Group 15

CS5322 Project 2 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

(wait we below also have) Introduction

Enterprise Planning Resource (ERP) system automates and streamlines processes in big organizations which often involve multiple parties by integrating the various functions of an organization through an unified system. While the benefits of such integrations are often talked about, it is also important to address new challenges in the area data privacy – such system has to ensure that restrictions are finely tuned so that employees have just the right amount of access to perform their tasks, but not too rigid such that granting additional access is made impossible or unreasonably difficult.

Taking into account such nuanced data access requirements in ERP systems, the team has decided to implement the datastore portion of such a system with Oracle Database and Oracle Label Security (OLS) for access control. The aim of this project is to show how OLS is able to provide granular yet flexible access control in settings where data is shared across multiple users physically but access to part of the data should be restricted logically to a set of users.

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 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.

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 tab.

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.

Schema Design for an HR Application with MAC-based Access Control:

Tables:

1. **Employees**:
   - EmployeeID (Primary Key)
   - FirstName
   - LastName
   - Email
   - Phone
   - Position
   - DepartmentID (Foreign Key)
   - Address
   - DateOfJoining
   - DateOfBirth
   - Salary
   - SecurityLevel (Values: Restricted, Confidential, Highly Confidential, Secret)

2. **Departments**:
   - DepartmentID (Primary Key)
   - DepartmentName

3. **PerformanceReviews**:
   - ReviewID (Primary Key)
   - EmployeeID (Foreign Key)
   - ReviewText
   - ReviewDate
   - Grading (Values: 1, 2, 3, 4, 5)
   - SecurityLevel

4. **MedicalRecords**:
   - RecordID (Primary Key)
   - EmployeeID (Foreign Key)
   - HealthDetails
   - MedicalHistory
   - SecurityLevel

5. **BackgroundChecks**:
   - CheckID (Primary Key)
   - EmployeeID (Foreign Key)
   - BackgroundDetails
   - DateChecked

- SecurityLevel

6. **SalaryBreakdowns**:
   - BreakdownID (Primary Key)
   - EmployeeID (Foreign Key)
   - SalaryDetails
   - Bonuses
   - Deductions
   - SecurityLevel

7. **UserRoles**:
   - UserID (Primary Key)
   - EmployeeID (Foreign Key)
   - Role (Values: HR, Manager, Employee)
   - AccessLevel (Values: Restricted, Confidential, Highly Confidential, Secret)

8. **TimeOffBalance**:
   - TimeOffID (Primary Key)
   - EmployeeID (Foreign Key)
   - TimeOffType (Values: Annual Leave, Child Care Leave, Medical Leave, Unpaid Leave, Compassionate Leave)
   - Balance (number of days or hours available)
   - DateLastUpdated
   - SecurityLevel

Access Control Using MAC:

- **Restricted**:
  - Access to basic contact information.
  - Employee’s own data (read-only).

- **Confidential**:
  - All "Restricted" access.
  - Access to an employee's performance reviews.

- **Highly Confidential**:
  - All "Restricted" and "Confidential" access.
  - Access to medical records, salary breakdowns, and time-off balance for direct reports.

- **Secret**:
  - Access to all data in the system, including background checks.
  - Full edit and delete capabilities.

****Write-up Situation:****

Situation:

An employee, James Smith, has recently been promoted to a managerial position. As part of his new role, he needs to have access to performance reviews of his direct reports, their contact information, and their medical records in case of emergencies.

However, when James logs into the HR system, he finds that he can only view basic contact information. The Performance Reviews and Medical Records sections are locked out for him, indicating that his access level is still set to "Restricted".

This indicates a flaw in the access control policy's automation. When an employee is promoted, their access level should be updated automatically based on their new position's requirements. However, in James' case, this did not happen.

To resolve the issue, the HR department, which has "Secret" access level, must manually update James' role in the `UserRoles` table, setting his AccessLevel to "Highly Confidential". This will ensure James can fulfill his managerial duties effectively, without compromising the security and integrity of sensitive data.

This schema and the situation highlight the complexity of implementing Mandatory Access Control in a real-world application. Proper audit mechanisms, automation, and periodic reviews are essential to ensure the integrity and security of data in such systems.

Table Design

```
-- Employees Table
CREATE TABLE Employees (
    EmployeeID NUMBER PRIMARY KEY,
    FirstName VARCHAR2(50 BYTE),
    LastName VARCHAR2(50 BYTE),
    Email VARCHAR2(100 BYTE) UNIQUE,
    Phone VARCHAR2(15 BYTE),
    Position VARCHAR2(50 BYTE),
    DepartmentID NUMBER REFERENCES Departments(DepartmentID),
    Address VARCHAR2(255 BYTE),
    DateOfJoining DATE,
```

```
    DateOfBirth DATE,
    Salary NUMBER(10,2),
    SecurityLevel VARCHAR2(20 BYTE) CHECK (SecurityLevel IN ('Restricted', 'Confidential',
'Highly Confidential', 'Secret'))
);

-- Departments Table
CREATE TABLE Departments (
    DepartmentID NUMBER PRIMARY KEY,
    DepartmentName VARCHAR2(100 BYTE)
);

-- PerformanceReviews Table
CREATE TABLE PerformanceReviews (
    ReviewID NUMBER PRIMARY KEY,
    EmployeeID NUMBER REFERENCES Employees(EmployeeID),
    ReviewText VARCHAR2(500 BYTE),
    ReviewDate DATE,
    Grading NUMBER CHECK (Grading BETWEEN 1 AND 5),
    SecurityLevel VARCHAR2(20 BYTE) CHECK (SecurityLevel IN ('Restricted', 'Confidential',
'Highly Confidential', 'Secret'))
);

-- MedicalRecords Table
CREATE TABLE MedicalRecords (
    RecordID NUMBER PRIMARY KEY,
    EmployeeID NUMBER REFERENCES Employees(EmployeeID),
    HealthDetails VARCHAR2(500 BYTE),
    MedicalHistory VARCHAR2(1000 BYTE),
    SecurityLevel VARCHAR2(20 BYTE) CHECK (SecurityLevel IN ('Restricted', 'Confidential',
'Highly Confidential', 'Secret'))
);

-- BackgroundChecks Table
CREATE TABLE BackgroundChecks (
    CheckID NUMBER PRIMARY KEY,
    EmployeeID NUMBER REFERENCES Employees(EmployeeID),
    BackgroundDetails VARCHAR2(1000 BYTE),
    DateChecked DATE,
    SecurityLevel VARCHAR2(20 BYTE) CHECK (SecurityLevel IN ('Restricted', 'Confidential',
'Highly Confidential', 'Secret'))
);

-- SalaryBreakdowns Table
```

```
CREATE TABLE SalaryBreakdowns (
    BreakdownID NUMBER PRIMARY KEY,
    EmployeeID NUMBER REFERENCES Employees(EmployeeID),
    SalaryDetails VARCHAR2(500 BYTE),
    Bonuses NUMBER(10,2),
    Deductions NUMBER(10,2),
    SecurityLevel VARCHAR2(20 BYTE) CHECK (SecurityLevel IN ('Restricted', 'Confidential',
'Highly Confidential', 'Secret'))
);

-- UserRoles Table
CREATE TABLE UserRoles (
    UserID NUMBER PRIMARY KEY,
    EmployeeID NUMBER REFERENCES Employees(EmployeeID),
    Role VARCHAR2(50 BYTE) CHECK (Role IN ('HR', 'Manager', 'Employee')),
    AccessLevel VARCHAR2(20 BYTE) CHECK (AccessLevel IN ('Restricted', 'Confidential',
'Highly Confidential', 'Secret'))
);

-- TimeOffBalance Table
CREATE TABLE TimeOffBalance (
    TimeOffID NUMBER PRIMARY KEY,
    EmployeeID NUMBER REFERENCES Employees(EmployeeID),
    TimeOffType VARCHAR2(50 BYTE) CHECK (TimeOffType IN ('Annual Leave', 'Child Care
Leave', 'Medical Leave', 'Unpaid Leave', 'Compassionate Leave')),
    Balance NUMBER(3) CHECK (Balance >= 0),
    DateLastUpdated DATE,
    SecurityLevel VARCHAR2(20 BYTE) CHECK (SecurityLevel IN ('Restricted', 'Confidential',
'Highly Confidential', 'Secret'))
);
```

Certainly. Here are 10 examples of Mandatory Access Control (MAC) based access control policies for the given tables:

1. **Hiring Confidentiality:**
- **Situation**: When hiring for a new executive position, HR compiles information about potential candidates in the system. This data is highly sensitive as it could impact stock prices, company morale, or reveal strategies if leaked.
- **Policy**: All data related to executive hiring is classified as 'Secret'. Only the CEO, CFO, HR Specialist (assigned for executive hiring), and the respective candidate can view and modify this information.

2. **Performance Review Access**:
   - **Policy**: Performance review data is classified based on the level of the employee:
     - 'Restricted' for junior employees.
     - 'Confidential' for mid-level managers.
     - 'Highly Confidential' for senior managers.
     - 'Secret' for C-level executives.
   Managers can only access the performance review data of their direct reports.

3. **Salary Information**:
   - **Policy**: All salary information is labeled 'Highly Confidential'. Only HR and the respective employee can view an employee's salary. Managers can request access but can't modify the information.

4. **Employee Personal Data**:
   - **Policy**: All personal data like `date_of_birth` is labeled 'Confidential'. Only the respective employee and HR can view or modify this data.

5. **Time-Off Balance**:
   - **Policy**: Employees' time-off balances are labeled 'Confidential'. Only the respective employee, their direct manager, and HR can view this information.

6. **Department Strategies**:
   - **Situation**: Departments may have strategies or plans that shouldn't be disclosed to other departments.
   - **Policy**: Each department's strategic information, if stored, is labeled 'Highly Confidential' and can only be accessed by members of that department and C-level executives.

~~7. **Employee Onboarding**:~~
~~- **Situation**: When a new employee is onboarded, they undergo training and receive company resources.~~
~~- **Policy**: All onboarding materials and training modules, if stored, are classified as 'Restricted'. This ensures new employees can access necessary resources without exposing them to sensitive information prematurely.~~

~~8. **Vendor Contracts**:~~
~~- **Situation**: The company may have contracts with various vendors providing IT services, marketing platforms, etc.~~
~~- **Policy**: All vendor contracts are labeled 'Confidential'. Only the respective department that deals with the vendor and the legal department can access these contracts.~~

~~9. **Employee Grievances**:~~
~~- **Situation**: Employees can report grievances or issues they face in the company.~~

~~- **Policy**: All grievance reports are labeled 'Highly Confidential'. Only HR and C-level executives can access these reports to ensure the anonymity and safety of the reporting employee.~~

~~**10. **IT Security Protocols**:**~~
~~- **Situation**: The IT department has specific security protocols and measures to safeguard company data.~~
~~- **Policy**: All IT security protocols and passwords, if stored, are labeled 'Secret'. Only the IT department and CTO can access this information to maintain system security.~~

For the MAC policies to be effective in the Oracle environment, they will require the use of Oracle Label Security (OLS). You would define data labels like 'Restricted', 'Confidential', 'Highly Confidential', and 'Secret' and assign them to the data. Access to the data would then be controlled based on these labels and the user's clearance level.

