UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA

DEPARTMENT OF THEORETICAL AND APPLIED SCIENCES

MASTER'S DEGREE IN
COMPUTER SCIENCE

Access control implementation in a smart camping system

Sassi Gabriele - Gaudiano Antonio

Contents

Chapter 1

DB setting

1.1 Introduction

This report details the implementation of an access control system for a smart camping management system. The project aims to establish a robust access control mechanism for an organization that oversees multiple campsites within the same chain, utilizing Oracle as the underlying database platform. The access control system is designed to ensure secure, efficient, and user-friendly management of access permissions, tailored to the specific needs of different user roles and departments within the organization. This implementation enhances the security and operational efficiency of managing a network of campsites.

1.2 Functionalities

Administrative management

The administrator oversees various roles and responsibilities by managing the organization of its own camping , with a primary focus on the customer satisfaction. This includes managing newsletters, advertising campaigns, and other customer engagement initiatives, which are handled by officer employees.

Operational organization

Within each campground, dedicated personnel are responsible for monitoring and maintaining various aspects of the facility to ensure smooth operation. This includes security personnel who oversee access control, garden maintenance staff, plumbers, and technicians responsible for maintaining electrical systems and other smart ecosystem components.

Reservation System

Leveraging smart technology, customers can access information about their accommodation, including energy consumption data, during their stay. This feature not only benefits guests but also enables the administrator and office employees to monitor the proper usage and functionality of accommodations.

1.3 DB schema

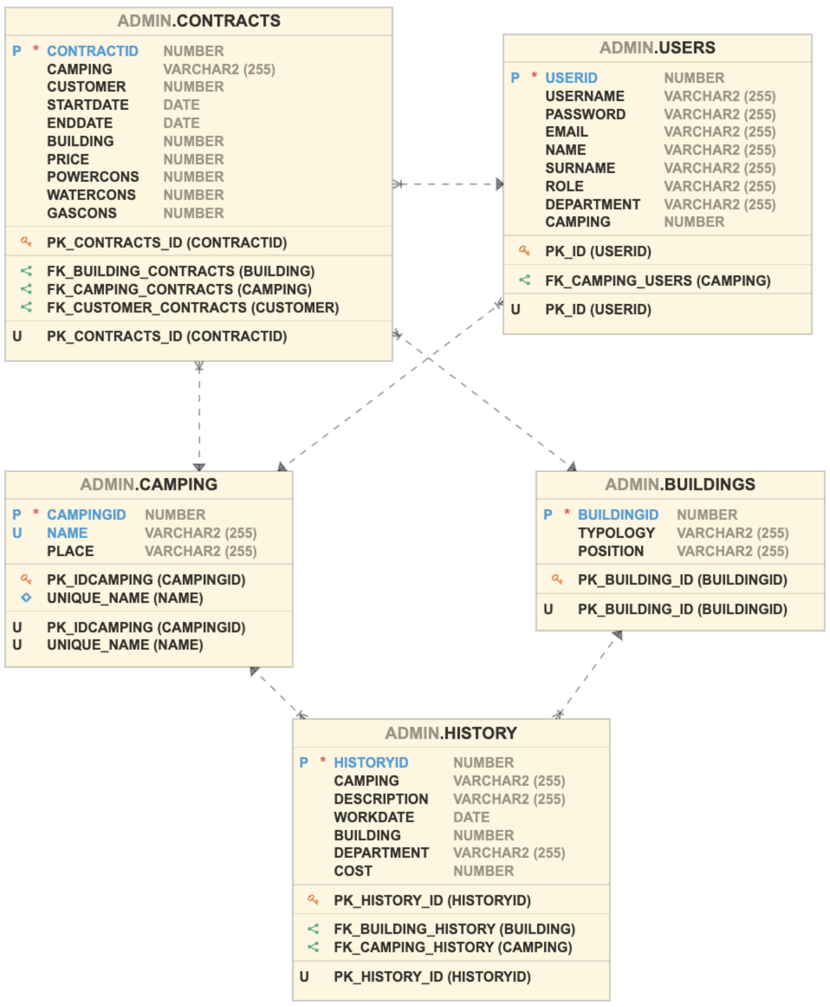


Figure 1.1: System logic scheme

In order to keep track of all information and organize the distributed campsites efficiently, a comprehensive database system has been implemented. This database serves as a centralized repository for recording and managing various aspects of campsites operations, including guest bookings, accommodation availability, maintenance schedules, energy consumption data, and employee assignments. Here is a description of the table's attributes represented in 1.1:

- **Camping**: This table holds the details of each camping site managed by the organization.
  - *CampingID*: Unique identifier for each camping.
  - *Name*: Name of the camping.
  - *Place*: Location of the camping.

- **Users**: This table contains the user information.
  - *userID*: identification number for the users.
  - *username*: username needed to log into the system.
  - password: password needed to log into the system.
  - *email*: email needed to sign in.
  - *name*: name of the user.
  - *surname*: surname of the user.
  - *role*: role of the user within the camping, such as: 'admin','employee','worker', 'customer', etc.
  - *department*: department to which the user belongs: 'Officer','Maintenance', 'Electrical', etc.
  - *camping*: identified of the camping associated with the user.
- **Buildings**: this table keeps track of the buildings within the camping sites.
  - *buildingID*: identification number for the building.
  - *typology*: type of building, such as 'bungalow', 'apartment', etc.
  - *position*: position of the building, such as 'near sea', 'internal', 'near forest', etc.
- **History**: this table records historical work activities related to the camping sites.
  - *historyID*: identification number for the maintenance operation.
  - *camping*: name of the camping associated with the history record.
  - *description*: description of the maintenance operation.
  - *workDate*: date of the maintenance operation.
  - *building*: building where the maintenance operation has been done.
  - *department*: area of the maintenance operation, such as 'Electrical', 'Surveillance', 'Plumber', etc.
  - *cost*: cost of the maintenance operation

- **Contracts**: This table stores information about the contracts made between cutomers and camping sites.
  - *contractID*: identification number for the rent contract.
  - *camping*: name of the camping where the contract has been created.
  - *customer*: person who signs the rent contract.
  - *startDate*, *endDate*: time span of the rent contract.
  - *building*: building that has been rented.
  - *price*: price of the rent contract.
  - *powerCons*, *waterCons*, *gasCons*: consuptions of the building during the rent time.

