELECT, UPDATE'
    );
    DBMS_RLS.ADD_POLICY (
        object_schema => 'INSURANCE',
        object_name => 'auditor_cd',
        policy_name => 'policy_adjuster_on_auditor_cd
            ',
        function_schema => 'INSURANCE',
        policy_function => 'adjuster_select_on_cd',
        statement_types => 'SELECT',
        sec_relevant_cols => 'auditor_name,
            auditor_email,auditor_number'
    );
```

```
    DBMS_RLS.ADD_POLICY (
        object_schema => 'INSURANCE',
        object_name => 'policy_investigator_cd',
        policy_name => '
            policy_adjuster_on_policy_investigator_cd
            ',
        function_schema => 'INSURANCE',
        policy_function => 'adjuster_select_on_cd',
        statement_types => 'SELECT',
        sec_relevant_cols => 'investigator_first_name,
            investigator_last_name,investigator_email,
            investigator_phone,assigned_region,
            investigator_role,investigator_status'
    );
    DBMS_RLS.ADD_POLICY (
        object_schema => 'INSURANCE',
        object_name => 'vendor_cd',
        policy_name => 'policy_adjuster_on_vendor_cd',
        function_schema => 'INSURANCE',
        policy_function => 'adjuster_select_on_cd',
        statement_types => 'SELECT',
        sec_relevant_cols => 'vendor_name_code,
            vendor_full_name,vendor_email,vendor_phone
            ,vendor_city,vendor_state,vendor_country,
            vendor_rating,service_rate'
    );
    DBMS_RLS.ADD_POLICY (
        object_schema => 'INSURANCE',
        object_name => 'payment',
        policy_name => 'policy_adjuster_on_payment',
        function_schema => 'INSURANCE',
        policy_function => 'adjuster_select_on_cd',
        statement_types => 'SELECT',
        sec_relevant_cols => 'payment_id,
            claim_contact_id,payment_date,
            payment_type_code,payment_method,
            previous_claim_id,payment_reference_number
            ,bank_account_number,payment_note,
            payment_currency,tax_amount,
            foreign_trans_fee'
    );
END;
```

IV-D Auditor

Auditors are required to cross-check the police reports, physically examine the incidents, and report back to the adjuster regarding the genuineness of the claim. Auditors in our application are listed in the auditor_cd table and have fine-grained access to other tables.

IV-D1 Need for Auditor to access data:

1) Auditors are assigned the cases by the adjuster and this will be assigned in two tables claims and status tables.
2) Once an auditor is assigned a case he will be notified and the status table will be the only table that he can edit.
3) He can reference the claims table where a policy will limit them to only access to the items that have been assigned to him and in the status table most of the columns are masked, only the claim_id, auditor_id, auditor_status are visible to him of which only the auditor_status is the only column he can edit.

IV-D2 VPD policies requirement and function definition:

The auditor is more of a fieldwork-oriented role so when it comes to the database aspect he would have less contribution towards the technical aspects which results in fewer policies possible.

Policy 1: Policy to view assigned claims tagged under a specific auditor_id : The auditor can view the entire claim details from the claim table, which are tagged to that specific auditor using his ID by the adjuster.

```
CREATE OR REPLACE FUNCTION
auditor_claim_lookup_policy_function(
p_schema IN VARCHAR2,
p_object IN VARCHAR2
)
RETURN VARCHAR2
IS
p_auditor_id NUMBER;
v_role_check VARCHAR2(1);

BEGIN
 v_role_check := SYS_CONTEXT('USERENV', 'SESSION_ROLES
     ');

 IF v_role_check LIKE '%AUDITOR%' THEN
   p_auditor_id := ( SELECT audior_id
                     FROM auditor_cd
                     WHERE auditor_name = SYS_CONTEXT
                         ('USERENV', 'SESSION_USER'));
   RETURN 'auditor_id = ' || p_auditor_id;

 ELSE
        RETURN NULL;
 END IF;
END;
```

Policy 2: Policy to view and update status table for internal status management : The auditor can view only a few of the columns(claim_id, auditor_id, auditor_status) that are tagged to that auditor, and have the UPDATE access only on the auditor_status where the restricted access of columns and operation will be specified with ADD_POLICY.