Create user
Use the company account to run all the below
Users: employee_1, manager_1, director_1, hr_1, vendor_1

Create user with password	CREATE USER <USERNAME> IDENTIFIED BY password_here;
Grant Privileges	-- Grant connect and resource roles to the user GRANT CONNECT, RESOURCE TO <USERNAME>; -- If you want the user to have the ability to create a session: GRANT CREATE SESSION TO <USERNAME>;
If users need to be able to create table	Warning: Do not run for user unless need to create table -- Grant the user the ability to create tables: GRANT CREATE TABLE TO <USERNAME>; ALTER USER <USERNAME> QUOTA UNLIMITED ON USERS;

Grant role to the user	GRANT role_name TO <USERNAME>;
Revoke Role	REVOKE role_name FROM <USERNAME>;
Drop the role	DROP ROLE role_name;

Groups -> User roles (Manager, employee)
Levels will be assigned to the groups for their access level
Compartments will be the departments that each of the employee falls into

3 Levels, (Restricted, Confidential, Secret)
2 Groups, (Employee, Manager)
2 Compartment (HR/IT)
More types of manager use the compartment to assign level

Sure, I'll provide a conceptual use case for a HR system in Oracle that utilizes Oracle Label Security (OLS), Groups, Compartments, and Levels to enforce data access controls.

****Use Case: HR System in Oracle****

1. **Entities:**
- Employees: They have various personal and professional details stored.

- Departments: Different business units or teams within the company.
- PerformanceReviews: Employee performance metrics and feedback.

2. **Security Labels:**

a. **Groups:** Define the primary affiliations or units within the company.
- HR
- Engineering
- Sales
- Finance
- Executive

b. **Compartments:** Separate data based on its nature.
- PersonalInfo: Personal details of employees.
- ProfessionalInfo: Professional details like salary, position, etc.
- Performance: Performance reviews and metrics.

c. **Levels:** Define the hierarchy of data sensitivity.
- Restricted
- Confidential
- Highly Confidential
- Secret

3. **Access Policies:**

a. HR group can access:
- PersonalInfo, ProfessionalInfo, and Performance compartments at all security levels.

b. Engineering, Sales, and Finance groups:
- Can access ProfessionalInfo at 'Restricted' and 'Confidential' levels.
- Can't access Performance or PersonalInfo compartments.

c. Executive group:
- Can access all compartments at all levels.

4. **Implementation using OLS:**

a. For each data row in the Employees and PerformanceReviews tables, associate an OLS label based on the employee's department and the nature of the data. For example, an employee's salary might have a label of `Finance:ProfessionalInfo:Confidential`.

b. For each user, set a session label and define which labels they can read and write to. For instance:

- A HR user might have the session label `HR:PersonalInfo:Secret` and have read/write access to all labels within the HR group.
- An engineer might have the session label `Engineering:ProfessionalInfo:Confidential` and can read only `Restricted` and `Confidential` labels in the ProfessionalInfo compartment.

c. Use OLS policies to enforce these label-based access controls on the Employees and PerformanceReviews tables.

5. **Use Case Scenario:**

a. An HR manager wants to update the salary of an engineer. They log into the HR system, and because they belong to the HR group, they can access the engineer's `ProfessionalInfo` details at the `Confidential` level and make the update.

b. An engineer wants to see their own salary. They can access their own `ProfessionalInfo` at the `Confidential` level. However, they cannot view or modify their own performance review because it's labeled `Engineering:Performance:Highly Confidential`.

c. An executive wants to view performance reviews for the engineering department. They can access the `Performance` compartment for all groups at all levels, including `Secret`.

In practice, implementing OLS involves creating a label security policy, defining levels, compartments, and groups, assigning labels to data rows, and setting up user privileges. OLS then automatically enforces the label-based access controls based on the policies and user session labels.

Please color code your stuff to track

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

Users: hr_1, globalit_1, itsupport_1

Tasks:

1. Create the tables and users
2. DAC for tables and users
3. Insert Dummy Data
4. Create OLS Container (policy = general_ols_policy)
    a. Security Labels

   b. Compartment Labels
      i. Medical, Financial, Retirement
      ii. Technical, Managerial, Safety
      iii. Report, Blueprint, Strategy, Budget, Review
   c. Group Labels
      i. Manager -> Engineer
      ii. Global IT -> IT Support, HR
5. OLS for Employees table
6. OLS for EmployeeBenefits table
7. OLS for EmployeeTrainingRecords table
8. OLS for Projects table
9. OLS for ProjectDocumentation table
10. OLS for MeetingNotes table

Task Tracking

1	2	3	4	5	6	7	8	9	10
Syl	XH/YH	Law	XH/YH	YH	Syl	Ed/YH	XH	Ed	Law

DAC Account Role

Table	Roles	
	MANAGER	EMPLOYEE
PROJECTS	SIU	S
PROJECTDOCUMENTATION	SIU	SIU
MEETINGNOTES	SIU	SIU
EMPLOYEETRAININGRECORDS	SIU	-
EMPLOYEES	SIU	SIU
EMPLOYEEBENEFITS	SIU	S

Database Account Roles

Account	Roles
ITSUPPORT_1	EMPLOYEE
GLOBALIT_1	MANAGER
HR_1	MANAGER

HR_2	EMPLOYEE
GLOBALIT_2	EMPLOYEE(Engineer)
ITSUPPORT_2	MANAGER
GLOBALIT_3	EMPLOYEE (Technician)

OLS Account Matrix

Account	**Maximum Level**	**Minimum Level**	**Default Level**	**Compartments**	**Groups**
globalit_1	S	R	S	MAN,TEC,SAF	IT_GLOBAL_MGR
hr_1	S	R	S	~~FIN, RET~~ MAN,TEC,SAF	HR_MGR
itsupport_1	C	R	R	TEC,SAF	IT_SUPPORT_TEC
globalit_2	C	R	C	TEC,SAF	IT_GLOBAL_ENG
hr_2	S	R	S	TEC,SAF	HR_REC
itsupport_2	S	R	S	MAN,TEC,SAF	IT_SUPPORT_MGR
globalit_3	C	R	R	TEC,SAF	IT_GLOBAL_TEC

8 DAC on tables
9 Create users for different use cases (VP, CEO, Manager, Employee1, Employee2)
10 Database context for VPD and OLS (user id, user role and access level)

New tables Example

Employee Training Table

Employee id	Training Certs	Security Level	TrainingType (Compartment)	Position (Group)
1	aws	unclass	IT training	Employee

2	Workplace conflict	unclass	HR training	Manager

IT Inventory

ID / SKU	Item Name	Security Level	Either: Department (Compartment)	Or: Project (Compartment)
1	Laptop	unclass	IT	IT_project1
2	Laptop	unclass	HR	HR_Recruitment

New Table Creation Code

```
CREATE TABLE EmployeeTrainingRecords (
   TrainingRecordID    NUMBER PRIMARY KEY,
   EmployeeID          NUMBER REFERENCES Employees(EmployeeID),
   TrainingCerts       VARCHAR2(255),
   SecurityLevel       VARCHAR2(50),   -- e.g., 'Restricted', 'Confidential', ‘Secret’
   TrainingType        VARCHAR2(100),  -- Compartment (e.g., 'Technical', 'Managerial',
'Safety')
   Position            VARCHAR2(100)   -- Group (e.g., 'Manager', 'Engineer', 'Technician')
);
```

```
CREATE TABLE ITInventory (
   SKU                 NUMBER PRIMARY KEY,
   ItemName            VARCHAR2(255),
   SecurityLevel       VARCHAR2(50),   -- e.g., 'Public', 'Internal', 'Restricted'
   AssociationType     VARCHAR2(50),   -- Compartment; can be 'Department' or 'Project'
   AssociatedWithName  VARCHAR2(100)   -- e.g., 'Finance', 'HR', 'ProjectA', 'ProjectB'
);
```

```
CREATE TABLE ProjectDocumentation (
   DocID               NUMBER PRIMARY KEY,
   ProjectName         VARCHAR2(255),
   DocumentType        VARCHAR2(100),  -- e.g., 'Technical', 'Financial', 'Strategy'
   Content             CLOB,
   SecurityLevel       VARCHAR2(50),   -- e.g., 'Restricted', 'Confidential', ‘Secret’
   Department          VARCHAR2(100),  -- Group; e.g., ‘Global IT’, 'HR', ‘IT Support’
   DataType            VARCHAR2(100)   -- Compartment; can be 'Report', 'Blueprint', 'Budget'
);
```

```
CREATE TABLE EmployeePersonalRecords (
   RecordID            NUMBER PRIMARY KEY,
   EmployeeID          NUMBER REFERENCES Employees(EmployeeID),
   Address             VARCHAR2(255),
   ContactNumber       VARCHAR2(15),
```

```
    EmergencyContact    VARCHAR2(255),
    SecurityLevel       VARCHAR2(50),   -- e.g., 'Private', 'Confidential'
    Position            VARCHAR2(100),  -- Group
    DataType            VARCHAR2(100)   -- Compartment; can be 'Medical', 'Financial', 'Family'
);
```

```
CREATE TABLE Employees (
    EmployeeID      NUMBER PRIMARY KEY,
    UserName        VARCHAR2(100),
    FirstName       VARCHAR2(100),
    LastName        VARCHAR2(100),
    Email           VARCHAR2(150) UNIQUE,
    DateOfJoining   DATE,
    SecurityLevel       VARCHAR2(50),   -- e.g., 'Restricted', 'Confidential', ‘Secret’
    Position        VARCHAR2(100),  -- Group; e.g., 'Manager', 'Engineer'
    Department      VARCHAR2(100)   -- Group; ‘HR’, ‘Global IT, IT Support
);
```

```
CREATE TABLE Departments (
    DepartmentID    NUMBER PRIMARY KEY,
    DepartmentName VARCHAR2(100)
);

– Departments: HR, Global IT (Parent Group), IT Support (Child Group)
```

```
CREATE TABLE Projects (
    ProjectID       NUMBER PRIMARY KEY,
    ProjectName     VARCHAR2(255),
    StartDate       DATE,
    EndDate         DATE,
    Department      VARCHAR2(100)   -- Group; ‘HR’, ‘Global IT, IT Support
);
```

```
CREATE TABLE EmployeeBenefits (
    BenefitID       NUMBER PRIMARY KEY,
    BenefitType     VARCHAR2(100),  -- e.g., 'Health Insurance', 'Stock Option'
    Details         VARCHAR2(500),
    SecurityLevel   VARCHAR2(50),   -- e.g., 'Restricted', 'Confidential', ‘Secret’
    Department      VARCHAR2(100),  -- Group; HR, Global IT, IT Support
    BenefitForm     VARCHAR2(100)   -- Compartment; can be 'Medical', 'Financial',
'Retirement'
);
```

```
CREATE TABLE MeetingNotes (
    NoteID          NUMBER PRIMARY KEY,
    MeetingDate     DATE,
    Topic           VARCHAR2(255),
    Content         CLOB,
    SecurityLevel   VARCHAR2(50),   -- e.g., 'Restricted', 'Confidential', ‘Secret’
    Department      VARCHAR2(100),  -- Group; HR, Global IT, IT Support
```

```
    MeetingType    VARCHAR2(100)   -- Compartment; e.g., 'Strategy', 'Budget', 'Review'
);
```

How to do define security levels

Security levels for each employee can be defined using a combination of attributes: their position (role), department, and any specific training or certifications they might have. Additionally, a specific security label or clearance level can be directly assigned to each employee. This label will determine the employee's access rights.

Here's a conceptual breakdown:

1. Position-Based Security:
Positions in a company typically come with inherent levels of trust and responsibility. For instance:
- A **Manager** might have access to data that regular employees don't.
- An **Executive** might have even higher access to strategic and sensitive information.

2. Department-Based Security:
Departments can often serve as a compartment in security models. The kind of data a Finance department interacts with is different from, say, the Marketing department.
- Someone in **HR** would need access to personal employee data but shouldn't necessarily access financial projections, which someone in **Finance** might need.