1.3.1 List of created users

In order to simulate and check the interactions within the campsites management system, several users have been created in the oracle database. Each user has specific responsibilities to ensure a comprehensive testing environment that mimics real-world scenarios. The following list details the created users:

- **Username:** jdoe
  - **Password:** Password_password_1
  - **Camping:** Mountain View
  - **Role:** Admin
- **Username:** gstone
  - **Password:** Password_password_16
  - **Camping:** Sunny Beach
  - **Role:** Admin
- **Username:** lgreen
  - **Password:** Password_password_6
  - **Camping:** Mountain View
  - **Role:** Employee
  - **Department:** Officer

- **Username:** vred
  - **Password:** Password_password_17
  - **Camping:** Sunny Beach
  - **Role:** Employee
  - **Department:** Officer
- **Username:** pblack
  - **Password:** Password_password_7
  - **Camping:** Sunny Beach
  - **Role:** Employee
  - **Department:** Maintenance
- **Username:** xcode
  - **Password:** Password_password_15
  - **Camping:** Lake Paradise
  - **Role:** Employee
  - **Department:** Maintenance
- **Username:** bwhite
  - **Password:** Password_password_3
  - **Role:** Leader
  - **Department:** Electrical
- **Username:** jparker
  - **Password:** Password_password_4
  - **Role:** Leader
  - **Department:** Plumber
- **Username:** asmith
  - **Password:** Password_password_2
  - **Role:** Worker
  - **Department:** Electrical

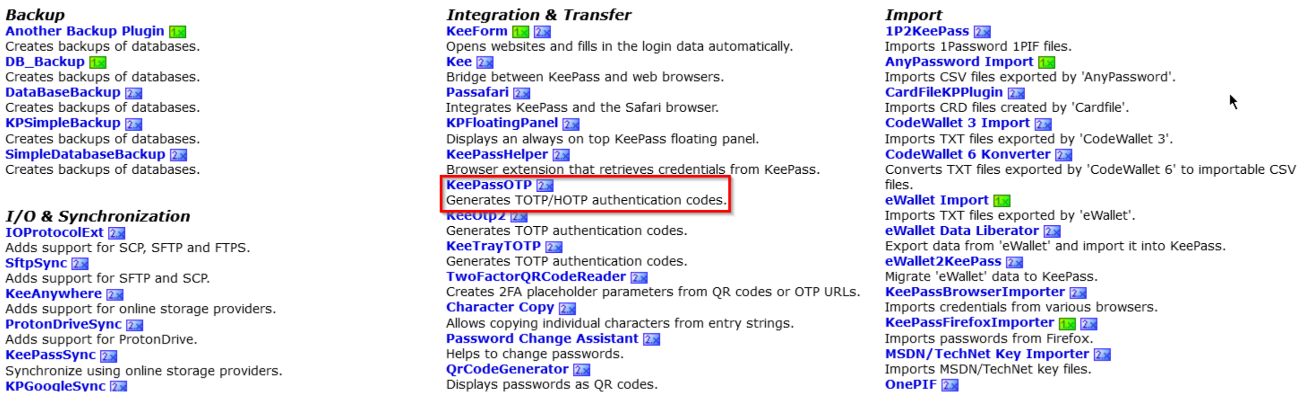
- **Username:** rjones
  - **Password:** Password_password_9
  - **Role:** Customer
- **Username:** kharries
  - **Password:** Password_password_10
  - **Role:** Guest

1.3.2 Oracle DB access

Here is the credentials to access on Oracle DB project:

- **Username:** `agaudiano@studenti.uninsubria.it`
- **Password:** `LABdata1!`

In order to authenticate with oracle two-factor authentication you will need KeePass with the KeePassOTP plugin installed from the offical page:



Backup
Another Backup Plugin
Creates backups of databases.
DB_Backup
Creates backups of databases.
DataBaseBackup
Creates backups of databases.
KPSimpleBackup
Creates backups of databases.
SimpleDatabaseBackup
Creates backups of databases.