```
CREATE OR REPLACE FUNCTION
auditor_status_lookup_policy_function(
   p_schema IN VARCHAR2,
   p_object IN VARCHAR2
)
RETURN VARCHAR2
IS
   p_auditor_id NUMBER;
   v_role_check VARCHAR2(1);
BEGIN
   v_role_check := SYS_CONTEXT('USERENV', '
       SESSION_ROLES');

   IF v_role_check LIKE '%AUDITOR%' THEN
      p_auditor_id := (SELECT auditor_id
                       FROM auditor_cd  -- Replace
                           with actual table that maps
                            users to auditor_id
                       WHERE auditor_name =
                           SYS_CONTEXT('USERENV', '
                           SESSION_USER'));

      RETURN 'auditor_id = ' || p_auditor_id;
   ELSE
      RETURN NULL;
   END IF;
END;
```

IV-D3 VPD policies definition:

Policy 1:

```
BEGIN
   DBMS_RLS.ADD_POLICY(
      object_schema   => 'INSURANCE',
      object_name     => 'claim',
      policy_name     => 'auditor_policy',
      function_schema => 'INSURANCE',
      policy_function => '
          auditor_claim_lookup_policy_function',
      statement_types => 'SELECT'
   );
END;
```

Policy 2:

```
BEGIN
   DBMS_RLS.ADD_POLICY(
      object_schema   => 'INSURANCE',
      object_name     => 'status',
      policy_name     => 'auditor_status_policy',
      function_schema => 'INSURANCE',
      policy_function => '
          auditor_status_lookup_policy_function',
      statement_types => 'SELECT, UPDATE',
      sec_relevant_cols => 'adjuster_id,
          investigator_id, investigator_status,
          manager_id, vendor_id, vendor_type_code,
          vendor_status',
      sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS
   );
END;
```

Queries that auditor can run:
Policy 1:

```
SELECT * FROM CLAIM;
```

Policy 2-a:

```
SELECT * FROM STATUS;
```

Policy 2-b:

```
UPDATE STATUS
SET auditor_status = 'Approved' // can be 'Rejected'
    as well.
WHERE claim_id = xxxx;
```

IV-E Investigator

Investigators validate claimant's auto policy, assess policy coverage and check for the policy's validity :

- **Policy 1: View Claims assigned to Investigator:** Investigators can only view claims where their id matches the one assigned to the claim. This helps ensure that investigators can access data related to only those claims which have been assigned to them.

```
CREATE OR REPLACE FUNCTION policy_investigator_vpd
     (
    schema_name IN VARCHAR2,
    table_name IN VARCHAR2
) RETURN VARCHAR2 AS
    v_predicate VARCHAR2(4000);
BEGIN
    -- Restrict data based on assigned
        investigator ID or assigned region
    v_predicate := 'investigator_id = SYS_CONTEXT
        (''USERENV'', ''SESSION_USER'') OR
        assigned_region = '''
                    || SYS_CONTEXT(''USERENV'', ''
                        REGION'') || '''';
    RETURN v_predicate;
END policy_investigator_vpd;
/

BEGIN
    DBMS_RLS.ADD_POLICY(
        object_schema   => 'INSURANCE',  -- Your
            schema
        object_name     => 'CLAIMS',  -- Table
            where VPD is applied
        policy_name     => '
            policy_investigator_policy',
        function_schema => 'INSURANCE',  -- Schema
             where VPD function is located
        policy_function => '
            policy_investigator_vpd',
        statement_types => 'SELECT, INSERT, UPDATE
            , DELETE',  -- Apply to all DML actions
        update_check    => TRUE  -- Enforce VPD on
             updates
    );
END;
```

- **Policy 2: View Policy Coverage for Claims:** Investigators can check policy coverage details for claims they are handling.