3. Training or Certification-Based Security:
Certain data or systems might require specific training or certifications to access.
- Only those with **data protection training** might be allowed to access personal customer information.
- Only those certified in **network security** might have access to critical system configurations.

4. Explicit Security Clearance:
In addition to the above, companies can assign explicit security clearances or labels to individuals.
- These can be in levels such as **'Restricted', 'Confidential', 'Highly Confidential', 'Secret'**, etc.
- An employee's clearance would determine which datasets or tables they can access. For instance, someone with a 'Confidential' clearance might not be able to access 'Secret' data.

To capture this in the database schema, you might introduce tables or fields like:

```sql
```

```
-- SecurityLevels Table:
CREATE TABLE SecurityLevels (
   LevelID     NUMBER PRIMARY KEY,
   LevelName   VARCHAR2(50)  -- e.g., 'Restricted', 'Confidential', 'Highly Confidential', 'Secret'
);

-- EmployeeSecurity Table:
CREATE TABLE EmployeeSecurity (
   EmployeeID      NUMBER REFERENCES Employees(EmployeeID),
   LevelID         NUMBER REFERENCES SecurityLevels(LevelID),
   TrainingCert    VARCHAR2(255),   -- e.g., 'Data Protection', 'Network Security'
   PRIMARY KEY (EmployeeID, LevelID)
);
```

In the `EmployeeSecurity` table, `EmployeeID` and `LevelID` form a composite primary key, ensuring that an employee can have multiple security levels if needed. The `TrainingCert` field can be used to further refine access based on training or certification.

Security policies, like those from Oracle Label Security, can then be used to enforce these levels and ensure employees can only access the data they're permitted to.

How to define OLS

In Oracle, Oracle Label Security (OLS) allows you to define security levels, groups, and compartments, and then assign these labels to the data rows. Users are then granted specific labels, which determine which rows they can access. Here's how you can define different security levels for groups and compartments for each database user:

Step 1: Define Policy, Levels, Groups, and Compartments

1. **Create an OLS Policy** - This is a container for the security configuration.

   ```sql
   BEGIN
     DBMS_LABELADM.CREATE_POLICY (
       policy_name          => 'HRPolicy',
       policy_description    => 'Human Resources Policy',
       enable_options       => 'OLS_DEFAULT');
   END;
   ```

```
/
```

2. **Define Security Levels** - For instance: 'Restricted', 'Confidential', 'Highly Confidential', 'Secret'.

```sql
BEGIN
  DBMS_LABELADM.CREATE_LEVEL (
    policy_name  => 'HRPolicy',
    level_name   => 'Restricted',
    level_num    => 10);

  -- ... Additional levels here ...
END;
/
```

3. **Define Compartments** - For instance: 'Training', 'Finance', 'IT'.

```sql
BEGIN
  DBMS_LABELADM.CREATE_COMPARTMENT (
    policy_name      => 'HRPolicy',
    compartment_name => 'Training',
    compartment_short_name => 'Train',
    compartment_num  => 1000);

  -- ... Additional compartments here ...
END;
/
```

4. **Define Groups** - For instance: 'Executive', 'Manager', 'Staff'.

```sql
BEGIN
  DBMS_LABELADM.CREATE_GROUP (
    policy_name => 'HRPolicy',
    group_name  => 'Executive',
    group_num   => 10);

  -- ... Additional groups here ...
END;
```

```
/
```

Step 2: Assign Label Defaults and Authorizations to Users

1. **Default Label for a User** - This is the label that will be assigned when a user inserts new rows.

```sql
BEGIN
  DBMS_LABELADM.SET_DEFAULT_LABEL (
    policy_name => 'HRPolicy',
    user_name   => 'HR_USER',
    dlabel      => 'Restricted:Training:Executive');
END;
/
```

2. **Add User Authorizations** - This sets which labels a user is allowed to access.

```sql
BEGIN
  DBMS_LABELADM.ADD_USER_LABEL (
    policy_name => 'HRPolicy',
    user_name   => 'HR_USER',
    label       => 'Restricted:Training:Executive',
    options     => 'READ,WRITE');
END;
/
```sql
BEGIN
  DBMS_LABELADM.ADD_USER_LABEL (
    policy_name => 'HRPolicy',
    user_name   => 'HR_USER',
    label       => 'Restricted:Medical:Executive',
    options     => 'READ,WRITE');
END;
/
```

Note: You can specify different labels for different access (READ and/or WRITE).

Step 3: Apply the OLS Policy to the Tables
```

Once you've defined the policy and user authorizations, you can apply the policy to your tables.

```sql
BEGIN
  DBMS_LABELADM.APPLY_POLICY(
    object_schema => 'HR_SCHEMA',
    object_name   => 'EmployeeTrainingRecords',
    policy_name   => 'HRPolicy');
END;
/
```

Step 4: Insert Data with Specific Labels

When inserting data, you can specify labels. If you don't, the default label assigned to the user will be used.

```sql
INSERT INTO EmployeeTrainingRecords VALUES ('... values ...',
'Restricted:Training:Manager');
```

The user's labels and the data row labels interact to determine access. For instance, if a user has access to 'Restricted:Training:Executive' but not 'Confidential:IT:Manager', they won't be able to see rows labeled with the latter.

This is a basic setup. Oracle OLS is a very comprehensive and flexible system, and you can customize further as per requirements, including creating more sophisticated access conditions and hierarchies.

If you're using Oracle Label Security (OLS) to manage access to rows in a table, the security label applied to each row is stored in a system-maintained column, typically named `ORA_SYSLABEL`. This column is automatically added to the table when an OLS policy is applied, and it's normally hidden from casual viewing.

However, for simplicity and demonstration purposes, if you want a dedicated visible column in your `EmployeeTrainingRecords` table to store the security label (like 'Restricted:Training:Manager'), you might name it something like `SecurityLabel`.

So, the `INSERT` statement would look something like:

```sql
INSERT INTO EmployeeTrainingRecords (EmployeeID, TrainingCerts, SecurityLevel,
TrainingType, Position, SecurityLabel)
```

VALUES (1, 'Certification A', 'High', 'Technical', 'Manager', 'Restricted:Training:Manager');
```

Note that this `SecurityLabel` column would be redundant in a real OLS implementation since `ORA_SYSLABEL` would handle the storage of the label behind the scenes. But, for demonstration or educational purposes, having a visible column might be beneficial. If using this approach, ensure consistency between the visible `SecurityLabel` and the actual OLS label.

# OLS Code

**DAC**

```
GRANT SELECT ON COMPANY.PROJECTS TO EMPLOYEE, MANAGER;
GRANT SELECT ON COMPANY.PROJECTDOCUMENTATION TO EMPLOYEE, MANAGER;
GRANT SELECT ON COMPANY.MEETINGNOTES TO EMPLOYEE, MANAGER;
~~GRANT SELECT ON COMPANY.EMPLOYEETRAININGRECORDS TO MANAGER;~~
GRANT SELECT ON COMPANY.EMPLOYEETRAININGRECORDS TO EMPLOYEE, MANAGER;
GRANT SELECT ON COMPANY.EMPLOYEES TO EMPLOYEE, MANAGER;
GRANT SELECT ON COMPANY.EMPLOYEEBENEFITS TO EMPLOYEE, MANAGER;

GRANT INSERT, UPDATE ON COMPANY.PROJECTS TO MANAGER;
GRANT INSERT, UPDATE ON COMPANY.PROJECTDOCUMENTATION TO EMPLOYEE, MANAGER;
GRANT INSERT, UPDATE ON COMPANY.MEETINGNOTES TO EMPLOYEE, MANAGER;
GRANT INSERT, UPDATE ON COMPANY.EMPLOYEETRAININGRECORDS TO MANAGER;
GRANT INSERT, UPDATE ON COMPANY.EMPLOYEES TO EMPLOYEE, MANAGER;
GRANT INSERT, UPDATE ON COMPANY.EMPLOYEEBENEFITS TO MANAGER;

GRANT EMPLOYEE TO ITSUPPORT_1; --  Technician IT Support
GRANT MANAGER TO GLOBALIT_1; -- manager global IT
GRANT MANAGER TO HR_1; -- HR Manager

GRANT MANAGER TO ITSUPPORT_2; -- manager IT Support
GRANT EMPLOYEE TO GLOBALIT_2;-- engineer global IT
GRANT EMPLOYEE TO HR_2; --recruiter

GRANT EMPLOYEE TO ITSUPPORT_3;-- engineer IT Support
GRANT EMPLOYEE TO GLOBALIT_3; -- technician global IT
GRANT EMPLOYEE TO HR_3; --recruiter
```

Security Label Setup

```
-- Create policy container
BEGIN
```
```

```
    SA_SYSDBA.CREATE_POLICY (
            policy_name      => 'general_ols_policy',
            column_name      => 'general_ols_col',
            default_options  => 'read_control, write_control');
END;

-- Create security policy labels
BEGIN
    SA_COMPONENTS.CREATE_LEVEL (
            policy_name => 'general_ols_policy',
            level_num   => 100,
            short_name  => 'R',
            long_name   => 'RESTRICTED');

    SA_COMPONENTS.CREATE_LEVEL (
            policy_name => 'general_ols_policy',
            level_num   => 200,
            short_name  => 'C',
            long_name   => 'CONFIDENTIAL');
    SA_COMPONENTS.CREATE_LEVEL (
            policy_name => 'general_ols_policy',
            level_num   => 300,
            short_name  => 'S',
            long_name   => 'SECRET');
END;

-- Create compartments
BEGIN
    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'MEDICAL',
            short_name       => 'MED',
            comp_num         =>  100);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'FINANCIAL',
            short_name       => 'FIN',
            comp_num         =>  101);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'RETIREMENT',
```

```
            short_name       => 'RET',
            comp_num         =>  102);
END;

BEGIN
    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'TECHNICAL',
            short_name       => 'TEC',
            comp_num         =>  200);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'MANAGERIAL',
            short_name       => 'MAN',
            comp_num         =>  201);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'SAFETY',
            short_name       => 'SAF',
            comp_num         =>  202);
END;

BEGIN
    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'REPORT',
            short_name       => 'REP',
            comp_num         =>  300);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'BLUEPRINT',
            short_name       => 'BLU',
            comp_num         =>  301);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'BUDGET',
            short_name       => 'BUD',
            comp_num         =>  302);
```

```
    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'STRATEGY',
            short_name       => 'STR',
            comp_num         =>  303);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'REVIEW',
            short_name       => 'REV',
            comp_num         =>  304);
END;

-- Create Groups

BEGIN
    SA_COMPONENTS.CREATE_GROUP (
            policy_name      => 'general_ols_policy',
            group_num        => 1000,
            short_name       => 'HR',
            long_name        => 'HUMAN_RESOURCE');
    SA_COMPONENTS.CREATE_GROUP (
            policy_name      => 'general_ols_policy',
            group_num        => 2000,
            short_name       => 'IT_GLOBAL',
            long_name        => 'TECHNOLOGY_GLOBAL');
    SA_COMPONENTS.CREATE_GROUP (
            policy_name      => 'general_ols_policy',
            group_num        => 2100,
            short_name       => 'IT_SUPPORT',
            long_name        => 'TECHNOLOGY_SUPPORT',
            parent_name      => 'IT_GLOBAL'
        );
END;

BEGIN
    SA_COMPONENTS.CREATE_GROUP (
            policy_name      => 'general_ols_policy',
            group_num        => 3000,
            short_name       => 'MANAGER',
            long_name        => 'MANAGER');

    SA_COMPONENTS.CREATE_GROUP (
```

```
            policy_name     => 'general_ols_policy',
            group_num       => 3100,
            short_name      => 'ENGINEER',
            long_name       => 'ENGINEER',
            parent_name     => 'MANAGER'
        );
    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 3110,
            short_name      => 'TECHNICIAN',
            long_name       => 'TECHNICIAN',
            parent_name     => 'ENGINEER'
        );
    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 3200,
            short_name      => 'RECRUITER',
            long_name       => 'RECRUITER',
            parent_name     => 'MANAGER'
        );
END;

BEGIN
    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 4000,
            short_name      => 'COMPANY',
            long_name       => 'COMPANY');

    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 4100,
            short_name      => 'HR_MGR',
            long_name       => 'HUMAN_RESOURCE_MANAGEMENT',
            parent_name     => 'COMPANY'
            );

    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 4110,
            short_name      => 'HR_REC',
            long_name       => 'HUMAN_RESOURCE_RECRUITERS',
            parent_name     => 'HR_MGR'
```

```
        );


    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 4200,
            short_name      => 'IT_GLOBAL_MGR',
            long_name       => 'TECHNOLOGY_GLOBAL_MANAGEMENT',
            parent_name     => 'COMPANY'
        );


    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 4210,
            short_name      => 'IT_SUPPORT_MGR',
            long_name       => 'TECHNOLOGY_SUPPORT_MANAGEMENT',
            parent_name     => 'IT_GLOBAL_MGR'
        );

    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 4211,
            short_name      => 'IT_SUPPORT_TEC',
            long_name       => 'TECHNOLOGY_SUPPORT_TECHNICIANS',
            parent_name     => 'IT_SUPPORT_MGR'
        );

    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 4220,
            short_name      => 'IT_GLOBAL_ENG',
            long_name       => 'TECHNOLOGY_GLOBAL_ENGINEERS',
            parent_name     => 'IT_GLOBAL_MGR'
        );
    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 4221,
            short_name      => 'IT_GLOBAL_TEC',
            long_name       => 'TECHNOLOGY_GLOBAL_TECHNICIANS',
            parent_name     => 'IT_GLOBAL_ENG'
        );
END;
```