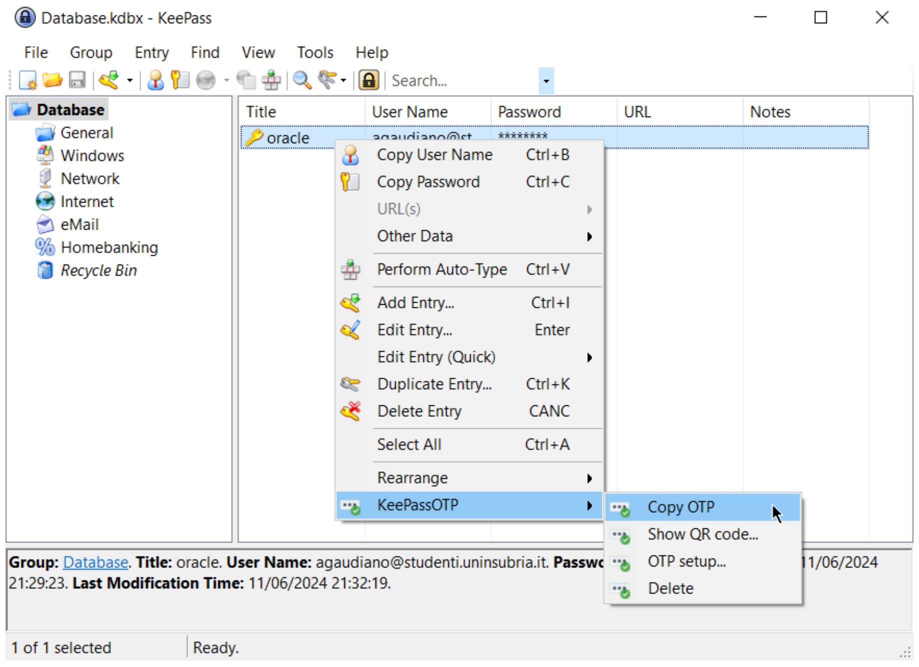
I/O & Synchronization
IOProtocolExt
Adds support for SCP, SFTP and FTPS.
SftpSync
Adds support for SFTP and SCP.
KeeAnywhere
Adds support for online storage providers.
ProtonDriveSync
Adds support for ProtonDrive.
KeePassSync
Synchronize using online storage providers.
KPGoogleSync

Integration & Transfer
KeeForm
Opens websites and fills in the login data automatically.
Kee
Bridge between KeePass and web browsers.
Passafari
Integrates KeePass and the Safari browser.
KPFloatingPanel
Displays an always on top KeePass floating panel.
KeePassHelper
Browser extension that retrieves credentials from KeePass.
KeePassOTP
Generates TOTP/HOTP authentication codes.
KeeOtp2
Generates TOTP authentication codes.
KeeTrayTOTP
Generates TOTP authentication codes.
TwoFactorQRCodeReader
Creates 2FA placeholder parameters from QR codes or OTP URLs.
Character Copy
Allows copying individual characters from entry strings.
Password Change Assistant
Helps to change passwords.
QrCodeGenerator
Displays passwords as QR codes.

Import
1P2KeePass
Imports 1Password 1PIF files.
AnyPassword Import
Imports CSV files exported by 'AnyPassword'.
CardFileKPPlugin
Imports CRD files created by 'Cardfile'.
CodeWallet 3 Import
Imports TXT files exported by 'CodeWallet 3'.
CodeWallet 6 Konverter
Converts TXT files exported by 'CodeWallet 6' to importable CSV files.
eWallet Import
Imports TXT files exported by 'eWallet'.
eWallet Data Liberator
Export data from 'eWallet' and import it into KeePass.
eWallet2KeePass
Migrate 'eWallet' data to KeePass.
KeePassBrowserImporter
Imports credentials from various browsers.
KeePassFirefoxImporter
Imports passwords from Firefox.
MSDN/TechNet Key Importer
Imports MSDN/TechNet key files.
OnePIF

Following the links you will download a file called *"KeePassOTP.plgx"* which needs to be put in the "plugin" directory of KeePass, then open the *"Database.kdbx"* file, two new windows will open asking for a password, both of them are "password".

Now you will be able to copy and paste the TOTP code needed to access the oracle cloud simply right-clicking on the entry like as follows:



Group: Database. **Title:** oracle. **User Name:** agaudiano@studenti.uninsubria.it. **Passwo** … 11/06/2024 21:29:23. **Last Modification Time:** 11/06/2024 21:32:19.

1 of 1 selected | Ready.

Chapter 2

Access control policies

2.1 Overview

Access control is a critical component of the smart camping management system, ensuring that users have appropriate levels of access based on their responsibilities. By implementing robust access control policies, we can protect sensitive information, streamline operations, and enhance the overall security of the system. This chapter outlines the access control policies implemented for the smart camping system, detailing the specific permissions and restrictions applied to different user.

2.2 Policies description

The following access control policies have been implemented to manage the interactions within the smart camping system:

1. The administrator of the camping has the privileges needed in order to see all the user information to manage the people in his camping.
2. The administrator of the camping has the privileges needed in order to see all the information to analyze, consult and monitor customer rents and works done in his camping.
3. The officer employee can only see the list of customer / guest of camping where they work except the credential (username and password).
4. The officer employee can access the list of contracts, in order to handle the rents of the camping.
5. The maintenance employee can manage the list of workers and leader of each department inside a specific camping.
6. The maintenance employee can access the history table of own camping in order to manage and monitor the logistic operations of each maintenance.
7. The Leader can see and modify specific maintenance only done by worker of its own department.
8. Workers, based on department, can only see and modify the information about the work that has been done, but they cannot access to sensitive information. (i.e. 'price').

9. The customer can see the entire information about its contract in all campsites.
10. The guest, which is the other component of the family who rent the building, can access only the typology of building and its price.

Each of these access control policies has been carefully implemented using different types of enforcement mechanism:

- **Discretional Access Control**.
- **Role Based Access Control**.
- **Fine Grained Access Control**.

Chapter 3

DAC enforcement

In this chapter we aim to explore the design, development and implementation of Discretional Access Control inside the system, based on the high-level policies.

3.1 Implementation

Policy 1

Based on the whole *'Users'* table, it's important to create different view to collect user for each camping.

```
CREATE VIEW mountain_view_user_list AS
SELECT userID, username, password, email, name, surname, role,
department, camping
FROM ADMIN.Users
WHERE camping = 1;

CREATE VIEW sunny_beach_user_list AS
SELECT userID, username, password, email, name, surname, role,
department, camping
FROM ADMIN.Users
WHERE camping = 2;

CREATE VIEW forest_retreat_user_list AS
SELECT userID, username, password, email, name, surname, role,
department, camping
FROM ADMIN.Users
WHERE camping = 3;

CREATE VIEW lake_paradise_user_list AS
SELECT userID, username, password, email, name, surname, role,
department, camping
FROM ADMIN.Users
WHERE camping = 4;
```

Now, the privileges can be managed only for the authorized users based on the operation that they can performed on the objects:

```
GRANT select,insert,update,delete ON mountain_view_user_list
TO jdoe WITH GRANT OPTION;
GRANT select,insert, update,delete ON sunny_beach_user_list
TO gstone WITH GRANT OPTION;
```

Users 'jdoe' and 'gstone' are admin respectively of 'Mountain View' and 'Sunny Beach' campsites. So they can access only information about individuals registered in their manages campgrounds.