```
    CREATE OR REPLACE FUNCTION policy_coverage_vpd
         (
    schema_name IN VARCHAR2,
    table_name IN VARCHAR2
) RETURN VARCHAR2 AS
    v_predicate VARCHAR2(4000);
BEGIN
    -- Restrict data based on investigator's
        assigned claim and active status
    v_predicate := 'policy_status = ''active'' AND
         investigator_id = SYS_CONTEXT(''USERENV
        '', ''SESSION_USER'')';
    RETURN v_predicate;
END policy_coverage_vpd;
/

BEGIN
    DBMS_RLS.ADD_POLICY(
        object_schema   => 'INSURANCE',  -- Your
            schema
        object_name     => 'AUTO_POLICY',  --
            Table where VPD is applied
        policy_name     => 'policy_coverage_policy
            ',
        function_schema => 'INSURANCE',
        policy_function => 'policy_coverage_vpd',
        statement_types => 'SELECT'
    );
```

- **Policy 3: View Legal status of Claims:** Investigators should only access the legal status of claims they are currently assigned to investigate.

```
   CREATE OR REPLACE FUNCTION
       claim_legal_status_vpd (
    schema_name IN VARCHAR2,
    table_name IN VARCHAR2
) RETURN VARCHAR2 AS
    v_predicate VARCHAR2(4000);
BEGIN
    -- Restrict access based on legal status
    v_predicate := 'legal_status = ''None'' OR
        legal_status = ''Ongoing'' OR legal_status
         = ''resolved''';
    RETURN v_predicate;
END claim_legal_status_vpd;
/

BEGIN
    DBMS_RLS.ADD_POLICY(
        object_schema   => 'INSURANCE',  -- Your
            schema
        object_name     => 'CLAIM',  -- Table
            where VPD is applied
        policy_name     => 'legal_status_policy',
        function_schema => 'INSURANCE',
        policy_function => 'claim_legal_status_vpd
            ',
        statement_types => 'SELECT'
    );
```

IV-F Manager

Managers oversee the overall operations and have the highest level of access:

- **Policy 1: Full Access to All Data.** Managers can access all tables and rows within the database without restrictions. Managers need complete visibility to ensure they can oversee operations, make decisions, and intervene in case of any issues.
  Full access allows managers to perform audits, investigate discrepancies, and ensure compliance with organizational and regulatory standards.

```
CREATE OR REPLACE FUNCTION manager_on_vendor (
    p_schema IN VARCHAR2,
    p_object IN VARCHAR2
) RETURN VARCHAR2 AS
    condition VARCHAR2(100);
BEGIN
    -- Check if the current user has a manager
        role
    IF insurance.has_manager_role = 1 THEN
        -- Allow access to all records for
            managers
        condition := NULL;  -- NULL condition
            grants access to all rows
    END IF;
    RETURN condition;
END manager_on_vendor;
```

- **Policy 2: Manage Security Policies.** Managers can modify and apply security policies across all other roles.
  This policy empowers managers to oversee and adapt the organization's security posture, granting them the authority to modify role-based access as needed while ensuring that only trusted individuals can alter critical security policies. This reduces the risk of unauthorized access, prevents misuse of privileges by lower-level roles, and effectively maintains data integrity and confidentiality across the system.

```
    CREATE OR REPLACE FUNCTION
        manager_security_policies (
    p_schema IN VARCHAR2,
    p_object IN VARCHAR2
) RETURN VARCHAR2 AS
    condition VARCHAR2(100);
BEGIN
    -- Check if the current user has a manager
        role
    IF insurance.has_manager_role = 1 THEN
        -- Allow access to create and modify
            security policies
        condition := NULL;  -- NULL condition
            grants access to all actions
    ELSE
        condition := '1=0'; -- Deny access if not
            a manager
    END IF;

    RETURN condition;
END manager_security_policies;
/
```

- **Policy 3: Assign Vendors for Claim.** Adjusters can assign vendors to claims by updating the 'vendor_id' and 'vendor_name'.
  This policy tightly controls vendor assignments, significantly reducing the risk of fraud or errors in vendor selection. By restricting this capability to specific roles, it safeguards against unauthorized or inappropriate assignments, thereby enhancing trust in the vendor-claim process and ensuring accountability.