OLS User Labels on Database Accounts

```
BEGIN
    SA_USER_ADMIN.SET_LEVELS (
            policy_name   => 'general_ols_policy',
            user_name     => 'globalit_1',
            max_level     => 'S');
  SA_USER_ADMIN.SET_LEVELS (
       policy_name  => 'general_ols_policy',
       user_name    => 'globalit_1',
       max_level    => 'S',
       min_level    => 'R');
SA_USER_ADMIN.SET_GROUPS (
            policy_name     => 'general_ols_policy',
            user_name       => 'globalit_1',
            read_groups     => 'IT_GLOBAL');
END;

BEGIN
    SA_USER_ADMIN.SET_LEVELS (
            policy_name   => 'general_ols_policy',
            user_name     => 'hr_1',
            max_level     => 'S');

  SA_USER_ADMIN.SET_LEVELS (
       policy_name  => 'general_ols_policy',
       user_name    => 'hr_1',
       max_level    => 'S',
       min_level    => 'R');

SA_USER_ADMIN.SET_GROUPS (
            policy_name     => 'general_ols_policy',
            user_name       => 'hr_1',
            read_groups     => 'HR');
END;

BEGIN
    SA_USER_ADMIN.SET_LEVELS (
            policy_name   => 'general_ols_policy',
            user_name     => 'itsupport_1',
            max_level     => 'R');
    SA_USER_ADMIN.SET_GROUPS (
```

```
            policy_name     => 'general_ols_policy',
            user_name       => 'itsupport_1',
            read_groups     => 'IT_SUPPORT');
END;


BEGIN
   SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'general_ols_policy',
         user_name    => 'hr_1',
         max_level    => 'S',
         min_level    => 'R');

    SA_USER_ADMIN.SET_COMPARTMENTS (
         policy_name   => 'general_ols_policy',
         user_name     => 'hr_1',
         read_comps    => 'MAN,TEC,SAF,REP,STR,REV,BLU',
         write_comps   => 'MAN,TEC,SAF,REP,STR,REV,BLU');

   SA_USER_ADMIN.SET_GROUPS (
         policy_name    => 'general_ols_policy',
         user_name      => 'hr_1',
         read_groups    => 'HR_MGR');
END;

BEGIN
   SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'general_ols_policy',
         user_name    => 'hr_2',
         max_level    => 'S',
         min_level    => 'R'
         );

    SA_USER_ADMIN.SET_COMPARTMENTS (
         policy_name   => 'general_ols_policy',
         user_name     => 'hr_2',
         read_comps    => 'TEC,SAF,REP,STR,REV,BLU',
         write_comps   => 'TEC,SAF,REP,STR,REV,BLU');

   SA_USER_ADMIN.SET_GROUPS (
         policy_name    => 'general_ols_policy',
         user_name      => 'hr_2',
         read_groups    => 'HR_REC',
         row_groups     => 'HR_REC');
```

```
END;

BEGIN
   SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'general_ols_policy',
         user_name    => 'globalit_1',
         max_level    => 'S',
         min_level    => 'R');

    SA_USER_ADMIN.SET_COMPARTMENTS (
         policy_name   => 'general_ols_policy',
         user_name     => 'globalit_1',
         read_comps    => 'MAN,TEC,SAF,FIN,REP,BUD,STR,REV,BLU',
         write_comps   => 'MAN,TEC,SAF,FIN,REP,BUD,STR,REV,BLU');

   SA_USER_ADMIN.SET_GROUPS (
         policy_name    => 'general_ols_policy',
         user_name      => 'globalit_1',
         read_groups    => 'IT_GLOBAL_MGR',
         row_groups     => 'IT_GLOBAL_MGR');
END;

BEGIN
   SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'general_ols_policy',
         user_name    => 'globalit_2',
         max_level    => 'C',
         min_level    => 'R');

    SA_USER_ADMIN.SET_COMPARTMENTS (
         policy_name   => 'general_ols_policy',
         user_name     => 'globalit_2',
         read_comps    => 'TEC,SAF,REP,REV,BLU,STR',
         write_comps   => 'TEC,SAF,REP,REV,BLU,STR');

   SA_USER_ADMIN.SET_GROUPS (
         policy_name    => 'general_ols_policy',
         user_name      => 'globalit_2',
         read_groups    => 'IT_GLOBAL_ENG',
         row_groups     => 'IT_GLOBAL_ENG');
END;


BEGIN
```

```
   SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'general_ols_policy',
         user_name    => 'globalit_3',
         max_level    => 'C',
         min_level    => 'R',
         def_level    => 'R',
         row_level    => 'R'  );

    SA_USER_ADMIN.SET_COMPARTMENTS (
         policy_name   => 'general_ols_policy',
         user_name     => 'globalit_3',
         read_comps    => 'TEC,SAF,REP,REV,BLU,STR',
         write_comps   => 'TEC,SAF,REP,REV');

   SA_USER_ADMIN.SET_GROUPS (
         policy_name    => 'general_ols_policy',
         user_name      => 'globalit_3',
         read_groups    => 'IT_GLOBAL_TEC',
         row_groups     => 'IT_GLOBAL_TEC');
END;

BEGIN
   SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'general_ols_policy',
         user_name    => 'itsupport_2', --manager
         max_level    => 'S',
         min_level    => 'R'
         );

    SA_USER_ADMIN.SET_COMPARTMENTS (
         policy_name   => 'general_ols_policy',
         user_name     => 'itsupport_2',
         read_comps    => 'MAN,TEC,SAF,REP,REV,BLU,STR',
         write_comps   => 'MAN,TEC,SAF,REP,REV,BLU,STR');

   SA_USER_ADMIN.SET_GROUPS (
         policy_name    => 'general_ols_policy',
         user_name      => 'itsupport_2',
         read_groups    => 'IT_SUPPORT_MGR',
         row_groups     => 'IT_SUPPORT_MGR');
END;

BEGIN
   SA_USER_ADMIN.SET_LEVELS (
```

```
        policy_name  => 'general_ols_policy',
        user_name    => 'itsupport_1',
        max_level    => 'C',
        min_level    => 'R',
        def_level    => 'R',
        row_level    => 'R'
        );

    SA_USER_ADMIN.SET_COMPARTMENTS (
        policy_name   => 'general_ols_policy',
        user_name     => 'itsupport_1',
        read_comps    => 'TEC,SAF,REP,REV,BLU,STR',
        write_comps   => 'TEC,SAF,REP,REV');

   SA_USER_ADMIN.SET_GROUPS (
        policy_name    => 'general_ols_policy',
        user_name      => 'itsupport_1',
        read_groups    => 'IT_SUPPORT_TEC',
        row_groups     => 'IT_SUPPORT_TEC');
END;
```

OLS Policy on PROJECTS Table

```
-- Apply OLS policy to PROJECTS table
BEGIN
    SA_POLICY_ADMIN.APPLY_TABLE_POLICY (
            policy_name    => 'general_ols_policy',
            schema_name    => 'company',
            table_name     => 'projects',
            table_options  => 'read_control, write_control, label_default');
END;

-- Enable OLS policy on PROJECTS table
BEGIN
    SA_POLICY_ADMIN.ENABLE_TABLE_POLICY (
            policy_name => 'general_ols_policy',
            schema_name => 'company',
            table_name  => 'projects');
END;
```

For EmployeeTrainingRecords, i only put security level

VPD on Employees Table

OLS Policy on Employees Table

```
BEGIN
  SA_POLICY_ADMIN.APPLY_TABLE_POLICY (
    policy_name    => 'general_ols_policy',
    schema_name    => 'COMPANY',
    table_name     => 'EMPLOYEES',
    table_options  => 'READ_CONTROL,WRITE_CONTROL');
END;

BEGIN
   SA_POLICY_ADMIN.ENABLE_TABLE_POLICY (
      policy_name => 'general_ols_policy',
      schema_name => 'COMPANY',
      table_name  => 'EMPLOYEES');
END;


----------add data label to general_ols_co of Employees Table
UPDATE COMPANY.EMPLOYEES
SET    general_ols_col = CHAR_TO_LABEL('general_ols_policy','R::HR_REC')
WHERE  UPPER(employeeid) IN (10001, 10004, 10007);

UPDATE COMPANY.EMPLOYEES
SET    general_ols_col =
CHAR_TO_LABEL('general_ols_policy','R::HR_REC,IT_GLOBAL_MGR')
WHERE  employeeid = 10002;

UPDATE COMPANY.EMPLOYEES
SET    general_ols_col =
CHAR_TO_LABEL('general_ols_policy','R::HR_REC,IT_SUPPORT_MGR')
WHERE  employeeid = 10003;

UPDATE COMPANY.EMPLOYEES
SET    general_ols_col =
CHAR_TO_LABEL('general_ols_policy','R::HR_REC,IT_GLOBAL_ENG')
WHERE  UPPER(employeeid) IN (10005, 10008, 10010);

UPDATE COMPANY.EMPLOYEES
```

```
SET    general_ols_col =
CHAR_TO_LABEL('general_ols_policy','R::HR_REC,IT_SUPPORT_TEC')
WHERE  UPPER(employeeid) IN (10006, 10009);

UPDATE COMPANY.EMPLOYEES
SET    general_ols_col =
CHAR_TO_LABEL('general_ols_policy','R::HR_REC,IT_GLOBAL_TEC')
WHERE  employeeid = 10010;
```

–Test data access via user hr_1, *Expecting to see all data in Employees Table since user is HR mgr*

```
SELECT FIRSTNAME, LASTNAME, EMPLOYEEID,
LABEL_TO_CHAR(general_ols_col) OLS_LABEL
FROM COMPANY.EMPLOYEES
ORDER BY general_ols_col;
```