Policy 2

Considering *'Contracts'* and *'History'* tables, a specific view for each camping in created.

```
CREATE VIEW mountain_view_contracts_list AS
SELECT contractID, camping, customer, startDate, endDate, building,
price, powercons, watercons, gascons
FROM ADMIN.Contracts
WHERE camping = 'Mountain View';

CREATE VIEW sunny_beach_contracts_list AS
SELECT contractID, camping, customer, startDate, endDate, building,
price, powercons, watercons, gascons
FROM ADMIN.Contracts
WHERE camping = 'Sunny Beach';

CREATE VIEW forest_retreat_contracts_list AS
SELECT contractID, camping, customer, startDate, endDate, building,
price, powercons, watercons, gascons
FROM ADMIN.Contracts
WHERE camping = 'Forest Retreat';

CREATE VIEW lake_paradise_contracts_list AS
SELECT contractID, camping, customer, startDate, endDate, building,
price, powercons, watercons, gascons
FROM ADMIN.Contracts
WHERE camping = 'Lake Paradise';
```

Similarly, administrators can only access contracts and work history related to their campsites. The following privileges are designed to support this control mechanism.

```
GRANT select,insert,update,delete ON mountain_view_contracts_list
TO jdoe WITH GRANT OPTION;
GRANT select,insert,update,delete ON mountain_view_history_list
TO jdoe WITH GRANT OPTION;
GRANT select,insert,update,delete ON buildings
TO jdoe WITH GRANT OPTION;

GRANT select,insert, update,delete ON sunny_beach_contracts_list
TO gstone WITH GRANT OPTION;
GRANT select,insert, update,delete ON sunny_beach_history_list
TO gstone WITH GRANT OPTION;
GRANT select,insert,update,delete ON buildings
TO gstone WITH GRANT OPTION;
```

Additionally, the "WITH GRANT OPTION" has been provided because, being in the field of management, they need the ability to administer the campsite.

Policy 3

Thanks to the previous policies, we can treat the view as a dedicated table for each specific campsite. To mask the *'username'* and *'password'* fields, we can create another view that excludes these sensitive fields, ensuring that officer employees cannot access them.

```
CREATE VIEW MV_employee_customer_list AS
SELECT userID, email, name, surname, role, camping
FROM ADMIN.mountain_view_user_list
WHERE role='Customer' OR role='Guest';

CREATE VIEW SB_employee_customer_list AS
SELECT userID, email, name, surname, role, camping
FROM ADMIN.sunny_beach_user_list
WHERE role='Customer' OR role='Guest';

CREATE VIEW FR_employee_customer_list AS
SELECT userID, email, name, surname, role, camping
FROM ADMIN.forest_retreat_user_list
WHERE role='Customer' OR role='Guest';
```

```
CREATE VIEW LP_employee_customer_list AS
SELECT userID, email, name, surname, role, camping
FROM ADMIN.lake_paradise_user_list
WHERE role='Customer' OR role='Guest';
```

These statements specify that if users ’lgreen’ and ’vred’ attempt to view the list of customers, they will only see those from their own campsite without sensitive fields.

```
GRANT select ON MV_employee_customer_list TO lgreen;
GRANT select ON SB_employee_customer_list TO vred;
```

Policy 4

Since the views already exist inside the database, it’s important only to grant the correct privileges on the correct objects, based on the policy, to the authorized users.

```
GRANT select,insert,update,delete ON mountain_view_contracts_list
TO lgreen;
GRANT select,insert,update,delete ON sunny_beach_contracts_list
TO vred;
```

Policy 5

By using the same logic idea of policy 3, and working on the already created views, new objects can be created to represent the list of workers and leaders for each camping.

```
CREATE VIEW MV_employee_worker_list AS
SELECT userID, email, name, surname, role, department, camping
FROM ADMIN.mountain_view_user_list
WHERE role='Leader' OR role='Worker';

CREATE VIEW SB_employee_worker_list AS
SELECT userID, email, name, surname, role, department, camping
FROM ADMIN.sunny_beach_user_list
WHERE role='Leader' OR role='Worker';

CREATE VIEW FR_employee_worker_list AS
SELECT userID, email, name, surname, role, department, camping
FROM ADMIN.forest_retreat_user_list
WHERE role='Leader' OR role='Worker';

CREATE VIEW LP_employee_worker_list AS
SELECT userID, email, name, surname, role, department, camping
FROM ADMIN.lake_paradise_user_list
WHERE role='Leader' OR role='Worker';
```

The maintenance employee ('pblack', 'xcode') can access and manage these information based on where they work, here are the instructions to give the correct privileges.

```
GRANT select,insert,update,delete ON SB_employee_worker_list TO pblack;
GRANT select,insert,update,delete ON LP_employee_worker_list TO xcode;
```

Policy 6

For the implementation of this policy, it is enough to assign the correct privileges so that only the necessary operations can be performed by users regarding the maintenance of their own campsite.

```
GRANT select,insert,update,delete ON sunny_beach_history_list TO pblack;
GRANT select,insert,update,delete ON lake_paradise_history_list TO xcode;
```