```
CREATE OR REPLACE FUNCTION manager_on_status (
    p_schema IN VARCHAR2,
    p_object IN VARCHAR2
) RETURN VARCHAR2 AS
    condition VARCHAR2(4000); -- Adjusted size for
        more complex conditions
BEGIN
    -- Check if the current user has a manager
        role
    IF insurance.has_manager_role = 1 THEN
        -- Allow update only on vendor_id,
            vendor_name, and vendor_status columns
        condition := ' (UPDATING AND (' ||
                     ' :NEW.vendor_id IS NOT NULL
                         OR ' ||
                     ' :NEW.vendor_name IS NOT
                         NULL OR ' ||
                     ' :NEW.vendor_status IS NOT
                         NULL)) ';
    ELSE
        condition := '1=0'; -- Deny access if not
            a manager
    END IF;

    RETURN condition;
END manager_on_status;
```

IV-G Vendor

Vendors are required to provide third party services to the claimants such as towing and labor, salvage and glass repairs. Vendors for our application vendor name code, defined in vendor_cd table.

IV-G1 Need for vendors to access data:

1) Vendors in our real time application will be needing access to the tables pertaining only to their operations, to let our insurance company know of the service status.
2) We have static vendor_cd table that contains the list of available vendors for each service and their primary ids are tagged to each claim if the vendor service is needed.
3) Vendors need read access to the claim location data and update access to let know our insurance company that he/she has uploaded all the invoices after service. This is explained as follows:

IV-G2 VPD policies requirement and function definition:

For Policy 1 and 2, the scenarios are viewing CLAIM table, but the policy functions are different(described below). For this purpose, in real time application, we assume that if vendor wants to see his/her data, he/she is presented with 2 options.

- Option 1: to see all pending claims assigned to them.
- Option 2: to see all claims filed in their state and assigned to them.

The vendor should select an option. That option will be set in application session context and will be used in the policy functions. If vendor chooses Option 1,

```
BEGIN DBMS_SESSION.SET_CONTEXT('USERENV', 'SCENARIO',
    '1'); END;
```

If vendor chooses Option 2,

```
BEGIN DBMS_SESSION.SET_CONTEXT('USERENV', 'SCENARIO',
    '2'); END;
```

- **Policy 1: Policy to view open claims tagged under the vendor :** A vendor should view the particular claim details(claim location details) from the claim table, which are tagged to that vendor, and which are in 'Pending' status.

```
CREATE OR REPLACE FUNCTION
    vendor_claim_lookup_policy_function(
  p_schema IN VARCHAR2,
  p_object IN VARCHAR2
)
RETURN VARCHAR2
IS
  p_vendor_id NUMBER;
  p_scenario VARCHAR2(20);
BEGIN
  -- Scenario(either total open claims lookup(1) (
      or) self-state claims lookup(2)) is got from
       the context
  p_scenario := SYS_CONTEXT('USERENV', 'SCENARIO')
      ;

  -- Apply this policy if SCENARIO = '1'
  IF p_scenario = '1' THEN
    p_vendor_id := ( SELECT vendor_id
                      FROM vendor_cd
                     WHERE vendor_name_code =
                         SYS_CONTEXT('USERENV', '
                         SESSION_USER'));
    RETURN 'claim_status = ''Pending'' AND
        vendor_id = ' || p_vendor_id;
  ELSE
    RETURN NULL;
  END IF;
END;
```

- **Policy 2: Policy to view all claims within the vendor's state :** A vendor can view all the claims (irrespective of the claim status) that are tagged to that vendor, in their specific operating state.