–Test data update via user hr_1 *Expecting able to update all data in Employees Table since user is HR*

```
UPDATE COMPANY.EMPLOYEES
SET    LASTNAME = 'Joe'
WHERE  UPPER(employeeid) IN (10001);
```

```
UPDATE COMPANY.EMPLOYEES
SET    LASTNAME = 'Lee'
WHERE  UPPER(employeeid) IN (10009);
```

–Test data access via user `'globalit_1'--`*Expecting to see employees of IT_GLOBAL and IT_SUPPORT in Employees Table since user is IT_GLOBAL manager*

```
SELECT FIRSTNAME, LASTNAME, EMPLOYEEID,
LABEL_TO_CHAR(general_ols_col) OLS_LABEL
FROM COMPANY.EMPLOYEES
ORDER BY general_ols_col;
```

–Test data update via user `'globalit_1'` *–Expecting able to update all data of IT_GLOBAL and IT_SUPPORT in Employees Table since user is IT_GLOBAL manager*

```
UPDATE COMPANY.EMPLOYEES
SET    LASTNAME = 'JOE'
WHERE  UPPER(employeeid) IN (10001); –fail

UPDATE COMPANY.EMPLOYEES
SET    LASTNAME = 'Smith'
WHERE  UPPER(employeeid) IN (10002);
```

–Test data access via user `'itsupport_1'--`*Expecting only can access self info in Employees Table since user is IT_SUPPORT is employee*

```
SELECT * FROM COMPANY.EMPLOYEES;
```

OLS Policy on Meeting Notes Table

```
BEGIN
   SA_POLICY_ADMIN.APPLY_TABLE_POLICY (
         policy_name    => 'general_ols_policy',
         schema_name    => 'company',
         table_name     => 'meetingnotes',
         column_name    => 'SecurityLevel', -- Ensure this is the correct column for OLS labels
         table_options  => 'read_control, write_control'); -- Specifying both read and write controls
END;
/

-- Enable OLS policy on MEETINGNOTES table
BEGIN
   SA_POLICY_ADMIN.ENABLE_TABLE_POLICY (
         policy_name => 'general_ols_policy',
         schema_name => 'company',
         table_name  => 'meetingnotes');
END;
/


UPDATE company.meetingnotes
SET general_ols_col =
    CASE
```

```
        WHEN SecurityLevel = 'Confidential' AND Department = 'Global IT' THEN
1000000001
        WHEN SecurityLevel = 'Secret' AND Department = 'HR' THEN 1000000004
        WHEN SecurityLevel = 'Restricted' AND Department = 'IT Support' THEN
1000000003
        -- Add other mappings as necessary.
        ELSE NULL -- Default or error case.
    END;
```

Testing of OLS Policy on MEETINGNOTES:

SELECT * FROM COMPANY.MEETINGNOTES (in each of the 3 different users GLOBALIT, ITSUPPORT, HR to see the access differences)

Test case 1 for updating (Expected result: GlobalIT able to update, HR and IT_Support unable to)

```
UPDATE COMPANY.MEETINGNOTES
SET    MeetingType = 'Strategy' (can use ‘Review’ also)
WHERE  UPPER(noteid) IN (61001);
```

Test case 2 for write-up (Expected result: Only GlobalIT able to insert, HR and IT_Support unable to)

```
INSERT INTO Company.MeetingNotes (NoteID, MeetingDate, Topic, Content,
SecurityLevel, Department, MeetingType)
VALUES (61006, TO_DATE('2023-07-30', 'YYYY-MM-DD'), 'New Project Kick-off 2',
'Second Content for new project kick-off...', 'Confidential', 'Global IT',
'Strategy');
```

Test case 3 for write-up (Expected result: Only HR able to insert, GlobalIT and IT_Support unable to)

```
INSERT INTO Company.MeetingNotes (NoteID, MeetingDate, Topic, Content,
SecurityLevel, Department, MeetingType)
VALUES (61007, TO_DATE('2023-01-15', 'YYYY-MM-DD'), 'HR Policies Update 2', '
Second Content for HR policy update...', 'Secret', 'HR', 'Strategy');
```

```
BEGIN
    SA_POLICY_ADMIN.DISABLE_TABLE_POLICY(
```

```
        policy_name => 'general_ols_policy',
        schema_name => 'company',
        table_name  => 'meetingnotes'
    );
END;
/
```

OLS Policy on EMPLOYEEBENEFITS Table

```
-- Apply OLS policy to employeebenefits table
BEGIN
    SA_POLICY_ADMIN.APPLY_TABLE_POLICY (
            policy_name    => 'general_ols_policy',
            schema_name    => 'company',
            table_name     => 'employeebenefits',
            table_options  => 'read_control, write_control, label_default');
END;

-- Enable or Disable OLS policy on employeebenefits table
BEGIN
    SA_POLICY_ADMIN.ENABLE_TABLE_POLICY (
            policy_name => 'general_ols_policy',
            schema_name => 'company',
            table_name  => 'employeebenefits');
END;

BEGIN
    SA_POLICY_ADMIN.DISABLE_TABLE_POLICY(
        policy_name => 'general_ols_policy',
        schema_name => 'company',
        table_name  => 'employeebenefits'
    );
END;

— Set conditions for each row of data based on the group and compartment
UPDATE company.employeebenefits
SET general_ols_col =
    CASE
        WHEN SecurityLevel = 'Confidential' AND Department = 'Global IT' THEN
1000000001
        WHEN SecurityLevel = 'Secret' AND Department = 'HR' THEN 1000000004
        WHEN SecurityLevel = 'Restricted' AND Department = 'IT Support' THEN
1000000003
        -- Add other mappings as necessary.
```

```
        ELSE NULL -- Default or error case.
    END;
UPDATE Company.EmployeeBenefits eb
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'R')
WHERE eb.SecurityLevel = 'Restricted';

UPDATE Company.EmployeeBenefits eb
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'S::HR')
WHERE eb.SecurityLevel = 'Secret';

UPDATE Company.EmployeeBenefits eb
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'C::IT_GLOBAL')
WHERE eb.SecurityLevel = 'Confidential' and DEPARTMENT = 'Global IT';

UPDATE Company.EmployeeBenefits eb
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'C:FIN:HR')
WHERE eb.SecurityLevel = 'Confidential' and DEPARTMENT = 'HR';

UPDATE Company.EmployeeBenefits eb
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'C:RET:HR,
zIT_SUPPORT')
WHERE eb.SecurityLevel = 'Confidential' and DEPARTMENT = 'IT Support';




UPDATE projectdocumentation
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy',
'C:TEC,REP:IT_GLOBAL')
WHERE docid = 40001;
UPDATE projectdocumentation
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'S:STR,BLU:HR')
WHERE docid = 40002;
UPDATE projectdocumentation
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy',
'R:FIN,BUD:IT_GLOBAL')
WHERE docid = 40003;
UPDATE projectdocumentation
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy',
'C:TEC,REP:IT_SUPPORT')
WHERE docid = 40004;
UPDATE projectdocumentation
```

```
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'S:STR,BLU:HR')
WHERE docid = 40005;
UPDATE projectdocumentation
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy',
'R:TEC,REP:IT_GLOBAL')
WHERE docid = 40006;
UPDATE projectdocumentation
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'C:STR,REP:HR')
WHERE docid = 40007;
UPDATE projectdocumentation
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy',
'S:TEC,BLU:IT_GLOBAL')
WHERE docid = 40008;
UPDATE projectdocumentation
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy',
'R:TEC,REP:IT_SUPPORT')
WHERE docid = 40009;
UPDATE projectdocumentation
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy',
'C:FIN,BUD:IT_GLOBAL')
WHERE docid = 40010;

UPDATE employeetrainingrecords
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'R')
WHERE trainingrecordid = 20001;
UPDATE employeetrainingrecords
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'C')
WHERE trainingrecordid = 20002;
UPDATE employeetrainingrecords
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'S')
WHERE trainingrecordid = 20003;
UPDATE employeetrainingrecords
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'R')
WHERE trainingrecordid = 20004;
UPDATE employeetrainingrecords
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'C')
WHERE trainingrecordid = 20005;
UPDATE employeetrainingrecords
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'S')
WHERE trainingrecordid = 20006;
UPDATE employeetrainingrecords
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'R')
WHERE trainingrecordid = 20007;
```

```
UPDATE employeetrainingrecords
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'C')
WHERE trainingrecordid = 20008;
UPDATE employeetrainingrecords
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'S')
WHERE trainingrecordid = 20009;
UPDATE employeetrainingrecords
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'R')
WHERE trainingrecordid = 20010;
```

OLS for EmployeeTrainingRecords (Requirement)

Employees cannot view training record of Mangers
Employees only can view training record of his/her own department and same position (Eng -Eng, Eng-Technician, Technician -Technician), but cannot view training record of Mangerial (S)
Same training type, ie technical, Engineer with C level, Technician with R level
Managers only can view his/her own department training record
HR Manager can view training record of everyone
HR recruiter can view training record of everyone except mgr’s
Data access control via L, C, 2Goups

Account	**Roles**
ITSUPPORT_1	EMPLOYEE
GLOBALIT_1	MANAGER
HR_1	MANAGER
HR_2	EMPLOYEE
GLOBALIT_2	EMPLOYEE(Engineer)
ITSUPPORT_2	MANAGER
GLOBALIT_3	EMPLOYEE (Technician)

What	Code

Group 15

CS5322 Project 2 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

To thrive in a competitive market, modern businesses often turn to Enterprise Resource Planning Systems (ERP) as indispensable tools for efficient and comprehensive management. These systems automate and streamline processes in big organisations which often involve multiple parties by integrating the various functions of an organisation through an unified system. While the benefits of such integrations are often talked about, it is also important to address new challenges in the area data privacy – such system has to ensure that restrictions are finely tuned so that employees have just the right amount of access to perform their tasks, but not too rigid such that granting additional access is made impossible or unreasonably difficult. Moreover, it is paramount that companies adhere to secure data security principles so as to protect company information and secrets from unauthorised access and manipulation.

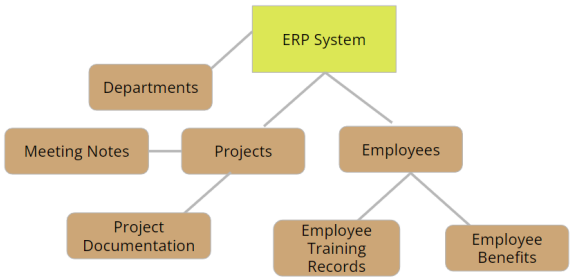

Figure 1.1: ERP Prototype System Functionality Illustration

Taking into account such nuanced data access requirements in ERP systems, the team has decided to recreate a simple ERP system database (as depicted in Fig 1) with Oracle database which comes with a multilevel security feature out of the box Oracle Label Security (OLS). The aim of the project is to demonstrate how multilevel security through OLS can be used to help secure company data by providing granular yet flexible access control in settings where data is shared across multiple users physically but access to part of the data should be restricted logically to a set of users.

Application

This ERP system database prototype is designed to serve company employees, mainly Employees from the HR and IT departments. The ERP is capable of storing and retrieving employee details, including their training records and employee benefits, department information, company projects and as well as meeting notes. The ERP system aims to serve employees and managers alike in a multitude of cases.

To simulate company onboarding, a new user will be tagged with their respective department and security level when receiving their account.

Employees who use the ERP system, be it from HR or IT departments, should be able to use the system to get and update information about their details as well as let the employee know what company benefits they are entitled to. Moreover, the ERP system aims to help them track their professional progression by giving them more details of their completed and pending training programs. Employees should also be able to view related meeting notes.