Policy 7

Different view can be created by using the *'History'* table, no matter the camping but it's fundamental that leader can only see the information about its department in order to handle its tasks.

```
CREATE VIEW electric_maintenance AS
SELECT * FROM ADMIN.History
WHERE department='Electrical';

CREATE VIEW plumber_maintenance AS
SELECT * FROM ADMIN.History
WHERE department='Plumber';

CREATE VIEW security_maintenance AS
SELECT * FROM ADMIN.History
WHERE department='Surveillance';
```

This view can use to filter information. In this way, 'bwhite' can perform operation related to electric maintenance and 'jparker' can access only data regarding the plumber operations.

```
GRANT select, insert, update ON electric_maintenance TO bwhite;
GRANT select, insert, update ON plumber_maintenance TO jparker;
```

Policy 8

Leveraging the previously created views, it is important that users labeled as 'worker', such as 'asmith', have access only to the information necessary to describe the work that has been done, based on their function inside the camping.

```
CREATE VIEW worker_electric_maintenance AS
SELECT historyID, camping, description, workDate, building
FROM ADMIN.electric_maintenance;

CREATE VIEW worker_plumber_maintenance AS
SELECT historyID, camping, description, workDate, building
FROM ADMIN.plumber_maintenance;

CREATE VIEW worker_security_maintenance AS
SELECT historyID, camping, description, workDate, building
FROM ADMIN.security_maintenance;

GRANT select, insert, update ON worker_electric_maintenance TO asmith;
```

3.2 Cost Analysis

To perform the DAC cost analysis for the given implementation we need to consider the following factors:

1. **View cost**: the cost associated with creating the view.
   - **#view**: total number of created views.
2. **Auth cost**: the cost related to grant the right permission on the object for each specific user.
   - **#auth**: total number of permission x number of users allowed to receive these permission on the specific object.

Policy 1

- Create views:
  - *mountain_view_user_list*;
  - *sunny_beach_user_list*;
  - *forest_retreat_user_list*;
  - *lake_paradise_user_list*
- Grant access (select, insert, update, delete) on these views to all administrator users inside the db.

#View: 4

#Auth: 4 x #admin

Policy 2

- Create views:
  - *mountain_view_history_list*;
  - *sunny_beach_history_list*;
  - *forest_retreat_history_list*;
  - *lake_paradise_history_list*;
  - *mountain_view_contracts_list*;
  - *sunny_beach_contracts_list*;
  - *forest_retreat_contracts_list*;
  - *lake_paradise_contracts_list*;
- Grant access (select, insert, update, delete) on these views to all administrator users inside the db.

#**View**: 8

#**Auth**: 4 x #admin

Policy 3

- Create views:
  - *MV_employee_customer_list*;
  - *SB_employee_customer_list*;
  - *FR_employee_customer_list*;
  - *LP_employee_customer_list*;
- Grant access (select) on these views to all officer employee users inside the db.

#**View**: 4

#**Auth**: 1 x #Officer employee

Policy 4

- Grant access (select,insert,update,delete) on specific views to all officer employee users inside the db.

#**Auth**: 4 x Officer employee

Policy 5

- Create views:
  - *MV_employee_worker_list*;
  - *SB_employee_worker_list*;
  - *FR_employee_worker_list*;
  - *LP_employee_worker_list*;
- Grant access (select,insert,update,delete) on these views to all maintenance employee users inside the db.

#**View**: 4

#**Auth**: 4 x #Maintenance employee

Policy 6

- Grant access (select,insert,update,delete) on specific views to all maintenance employee users inside the db.

#**Auth**: 4 x #Maintenance employee

Policy 7

- Create views:
  - *electric_maintenance*;
  - *plumber_maintenance*;
  - *security_maintenance*;
- Grant access (select,insert,update) on these views to all leader users inside the db.

#**View**: 3

#**Auth**: 3 x #Leader

Policy 8

- Create views:
  - *worker_electric_maintenance*;
  - *worker_plumber_maintenance*;
  - *worker_security_maintenance*;
- Grant access (select,insert,update) on these views to all worker users inside the db.

#**View**: 3

#**Auth**: 3 x #Worker

Chapter 4

RBAC enforcement

The effective management of access privileges is essential to ensure the security, efficiency and integrity of operational processes. Role Based Access Control (RBAC) emerges as a important framework for access control governance, offering a structured approach to managing user permissions based on their roles, responsibilities, and organizational hierarchy.

4.1 Implementation

4.1.1 Roles

To define Role-Based Access Control (RBAC), we need to establish, inside the database, a set of roles:

- **Manager:** perform all the operation allowed for the 'admin'
- **Officer employee**
- **Maintenance employee**
- **Leader:** electrical, plumber, security
- **Worker:** electrical, plumber, security

4.1.2 Hierarchy graph

In RBAC, roles can be organized into hierarchies to simplify the management of permissions. Each role in the hierarchy inherits the permission of its parent role, allowing for a structured and scalable approach to access control. To better understand the concept, we must represent the hierarchy graphs making a distinction between the different campsites and between camping and maintenance. The visual representation in figures 4.1,4.2,4.3 helps to illustrate how roles are structured and how permission are inherited inside the smart systems.