```
CREATE OR REPLACE FUNCTION
    vendor_claim_self_state_lookup_policy_function
    (
  p_schema IN VARCHAR2,
  p_object IN VARCHAR2
)
RETURN VARCHAR2
IS
  p_state VARCHAR2(100);
  p_scenario VARCHAR2(10);
BEGIN
  -- Scenario(either total open claims lookup(1) (
      or) self-state claims lookup(2)) is got from
       the context
  p_scenario := SYS_CONTEXT('USERENV', 'SCENARIO')
      ;

  -- Apply this policy if SCENARIO = '2'
  IF p_scenario = '2' THEN
    SELECT p_state
      FROM vendor_cd
     WHERE vendor_name_code = SYS_CONTEXT('USERENV
         ', 'SESSION_USER');
    RETURN 'claim_incident_state = ''' || p_state
        || '''';
  ELSE
    RETURN NULL;
  END IF;
END;
```

- **Policy 3: Policy to update invoice submission indicator :** A vendor can upload the related documents after service to our insurance company and has the access to update that the upload/submission is successful in the claims table(through a flag). This is a mandated process for vendor, lets say if for this month, the vendor is assigned 10 claims, he/she should complete the service for them and update all the 10 claims that the invoices are submitted; even if one invoice is not submitted, he/she is not allowed to update in our application.

```
CREATE OR REPLACE FUNCTION
    vendor_claim_update_policy_function(
  p_schema IN VARCHAR2,
  p_object IN VARCHAR2
)
RETURN VARCHAR2
IS
  p_vendor_id NUMBER;
BEGIN
  p_vendor_id := (SELECT vendor_id
                   FROM vendor_cd
                  WHERE vendor_name_code =
                      SYS_CONTEXT('USERENV', '
                      SESSION_USER'));

  RETURN 'vendor_id = ' || p_vendor_id || ' AND
      vendor_invoice_submitted_flag = ''N''';
END;
```

IV-G3 VPD policies definition:

Policy 1:

```
BEGIN
  DBMS_RLS.ADD_POLICY(
    object_schema   => 'INSURANCE',
    object_name     => 'claim',
    policy_name     => 'vendor_claim_lookup_policy',
    function_schema => 'INSURANCE',
    policy_function => '
        vendor_claim_lookup_policy_function',
    statement_types => 'SELECT',
    sec_relevant_cols => 'claim_id,
        claim_incident_address_line_1,
        claim_incident_address_line_2,
        claim_incident_city, claim_incident_state,
        claim_incident_country,
        claim_incident_postal_code,
        claim_incident_contact_phone, vendor_id,
        vendor_invoice_submitted_flag');
END;
```

Policy 2:

```
BEGIN
  DBMS_RLS.ADD_POLICY(
    object_schema   => 'INSURANCE',
    object_name     => 'claim',
    policy_name     => '
        vendor_claim_self_state_lookup_policy',
    function_schema => 'INSURANCE',
    policy_function => '
        vendor_claim_self_state_lookup_policy_function
        ',
statement_types => 'SELECT',
sec_relevant_cols => 'claim_id,
    claim_incident_address_line_1,
    claim_incident_address_line_2, claim_incident_city
    , claim_incident_state, claim_incident_country,
    claim_incident_postal_code,
    claim_incident_contact_phone, vendor_id,
    vendor_invoice_submitted_flag');
END;
```

Policy 3:

```
BEGIN
  DBMS_RLS.ADD_POLICY(
    object_schema   => 'INSURANCE',
    object_name     => 'claim',
    policy_name     => 'vendor_claim_update_policy',
    function_schema => 'INSURANCE',
    policy_function => '
        vendor_claim_update_policy_function',
```

```
    statement_types => 'UPDATE',
    update_check    => TRUE,
    sec_relevant_cols => '
        vendor_invoice_submitted_flag',
  );
END;
```

Queries that vendor will run:
Policy 1:

```
SELECT * FROM CLAIM;
```

Policy 2:

```
SELECT * FROM CLAIM;
```

Policy 3:

```
UPDATE CLAIM
SET vendor_invoice_submitted_flag = 'Y';
```

V Conclusion

Oracle's VPD offers a flexible and robust solution for ensuring role-based data access control in an insurance business context. This implementation helps safeguard sensitive customer and policy data while still allowing business operations to continue smoothly. With the implemented security policies, our application is able to ensure robust protection and maintain a higher standard of data integrity and confidentiality.