Our ERP system aims to allow managers of the departments to also streamline their work by managing their team and projects more effectively. This can come in the form of viewing information of their team such as their training records and details, and also manage morale with the availability of employee benefits to each tier of employees. Moreover, managers can better track the progress of their projects through the ERP system as well.

OLS Implementation

The OLS implemented in our ERP system database is anchored by a policy container that ensures read and write controls are enforced on the 'general_ols_col' column, which is a part of each table within the ERP system database when the OLS policy created is applied to the table.

Firstly, our ERP system defines three principal security labels to classify the sensitivity of our data objects. The lowest sensitivity label is 'RESTRICTED' with a level number of 100, 'CONFIDENTIAL' at level 200, and the highest sensitivity is 'SECRET' at level 300. The security levels assigned help to prevent “read up” as users with a lower security level such as 'RESTRICTED' would not be able to read data tagged with a higher security level such as 'CONFIDENTIAL' or 'SECRET', as seen in Figure 3.1.

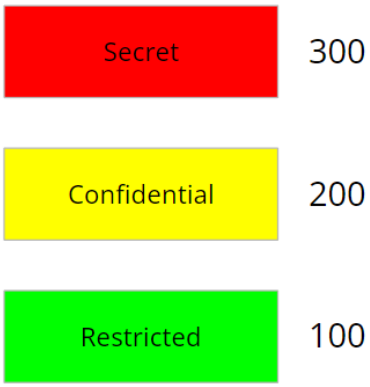

Figure 3.1: Security Levels

Next, to allow for more refined data access within each security label, a range of compartments were introduced. This helps to prevent employees with higher security levels from viewing data that is not related to their role. The compartments correspond with the various data domains within the organisation. This includes: 'MEDICAL', 'FINANCIAL', 'RETIREMENT', 'TECHNICAL', 'MANAGERIAL', 'SAFETY', 'REPORT', 'BLUEPRINT', 'BUDGET', 'STRATEGY', and 'REVIEW'. Each of the compartments are assigned unique numbers from 100 to 304, enabling granular control over the different sensitive information.

Moreover, our ERP system can help enforce organisational roles and functions with regards to data access. This was done by creating OLS groups which allow for a hierarchical structure.

One OLS group implemented in the ERP system database was created to reflect departments in the organisation. There are two main departments in this organization - Human Resource (HR) and Technology (IT). IT is split to one parent and one child department, which are IT Global (Parent) and IT Support (Child).

The department labels start with COMPANY, which represents the executives in an organisation which have access to data belonging to all departments. This is followed by the management level of two main departments 'HUMAN_RESOURCE_MANAGEMENT' and 'TECHNOLOGY_GLOBAL_MANAGEMENT' to represent the department managers role. Next is either the employees roles of each division themselves, as seen in the case of HR where the following label is HUMAN_RESOURCE_RECRUITERS, or sub-divisions within a department as seen in IT where the following labels are TECHNOLOGY_SUPPORT_MANAGEMENT and TECHNOLOGY_GLOBAL_ENGINEERS to represent the middle managers in each sub-division. In IT, this is finally followed by the employee groups TECHNOLOGY_SUPPORT_TECHNICIAN and TECHNOLOGY_GLOBAL_TECHNICIAN respectively.

Each of the groups is identified by their own unique group numbers and some groups are affiliated through a parent-child relationship. This label-based access is able to reflect the departmental hierarchy and organisational hierarchy to better manage user access to the ERP, as seen in Figure 3.2 which shows the group hierarchy for parent-child relationship.

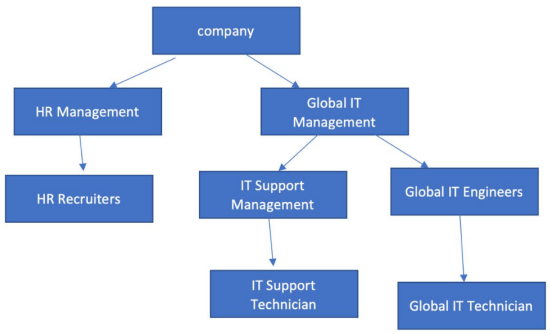

Figure 3.2 Group hierarchy

Lastly, the OLS policy also extends to the individual database user accounts and not just the data objects. Minimum and maximum security levels based on the job roles have been efficiently assigned for all users to ensure that their access to data is controlled. A default session level has also been assigned for each user as well.

For example, user 'globalit_1' is assigned a maximum security level of 'SECRET' and is associated with the 'TECHNOLOGY_GLOBAL_MANAGEMENT' group, while 'hr_1' also retains a 'SECRET' level but is linked to 'HUMAN_RESOURCE_MANAGEMENT'. This shows the approach to role-based access control based on the granularity of the users.

Moreover, we have added 'label_default' under the default_options parameter of the policy. When a user inserts data into any table, the data object will be tagged with the same OLS labels as what the user will have in the current session. This is to help prevent a “write down” situation where a user with 'SECRET' level privileges inserts data with only a 'RESTRICTED' label.

The SQL code for the above OLS setup can be found in Appendix A of this report.

Overall, the OLS implementation in our ERP system database focuses on preventing illegal information flow.

Security Policies

Other than the implementation of OLS, other access control strategies have been implemented to support access control of the ERP system. These additional access control measures include Discretionary Access Control (DAC) and Virtual Private Database (VPD). The DAC implementation is structured around the two primary user roles within the ERP system, which are namely employee and manager. The employee roles include: HUMAN_RESOURCE_RECRUITERS, TECHNOLOGY_GLOBAL_ENGINEERS, TECHNOLOGY_SUPPORT_TECHNICIAN or TECHNOLOGY_GLOBAL_TECHNICIAN. These roles that are defined determine the level of access to each of the database tables. VPD on the other hand is implemented for certain tables within the ERP system where users should only be able to access their own data.

Discretionary Access Control

Firstly, an employee has been granted 'SELECT' privileges on tables such as PROJECTS, PROJECTDOCUMENTATION, MEETINGNOTES, EMPLOYEES, EMPLOYEEBENEFITS and EMPLOYEETRAININGRECORDS. By granting the employees’ access rights in this manner allows relevant users assigned the employee role to view and access data across a range of operations within the ERP. This facilitates sufficient privileges for employees to obtain the necessary information to perform their job functions.

In addition to read permissions, the employee role is also granted 'INSERT' and 'UPDATE' rights on some of the tables mentioned previously. The limited write access is also granted to enable employees to contribute to the relevant tables of documentation and maintain current records within their job scope.

In order to provide more privileged access, the DAC is specially designated to the manager (parent) roles. This also includes all the 'SELECT' privileges of the employee (child) roles. This provision of modified access to this table for managers allows them to view training records of their subordinates, which is essential for overseeing their development and performance within the organisation. Furthermore, users with similar levels to others can also view the training records of their peers, to increase awareness. This is further explained in the Database section below.

Furthermore, the manager role has also been granted 'INSERT' and 'UPDATE' permissions on all tables. This additional DAC takes into account the performance of managerial duties

which involve updating project statuses, providing edits to meeting documentation and the management of employee records.

A summary of the DAC policies can be found in table 1 below.

To further illustrate the various role-based permissions in practice, organisational user accounts have been created, as illustrated in the table below.

Account	**Roles**
ITSUPPORT_1	EMPLOYEE
GLOBALIT_1	MANAGER
HR_1	MANAGER
HR_2	EMPLOYEE
GLOBALIT_2	EMPLOYEE (Engineer)
ITSUPPORT_2	MANAGER
GLOBALIT_3	EMPLOYEE (Technician)

One user to highlight would be 'ITSUPPORT_1' who has been assigned an employee role. This will showcase the read and limited write permissions for the ERP tables. Similarly, 'GLOBALIT_1', 'HR_1' and ITSUPPORT_2 have been created and assigned a manager role. The hierarchical roles reflect the requirement for broader data access within ERP databases, which also includes the Engineers and Technicians.

The DAC implementation has been carried out to ensure that it aligns with the principles of least privilege and separation of organisational duties This ensures that users of the ERP system are only given the level of access that is necessary for their job functions. Overall, DAC implementation in our ERP system helps to quickly govern if users are allowed to access data based on their role and table which they are accessing data from.

Table	**Database Roles**	
	MANAGER	*EMPLOYEE*
projects	S, I, U	S
project_documentation	S, I, U	S, I, U
meeting_notes	S, I, U	S, I, U
employee_training_records	S, I, U	S
employee_benefits	S, I, U	S
employee	S, I, U	S, I, U
S = SELECT, U = UPDATE, I = INSERT		

Table 1: DAC policies applied to each table

Virtual Private Database

The ERP system that is implemented with OLS enforces fine-grained access control, ensuring data security and compliance with the organisational security policies. VPD has also been implemented to complement the OLS by dynamically appending predicates to SQL statements based on the security policies which are linked to the user properties. These user properties include the 1) **username** which are identified in both the EMPLOYEES table and the database account, retrievable via USERENV namespace using the SESSION_USER parameter 2) the user's **position** stored in the POSITION column in the EMPLOYEES table and accessible through a custom context. The position property is automatically loaded into the context using database logon triggers.

The layer approach to security incorporating OLS, VPD and DAC implementation ensures that the ERP database adheres to the principle of least privilege and separation of job-related duties minimising the risk of illegal information flow.

Database Tables

With the OLS labels described above implemented in our ERP system, this section will be devoted to showcasing how the multitude of security access controls will affect the key tables in our ERP system database. In all of our tables, the security level, compartments and groups have their own column for the ease of explanation. In a realistic deployment, not all the information will be shown.

The EMPLOYEES table stores personal information about the employees and is associated with each employee. This includes information such as their names, contact information, security level, position and department. The position and department columns denote the employee's role and organisational group. This is aligned with the implemented OLS which can be used to control access to this table based on the security level, position and department of the user. Since all tuples are coded with 'RESTRICTED' label, this would mean that any user will be able to view all employee information. This would also not be solved if we changed the security levels of the employee data as a 'SECRET' level user would then be able to view all employee data. Hence, a VPD implementation (in Appendix C) is required for more fine-grained access control. Our VPD ensures that normal IT employees will only be able to view their own employee records while IT managers would be only able to view only the information of employees in their department. Meanwhile, HR employees would still be able to view all the employee data for their job. Utilising a VPD here allows us to provide more intricate access control on the EMPLOYEES table, giving different types of users different types of accesses based on their job requirements.

Next, the DEPARTMENTS table stores information regarding the different departments that are part of the company.

Following which, the PROJECTS table stores information about the project ids, project names, start date and end date to help determine which projects are ongoing and which projects that have been undertaken by the company have been completed. The OLS can be used to determine user access based on their department and security level of the project. OLS labels used in this table primarily revolve around security labels and group labels to represent the sensitivity of each project and the department undertaking the projects. Combined with our implemented DAC, employees in each department should not be able to view projects that are undertaken by other departments, and that employees can only view projects that are equal or lower than their security clearance in the organisation. Managers on the other hand are able to view, add and update projects which are related to their department and for their own security level as well.

Moreover, the PROJECTDOCUMENTATION table stores the documentation of the projects that have been undertaken by the company. This closely follows the PROJECTS table by including a foreign key relation to the PROJECTID column in the PROJECTS table, ensuring that a project document must be tied to an existing project. Here a project document will contain its security label, department group, as well as the compartments which this document falls into. Here, some compartment categories are its document type such as 'FINANCIAL', 'TECHNICAL' and 'STRATEGY'. Another compartment is determined by the datatype of the document as well such as 'REPORT', 'BLUEPRINT' and 'BUDGET'. Along with the security label and department, the OLS label will determine the user access to this table.