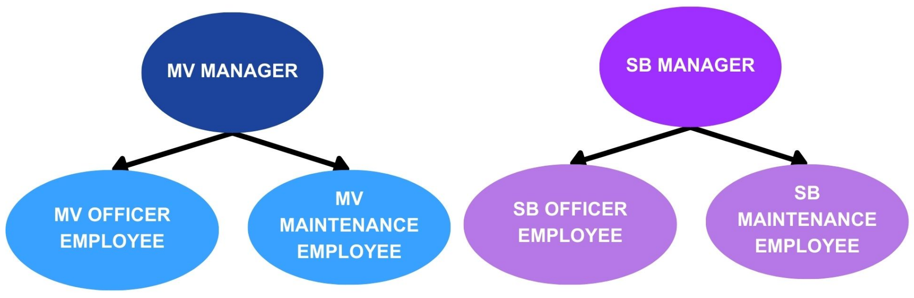


Figure 4.1: Mountain View - Sunny Beach hierarchy graphs

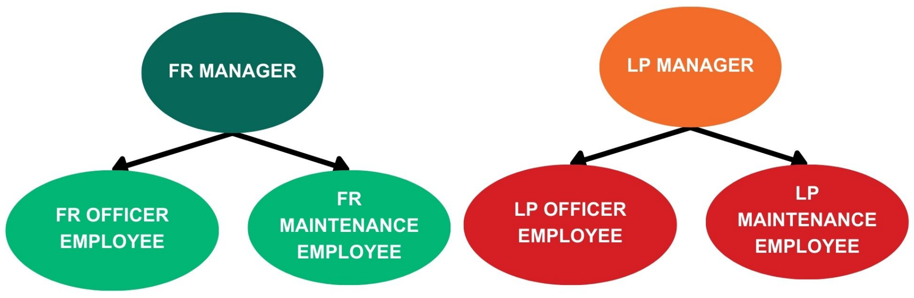


Figure 4.2: Forest Retreat - Lake Paradise hierarchy graphs

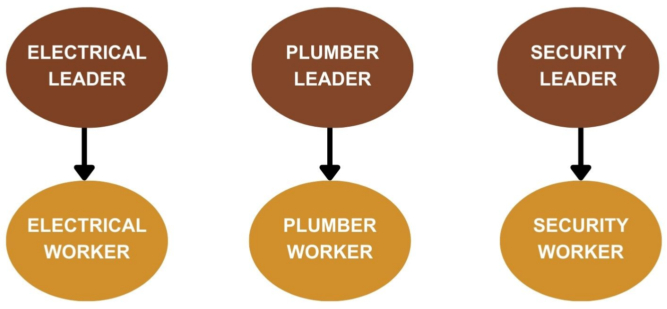


Figure 4.3: Maintenance hierarchy graph

Here are the SQL statements to describe the role hierarchy:

```sql
--Mountain view
GRANT MV_OFFICER_EMPLOYEE TO MV_MANAGER;
GRANT MV_MAINTENANCE_EMPLOYEE TO MV_MANAGER;

--Sunny Beach
GRANT SB_OFFICER_EMPLOYEE TO SB_MANAGER;
GRANT SB_MAINTENANCE_EMPLOYEE TO SB_MANAGER;

--Forest Retreat
GRANT FR_OFFICER_EMPLOYEE TO FR_MANAGER;
GRANT FR_MAINTENANCE_EMPLOYEE TO FR_MANAGER;

--Lake Paradise
GRANT LP_OFFICER_EMPLOYEE TO LP_MANAGER;
GRANT LP_MAINTENANCE_EMPLOYEE TO LP_MANAGER;

GRANT ELECTRICAL_WORKER TO ELECTRICAL_LEADER;
GRANT PLUMBER_WORKER TO PLUMBER_LEADER;
GRANT SECURITY_WORKER TO SECURITY_LEADER;
```

4.1.3 Privileges to roles - (PA)

Each role must have its own distinct set of permissions, allowing users to perform specific actions within the system:

```sql
--Policy 1-2
GRANT select,insert,update,delete ON mountain_view_user_list
TO MV_MANAGER WITH ADMIN OPTION;
GRANT select,insert,update,delete ON mountain_view_contracts_list
TO MV_MANAGER WITH ADMIN OPTION;
GRANT select,insert,update,delete ON mountain_view_history_list
TO MV_MANAGER WITH ADMIN OPTION;
GRANT select,insert,update,delete ON buildings
TO MV_MANAGER WITH ADMIN OPTION;

GRANT select,insert, update,delete ON sunny_beach_user_list
TO SB_MANAGER WITH ADMIN OPTION;
GRANT select,insert, update,delete ON sunny_beach_contracts_list
TO SB_MANAGER WITH ADMIN OPTION;
GRANT select,insert, update,delete ON sunny_beach_history_list
TO SB_MANAGER WITH ADMIN OPTION;
GRANT select,insert,update,delete ON buildings
TO SB_MANAGER WITH ADMIN OPTION;
```

```
--Policy 3
GRANT select ON MV_employee_customer_list TO MV_OFFICER_EMPLOYEE;
GRANT select ON SB_employee_customer_list TO SB_OFFICER_EMPLOYEE;

--Policy 4
GRANT select,insert,update,delete ON mountain_view_contracts_list
TO MV_OFFICER_EMPLOYEE;
GRANT select,insert,update,delete ON sunny_beach_contracts_list
TO SB_OFFICER_EMPLOYEE;

--Policy 5
GRANT select,insert,update,delete ON SB_employee_worker_list
TO SB_MAINTENANCE_EMPLOYEE;
GRANT select,insert,update,delete ON LP_employee_worker_list
TO LP_MAINTENANCE_EMPLOYEE;

--Policy 6
GRANT select,insert,update,delete ON sunny_beach_history_list
TO SB_MAINTENANCE_EMPLOYEE;
GRANT select,insert,update,delete ON lake_paradise_history_list
TO LP_MAINTENANCE_EMPLOYEE;

--Policy 7
GRANT select, insert, update ON electric_maintenance TO ELECTRICAL_LEADER;
GRANT select, insert, update ON plumber_maintenance TO PLUMBER_LEADER;

--Policy 8
GRANT select, insert, update ON worker_electric_maintenance
TO ELECTRICAL_WORKER;
```