The EMPLOYEEBENEFITS table stores information regarding the generic employee benefits that are made available to the employees throughout the company, depending on their department, employee level (based on their security access), benefit type, benefit amount and benefit form. The OLS enables the HR department to have wider access in order to make changes to the benefits available, based on their individual groups and compartments as defined.

The MEETINGNOTES table stores information of meetings that have been held in the company. Here a meeting will contain its security label, department group, as well as the meeting type compartment. The compartment of either 'REVIEW' or 'STRATEGY' enables more fine-grained access control to ensure that users are able to only see meeting notes which correspond to their security label, department hierarchy and compartment as well.

Lastly, for the EMPLOYEETRAININGRECORDS table, it stores information regarding the type of training and the training certification(s) obtained by an employee. An employee is able to view their own training record if it is within their security clearance level. This is intended as there might be training which the employee might have been given and passed which they should not know about. Because our detailed and robust OLS group combines both departments and organisation hierarchy, we are able to enable flexible access controls to fit our ERP system requirements. A user from IT Global can view training records of another user from IT Support while any user from the HR department would be able to view the records of users from other departments.
The inbuilt organisational hierarchy allows managers to view employee training records from their department while employees are unable to view the training records of managers. Moreover, since the 'ENGINEER' role is a parent to the 'TECHNICIAN' role, this allows an Engineer to view the training records of other Engineers and Technicians as well. Importantly, we intentionally allowed users who have similar roles and security labels to view each other's training records as a form of motivation.

Additionally, the OLS compartments implemented for Employee Training Records are 'TECHNICAL', 'MANAGERIAL' and 'SAFETY'. Similarly, this helps with implementing fine-grained access control. Only an engineer with a 'SAFETY' label would then be able to view records containing the 'SAFETY' compartment.

Overall, utilising OLS labels as well as other access control policies differently for the different tables allow for varied use cases which our ERP system database requires.

Results & Discussion

Firstly, the application of OLS has effectively prevented unauthorised access and any potential security breaches to the ERP system. This reinforces the robustness of the system in protecting sensitive organisation data being exchanged at a high rate throughout the work day.

However, we understand that further improvements to the table design is required to make this a fully usable ERP solution for small enterprises, as it lacks several tables such as matching each employee to their entitled employee benefits, for example. This is not included as it is beyond the scope of this project to demonstrate the effectiveness of OLS policies on the discretionary access controls.

Performance wise, the implementation of the OLS did not impede the efficiency of the ERP system. Despite the variations in query execution times, the differences were not substantial in illustrating the degradation of performance over time. This further highlights the capability of OLS in balancing between providing a high level of security and operational efficiency.

Next, there is a need to consider the scalability of the ERP system as the organisation continues to grow. The ERP system will become increasingly complex with more tables being incorporated, which requires the system continuously adapt. There will be the need to modify the OLS implementation in order to address the evolving organisational structure and new security risks.

Also, the integration of OLS has enabled organisational security to be achieved. With the application of OLS, detailed security labels and

role-based access control has been crucial in managing user access and the exchange of information across the organisation. This has been implemented in accordance with the principles and least privilege and separation of duties. The OLS approach adopted minimises the risks of illegal information flow by preventing users of a lower security level from reading data from a higher security level. Moreover, our OLS implementation can also prevent users of high security level to bypass access control by reading, for example, a 'SECRET' document then "writing down" as data with a lower security level. This can be done to ensure the integrity and confidentiality of the data.

While the currently implemented OLS, DAC and VPD provide robust access control, the ERP could potentially integrate Dynamic Data Masking (DDM) to further enhance data privacy. DDM is able to dynamically obscure specific data within a database. This ensures that the sensitive data is not exposed to unauthorised users when they access the database. This is particularly useful for scenarios where employees need to access databases containing sensitive information but visibility of all data is not required. For example, employees in HR are required to access personal details of staff but are not required to see their salary information. The potential implementation of DDM can mask these sensitive details dynamically as determined by the user roles and access levels within the ERP.

Lastly, the multiple implementation of OLS, DAC and VPD, adds a layer of complexity in managing security policies for the administrator. The implementation of multiple security policies requires rigorous planning and regular updates to ensure policies remain relevant. Especially for organisations with ever evolving structure and data access needs based on its growth. Furthermore, the additional layer of complexity will necessitate extensive training for database administrators and end users to better understand how to best utilise the ERP system. This could be resource-intensive, which will be incurred by the company.

Conclusion

This project has demonstrated the effective implementation of OLS in an ERP system, which has been tailored to meet the various data access and privacy needs. The ERP system encompasses multiple departments and different user roles to illustrate a robust framework for managing sensitive data through a multi-layered security approach. By integrating OLS, DAC, and VPD, the established ERP system adheres to stringent data security standards and offers flexibility and scalability required by the company.

Next, the implementation of tiered security labels, refined data access compartments, and role-based access control have also effectively addressed the complex requirements of data confidentiality and integrity. This enables the ERP system to share sensitive data securely across different organisational levels and users. The implementation also highlights the importance of a balanced approach to security and operational efficiency, ensuring that enhanced data protection does not impede system performance.

In conclusion, this project highlights the capability of OLS in ensuring the data security within ERP systems. This project potentially serves as an example for organisations to develop a scalable model aimed at protecting sensitive data while maintaining high operational efficiency. This project also illustrates the contributions to data security in ERP systems, offering a practical and effective solution to modern-day security challenges.

Appendix A: OLS Setup

```
-- Create policy container
BEGIN
    SA_SYSDBA.CREATE_POLICY (
            policy_name      => 'general_ols_policy',
            column_name      => 'general_ols_col',
            default_options  => 'read_control, write_control,
label_default');
END;

-- Create security policy labels
BEGIN
    SA_COMPONENTS.CREATE_LEVEL (
            policy_name => 'general_ols_policy',
            level_num   => 100,
            short_name  => 'R',
            long_name   => 'RESTRICTED');

    SA_COMPONENTS.CREATE_LEVEL (
            policy_name => 'general_ols_policy',
            level_num   => 200,
            short_name  => 'C',
            long_name   => 'CONFIDENTIAL');
    SA_COMPONENTS.CREATE_LEVEL (
            policy_name => 'general_ols_policy',
            level_num   => 300,
            short_name  => 'S',
            long_name   => 'SECRET');
END;

-- Create compartments
BEGIN
    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'MEDICAL',
            short_name       => 'MED',
            comp_num         =>  100);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
```

```
            long_name        => 'FINANCIAL',
            short_name       => 'FIN',
            comp_num         =>  101);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'RETIREMENT',
            short_name       => 'RET',
            comp_num         =>  102);
END;

BEGIN
    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'TECHNICAL',
            short_name       => 'TEC',
            comp_num         =>  200);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'MANAGERIAL',
            short_name       => 'MAN',
            comp_num         =>  201);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'SAFETY',
            short_name       => 'SAF',
            comp_num         =>  202);
END;

BEGIN
    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'REPORT',
            short_name       => 'REP',
            comp_num         =>  300);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'BLUEPRINT',
            short_name       => 'BLU',
            comp_num         =>  301);
```

```
    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'BUDGET',
            short_name       => 'BUD',
            comp_num         =>  302);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'STRATEGY',
            short_name       => 'STR',
            comp_num         =>  303);

    SA_COMPONENTS.CREATE_COMPARTMENT (
            policy_name      => 'general_ols_policy',
            long_name        => 'REVIEW',
            short_name       => 'REV',
            comp_num         =>  304);
END;

-- Create Groups
BEGIN
    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 4000,
            short_name      => 'COMPANY',
            long_name       => 'COMPANY');

    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 4100,
            short_name      => 'HR_MGR',
            long_name       => 'HUMAN_RESOURCE_MANAGEMENT',
            parent_name     => 'COMPANY'
            );

    SA_COMPONENTS.CREATE_GROUP (
            policy_name     => 'general_ols_policy',
            group_num       => 4110,
            short_name      => 'HR_REC',
            long_name       => 'HUMAN_RESOURCE_RECRUITERS',
            parent_name     => 'HR_MGR'
        );
```

```
SA_COMPONENTS.CREATE_GROUP (
        policy_name     => 'general_ols_policy',
        group_num       => 4200,
        short_name      => 'IT_GLOBAL_MGR',
        long_name       => 'TECHNOLOGY_GLOBAL_MANAGEMENT',
        parent_name     => 'COMPANY'
    );


SA_COMPONENTS.CREATE_GROUP (
        policy_name     => 'general_ols_policy',
        group_num       => 4210,
        short_name      => 'IT_SUPPORT_MGR',
        long_name       => 'TECHNOLOGY_SUPPORT_MANAGEMENT',
        parent_name     => 'IT_GLOBAL_MGR'
    );

SA_COMPONENTS.CREATE_GROUP (
        policy_name     => 'general_ols_policy',
        group_num       => 4211,
        short_name      => 'IT_SUPPORT_TEC',
        long_name       => 'TECHNOLOGY_SUPPORT_TECHNICIANS',
        parent_name     => 'IT_SUPPORT_MGR'
    );

SA_COMPONENTS.CREATE_GROUP (
        policy_name     => 'general_ols_policy',
        group_num       => 4220,
        short_name      => 'IT_GLOBAL_ENG',
        long_name       => 'TECHNOLOGY_GLOBAL_ENGINEERS',
        parent_name     => 'IT_GLOBAL_MGR'
    );


SA_COMPONENTS.CREATE_GROUP (
        policy_name     => 'general_ols_policy',
        group_num       => 4221,
        short_name      => 'IT_GLOBAL_TEC',
        long_name       => 'TECHNOLOGY_GLOBAL_TECHNICIANS',
        parent_name     => 'IT_GLOBAL_ENG'
    );
```

```
END;

--Authorize Users for the Label Security Policy

BEGIN
   SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'general_ols_policy',
         user_name    => 'hr_1',
         max_level    => 'S',
         min_level    => 'R');

    SA_USER_ADMIN.SET_COMPARTMENTS (
         policy_name   => 'general_ols_policy',
         user_name     => 'hr_1',
         read_comps    => 'MAN,TEC,SAF,REP,STR,REV,BLU,RET,MED,FIN',
         write_comps   => 'MAN,TEC,SAF,REP,STR,REV,BLU,RET,MED,FIN');

   SA_USER_ADMIN.SET_GROUPS (
         policy_name    => 'general_ols_policy',
         user_name      => 'hr_1',
         read_groups    => 'HR_MGR');
END;

BEGIN
   SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'general_ols_policy',
         user_name    => 'hr_2',
         max_level    => 'S',
         min_level    => 'R'
         );

    SA_USER_ADMIN.SET_COMPARTMENTS (
         policy_name   => 'general_ols_policy',
         user_name     => 'hr_2',
         read_comps    => 'TEC,SAF,REP,STR,REV,BLU,RET,MED,FIN',
         write_comps   => 'TEC,SAF,REP,STR,REV,BLU,RET,MED,FIN');

   SA_USER_ADMIN.SET_GROUPS (
         policy_name    => 'general_ols_policy',
         user_name      => 'hr_2',
         read_groups    => 'HR_REC',
         row_groups     => 'HR_REC');
END;
```

```
BEGIN
   SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'general_ols_policy',
         user_name    => 'globalit_1',
         max_level    => 'S',
         min_level    => 'R');

    SA_USER_ADMIN.SET_COMPARTMENTS (
         policy_name   => 'general_ols_policy',
         user_name     => 'globalit_1',
         read_comps    => 'MAN,TEC,SAF,FIN,REP,BUD,STR,REV,BLU,RET,MED',
         write_comps   => 'MAN,TEC,SAF,FIN,REP,BUD,STR,REV,BLU,RET,MED');

   SA_USER_ADMIN.SET_GROUPS (
         policy_name    => 'general_ols_policy',
         user_name      => 'globalit_1',
         read_groups    => 'IT_GLOBAL_MGR',
         row_groups     => 'IT_GLOBAL_MGR');
END;

BEGIN
   SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'general_ols_policy',
         user_name    => 'globalit_2',
         max_level    => 'C',
         min_level    => 'R');

    SA_USER_ADMIN.SET_COMPARTMENTS (
         policy_name   => 'general_ols_policy',
         user_name     => 'globalit_2',
         read_comps    => 'TEC,SAF,REP,REV,BLU,STR,RET,MED',
         write_comps   => 'TEC,SAF,REP,REV,BLU,STR');

   SA_USER_ADMIN.SET_GROUPS (
         policy_name    => 'general_ols_policy',
         user_name      => 'globalit_2',
         read_groups    => 'IT_GLOBAL_ENG',
         row_groups     => 'IT_GLOBAL_ENG');
END;


BEGIN
   SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'general_ols_policy',
```

```
        user_name    => 'globalit_3',
        max_level    => 'C',
        min_level    => 'R',
        def_level    => 'R',
        row_level    => 'R'  );

    SA_USER_ADMIN.SET_COMPARTMENTS (
        policy_name   => 'general_ols_policy',
        user_name     => 'globalit_3',
        read_comps    => 'TEC,SAF,REP,REV,BLU,STR,RET,MED',
        write_comps   => 'TEC,SAF,REP,REV');

   SA_USER_ADMIN.SET_GROUPS (
        policy_name    => 'general_ols_policy',
        user_name      => 'globalit_3',
        read_groups    => 'IT_GLOBAL_TEC',
        row_groups     => 'IT_GLOBAL_TEC');
END;

BEGIN
   SA_USER_ADMIN.SET_LEVELS (
        policy_name  => 'general_ols_policy',
        user_name    => 'itsupport_2', --manager
        max_level    => 'S',
        min_level    => 'R'
        );

    SA_USER_ADMIN.SET_COMPARTMENTS (
        policy_name   => 'general_ols_policy',
        user_name     => 'itsupport_2',
        read_comps    => 'MAN,TEC,SAF,REP,REV,BLU,STR,RET,MED',
        write_comps   => 'MAN,TEC,SAF,REP,REV,BLU,STR');

   SA_USER_ADMIN.SET_GROUPS (
        policy_name    => 'general_ols_policy',
        user_name      => 'itsupport_2',
        read_groups    => 'IT_SUPPORT_MGR',
        row_groups     => 'IT_SUPPORT_MGR');
END;

BEGIN
   SA_USER_ADMIN.SET_LEVELS (
        policy_name  => 'general_ols_policy',
        user_name    => 'itsupport_1',
```

```
        max_level    => 'C',
        min_level    => 'R',
        def_level    => 'R',
        row_level    => 'R'
        );

    SA_USER_ADMIN.SET_COMPARTMENTS (
        policy_name   => 'general_ols_policy',
        user_name     => 'itsupport_1',
        read_comps    => 'TEC,SAF,REP,REV,BLU,STR,RET,MED',
        write_comps   => 'TEC,SAF,REP,REV');

   SA_USER_ADMIN.SET_GROUPS (
        policy_name    => 'general_ols_policy',
        user_name      => 'itsupport_1',
        read_groups    => 'IT_SUPPORT_TEC',
        row_groups     => 'IT_SUPPORT_TEC');
END;
```

Appendix B: OLS Policy Table Setup

The following code represents how the OLS policy created in Appendix A is applied to a table. In this example, the OLS policy is applied to PROJECTS table. The same is done across other tables depending on their use case, but the steps are largely similar.

```
-- Apply OLS policy to PROJECTS table
BEGIN
    SA_POLICY_ADMIN.APPLY_TABLE_POLICY (
            policy_name    => 'general_ols_policy',
            schema_name    => 'company',
            table_name     => 'projects',
            table_options  => 'read_control, write_control, label_default');
END;

-- Enable OLS policy on PROJECTS table
BEGIN
    SA_POLICY_ADMIN.ENABLE_TABLE_POLICY (
            policy_name => 'general_ols_policy',
            schema_name => 'company',
            table_name  => 'projects');
END;
```

For existing data added, the rows can also be updated by the administrator as such to suit the appropriate access control levels. Here is an example from COMPANY.EMPLOYEEBENEFITS

```
UPDATE Company.EmployeeBenefits eb
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'R')
WHERE eb.SecurityLevel = 'Restricted';

UPDATE Company.EmployeeBenefits eb
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'S::HR')
WHERE eb.SecurityLevel = 'Secret';

UPDATE Company.EmployeeBenefits eb
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'C::IT_GLOBAL')
WHERE eb.SecurityLevel = 'Confidential' and DEPARTMENT = 'Global IT';

UPDATE Company.EmployeeBenefits eb
SET general_ols_col = CHAR_TO_LABEL('general_ols_policy', 'C:FIN:HR')
WHERE eb.SecurityLevel = 'Confidential' and DEPARTMENT = 'HR';
```

Appendix C: VPD to facilitate Insert/Update access

VPD on Employees Table

```
CREATE CONTEXT employee_position USING company.context_employee_position_package;
/

CREATE OR REPLACE PACKAGE company.context_employee_position_package AS
  PROCEDURE set_employee_position_context;
END;
/

CREATE OR REPLACE PACKAGE BODY company.context_employee_position_package IS
  PROCEDURE set_employee_position_context IS
    v_username  VARCHAR2(40);
    v_employee_position VARCHAR2(12);
  BEGIN
    v_username := SYS_CONTEXT('USERENV','SESSION_USER');
    BEGIN
      SELECT position
      INTO   v_employee_position
      FROM   COMPANY.EMPLOYEES
      WHERE  UPPER(username) = v_username;

      DBMS_OUTPUT.PUT_LINE('set_employee_position_context position: ' ||
v_employee_position);

      DBMS_SESSION.set_context('employee_position','position', v_employee_position);
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        DBMS_SESSION.set_context('employee_position','position', NULL);
    END;
  END set_employee_position_context;
END context_employee_position_package;
/

GRANT EXECUTE ON company.context_employee_position_package TO PUBLIC;
CREATE PUBLIC SYNONYM context_employee_position_package FOR
company.context_employee_position_package;

GRANT ADMINISTER DATABASE TRIGGER TO COMPANY

CREATE OR REPLACE TRIGGER company.set_employee_position_trigger AFTER LOGON
ON DATABASE
BEGIN
```

```
  company.context_employee_position_package.set_employee_position_context;
END;
/

CREATE OR REPLACE FUNCTION company.employee_view_emplopyee_info(
          p_schema IN VARCHAR2, p_object IN VARCHAR2)
RETURN VARCHAR2 AS
   v_username VARCHAR2(40);
   v_position VARCHAR2(12);
   condition VARCHAR2(200);
BEGIN
   v_username := UPPER(SYS_CONTEXT('USERENV','SESSION_USER'));
   v_position := UPPER(SYS_CONTEXT('employee_position','position'));
   DBMS_OUTPUT.PUT_LINE('v_username: ' || v_username);
   DBMS_OUTPUT.PUT_LINE('v_position: ' || v_position);

  IF (v_username = 'COMPANY' OR v_username = 'SYSTEM' OR v_position = 'MANAGER' OR
v_position = 'RECRUITER') THEN
      DBMS_OUTPUT.PUT_LINE('employee_view_emplopyee_info return condition: ' || '1=1');
      RETURN '1=1';
   ELSE
      condition := 'UPPER(username) = SYS_CONTEXT("USERENV","SESSION_USER")';
      DBMS_OUTPUT.PUT_LINE('employee_view_emplopyee_info return condition: ' ||
condition);
      RETURN condition;
   END IF;

END employee_view_emplopyee_info;

BEGIN
          DBMS_RLS.ADD_POLICY (
          object_schema    => 'COMPANY',
          object_name      => 'EMPLOYEES',
          policy_name      => 'employee_view_emplopyee_info_policy',
          function_schema  => 'COMPANY',
          policy_function  => 'employee_view_emplopyee_info',
       update_check     => true
           );
END;
```

Appendix D: Database Table Creation Code

```
CREATE TABLE EmployeeTrainingRecords (
   TrainingRecordID    NUMBER PRIMARY KEY,
   EmployeeID          NUMBER REFERENCES Employees(EmployeeID),
   TrainingCerts       VARCHAR2(255),
   SecurityLevel       VARCHAR2(50),   -- e.g., 'Restricted', 'Confidential', ‘Secret’
   TrainingType        VARCHAR2(100),  -- Compartment (e.g., 'Technical', 'Managerial',
'Safety')
   Position            VARCHAR2(100)   -- Group (e.g., 'Manager', 'Engineer', 'Technician')
);
```

```
CREATE TABLE ProjectDocumentation (
   DocID               NUMBER PRIMARY KEY,
   ProjectName         VARCHAR2(255),
   DocumentType        VARCHAR2(100),  -- e.g., 'Technical', 'Financial', 'Strategy'
   Content             CLOB,
   SecurityLevel       VARCHAR2(50),   -- e.g., 'Restricted', 'Confidential', ‘Secret’
   Department          VARCHAR2(100),  -- Group; e.g., ‘Global IT’, 'HR', ‘IT Support’
   DataType            VARCHAR2(100)   -- Compartment; can be 'Report', 'Blueprint', 'Budget'
);
```

```
CREATE TABLE Employees (
   EmployeeID     NUMBER PRIMARY KEY,
   UserName       VARCHAR2(100),
   FirstName      VARCHAR2(100),
   LastName       VARCHAR2(100),
   Email          VARCHAR2(150) UNIQUE,
   DateOfJoining  DATE,
   SecurityLevel       VARCHAR2(50),   -- e.g., 'Restricted', 'Confidential', ‘Secret’
   Position       VARCHAR2(100),  -- Group; e.g., 'Manager', 'Engineer'
   Department     VARCHAR2(100)   -- Group; ‘HR’, ‘Global IT, IT Support
);
```

```
CREATE TABLE Departments (
   DepartmentID   NUMBER PRIMARY KEY,
   DepartmentName VARCHAR2(100)
);

– Departments: HR, Global IT (Parent Group), IT Support (Child Group)
```

```
CREATE TABLE Projects (
   ProjectID      NUMBER PRIMARY KEY,
   ProjectName    VARCHAR2(255),
   StartDate      DATE,
   EndDate        DATE,
   Department     VARCHAR2(100)   -- Group; ‘HR’, ‘Global IT, IT Support
);
```

```
CREATE TABLE EmployeeBenefits (
   BenefitID      NUMBER PRIMARY KEY,
   BenefitType    VARCHAR2(100),  -- e.g., 'Health Insurance', 'Stock Option'
   Details        VARCHAR2(500),
   SecurityLevel  VARCHAR2(50),   -- e.g., 'Restricted', 'Confidential', ‘Secret’
   Department     VARCHAR2(100),  -- Group; HR, Global IT, IT Support
   BenefitForm    VARCHAR2(100)   -- Compartment; can be 'Medical', 'Financial',
'Retirement'
);
```

```
CREATE TABLE MeetingNotes (
   NoteID         NUMBER PRIMARY KEY,
   MeetingDate    DATE,
   Topic          VARCHAR2(255),
   Content        CLOB,
   SecurityLevel  VARCHAR2(50),   -- e.g., 'Restricted', 'Confidential', ‘Secret’
   Department     VARCHAR2(100),  -- Group; HR, Global IT, IT Support
   MeetingType    VARCHAR2(100)   -- Compartment; e.g., 'Strategy', 'Budget', 'Review'
);
```