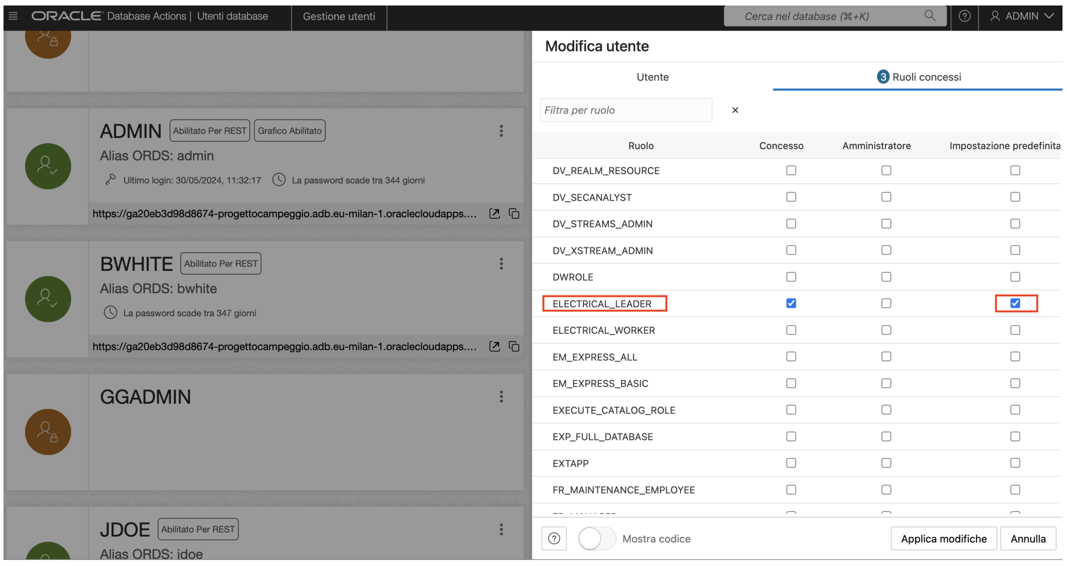
4.1.4 Granting roles - (RA)

With DAC, privileges were granted based on user discretion, assigning necessary privileges to each specific user, whose responsibility was represented as an attribute in the users table, allowing them to perform only their tasks. With RBAC, the assignment of privileges is managed by a central authority and enforced by the system, using a function-based access instead of ownership-based access. This is achieved by creating specific roles within the system, each with specific functionalities. Users assigned to a particular role are automatically enabled to perform the operations permitted for that role.

```sql
GRANT MV_MANAGER TO jdoe;
GRANT SB_MANAGER TO gstone;

GRANT MV_OFFICER_EMPLOYEE TO lgreen;
GRANT SB_OFFICER_EMPLOYEE TO vred;

GRANT SB_MAINTENANCE_EMPLOYEE TO pblack;
GRANT LP_MAINTENANCE_EMPLOYEE TO xcode;

GRANT ELECTRICAL_LEADER TO bwhite;
GRANT PLUMBER_LEADER TO jparker;

GRANT ELECTRICAL_WORKER TO asmith;
```

4.1.5 User session assignment - (UA)

Thanks to Oracle, it is possible to automatically set a role when the user logs into the system (showed in Fig.4.4). This features ensures that users have the necessary privileges assigned to them as soon as they start their session.



ADMIN Abilitato Per REST Grafico Abilitato
Alias ORDS: admin
Ultimo login: 30/05/2024, 11:32:17 La password scade tra 344 giorni
https://ga20eb3d98d8674-progettocampeggio.adb.eu-milan-1.oraclecloudapps....

BWHITE Abilitato Per REST
Alias ORDS: bwhite
La password scade tra 347 giorni
https://ga20eb3d98d8674-progettocampeggio.adb.eu-milan-1.oraclecloudapps....

GGADMIN

JDOE Abilitato Per REST
Alias ORDS: jdoe

Modifica utente

Utente | 3 Ruoli concessi

Filtra per ruolo

Ruolo	Concesso	Amministratore	Impostazione predefinita
DV_REALM_RESOURCE	☐	☐	☐
DV_SECANALYST	☐	☐	☐
DV_STREAMS_ADMIN	☐	☐	☐
DV_XSTREAM_ADMIN	☐	☐	☐
DWROLE	☐	☐	☐
ELECTRICAL_LEADER	✓	☐	✓
ELECTRICAL_WORKER	☐	☐	☐
EM_EXPRESS_ALL	☐	☐	☐
EM_EXPRESS_BASIC	☐	☐	☐
EXECUTE_CATALOG_ROLE	☐	☐	☐
EXP_FULL_DATABASE	☐	☐	☐
EXTAPP	☐	☐	☐
FR_MAINTENANCE_EMPLOYEE	☐	☐	☐

Figure 4.4: User session assignment

However, if a user is unable to execute queries on an object for which they have privileges, it's important to explicitly set the role in the current session, once log on with user, using the following statement:

```
SET ROLE role_name;
```

This command activates the specified role (only if he/she can actually "play" this role), enabling the user to perform operations permitted by that role.

4.2 Cost analysis

To perform a Role-Based Access Control (RBAC) cost analysis for the given policy, we need to consider the following factors:

1. **View cost:** the cost associated with creating the views[1].
2. **Auth cost - (PA):** the cost associated with managing the permission to role.
   - **#auth:** number of permission x the considered role/s.
3. **Role cost - (RA):** the cost related to add the specific role to the user's role set.
   - **#role:** number of user allowed to have the specific role/s.

Policy 1

- It's allowed to select,insert,update,delete on each created view.
- Permissions are granted to 'Manager' role, for each camping.
- The 'Manager' is assigned to users able to play that role.

#view: 4

#auth: 4 x 4

#role: 2

Policy 2

- It's allowed to select,insert,update,delete on each created view.
- Permissions are granted to 'Manager' role, for each camping.

#view: 8

#role: 4 x 4

#auth: Already done for policy 1

[1]This field represents the same value as the analysis done for DAC.

Policy 3

- It's allowed to select on each created view.
- Permissions are granted to 'Officer employee' role, for each camping.
- The 'Officer employee' is assigned to users able to play that role.

#view: 4

#role: 1 x 4

#auth: 2

Policy 4

- It's allowed to select,insert,update,delete on each specific view.
- Permissions are granted to 'Officer employee' role, for each camping.

#role: 4 x 4

#auth: Already done for policy 3

Policy 5

- It's allowed to select,insert,update,delete on each created view.
- Permissions are granted to 'Maintenance employee' role, for each camping.
- The role is assigned to users able to play that role.

#view: 4

#role: 4 x 2

#auth: 2

Policy 6

- It's allowed to select,insert,update,delete on each specific view.
- Permissions are granted to 'Maintenance employee' role, for each camping.

#role: 4 x 2

#auth: Already done for policy 5

Policy 7

- It’s allowed to select,insert,update on each created view.
- Permissions are granted to ’Leader’ role, for each department.
- The ’Leader’ is assigned to users able to play that role.

#view: 3

#role: 3 x 3

#auth: 2

Policy 8

- It’s allowed to select,insert,update on each created view.
- Permissions are granted to ’Worker’ role, for each department.
- The ’Worker’ is assigned to users able to play that role.

#view: 3

#role: 3 x 3

#auth: 1

Chapter 5

FGAC enforcement

5.1 Introduction

Oracle VPD is a feature of Oracle Database that allows to implement fine-grained access control policies on tables, views, etc. In enables to restrict data access at the row level based on dynamic conditions. Policies are evaluated at runtime based on various factors such as: user identity, application context, session attributes. Virtual Private Database represents a valuable alternative to view-based approach using in SQL, which support a type of FGAC, but:

- In the presence of many FGAC policies, there is the need of generating and maintaining a huge number of views.
- Not all FGAC requirements can be enforced through views.

5.2 Implementation

Here is step-by-step implementation inside the smart camping system:

1. **Application context creation**;
2. **Policy function creation**;
3. **Attach the policy to a table**

5.2.1 Application context creation

An application context is a namespace within the database that can store session-specific attribute information that can be used by policy functions.

```
--Context creation
CREATE CONTEXT camping_ctx USING camping_ctx_pkg;

--Package definition
CREATE OR REPLACE EDITIONABLE PACKAGE admin.camping_ctx_pkg IS
  PROCEDURE set_user_data;
END camping_ctx_pkg;
```

In the smart camping management scenario, the most useful attributes of connected user to retrieve using VPD are:

- **userID**;
- **role**;
- **department**;
- **campingID**

```
CREATE OR REPLACE EDITIONABLE PACKAGE BODY admin.camping_ctx_pkg IS
  PROCEDURE set_user_data IS
    username_1 VARCHAR(400);
    user_id USERS.userID%TYPE;
    user_role USERS.role%TYPE;
    user_department USERS.department%TYPE;
    camping_id CAMPING.campingID%TYPE;

  BEGIN
    --retrieve user's name from session
    username_1 := SYS_CONTEXT('USERENV', 'SESSION_USER');
    DBMS_OUTPUT.PUT_LINE('Username: ' || LOWER(username_1));

    --retrieve id, role, department and campingID of connected user
    SELECT USERS.userID, USERS.role, USERS.department, USERS.camping
    INTO user_id, user_role, user_department, camping_id
    FROM ADMIN.USERS
    WHERE LOWER(USERS.username) = LOWER(username_1);

    DBMS_OUTPUT.PUT_LINE('user_id: ' || user_id);
    DBMS_OUTPUT.PUT_LINE('user_role: ' || user_role);
    DBMS_OUTPUT.PUT_LINE('user_department: ' || user_department);
    DBMS_OUTPUT.PUT_LINE('camping_id: ' || camping_id);

     -- setting up context
    DBMS_SESSION.SET_CONTEXT('camping_ctx', 'user_id', user_id);
    DBMS_SESSION.SET_CONTEXT('camping_ctx', 'user_role', user_role);
    DBMS_SESSION.SET_CONTEXT('camping_ctx', 'user_department',
    user_department);
    DBMS_SESSION.SET_CONTEXT('camping_ctx', 'camping_id', camping_id);
```

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('No data found for user ' || username_1);
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Unexpected error while setting user context.');
  END set_user_data;

END camping_ctx_pkg;
```

A Trigger has been defined in order to execute the procedure of context package just after the logon.

```
CREATE OR REPLACE EDITIONABLE TRIGGER admin.camping_ctx_trig
AFTER LOGON ON DATABASE
BEGIN
    admin.camping_ctx_pkg.set_user_data;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Unexpected error in logon trigger.');
END camping_ctx_trig;
```

5.2.2 VPD policy functions creation

The PL/SQL function for each policy is created. It is used to construct and return the dynamic predicate, based on session attributes that enforce row-level access control. Is called every time a user execute a query on the table, the return predicate is add to the *'where'* clause of the query.

Note that: The implemented policies are the same described in Chapter 2.2. However some modifications are made in order to make them suitable for VPD implementation.

Policy 1

The administrator can access and manage only users registered in his camping.

```
CREATE OR REPLACE FUNCTION admin.admin_user_info (schema_var VARCHAR2,
table_var VARCHAR2)
RETURN VARCHAR2 IS
  predicate VARCHAR2(4000);
  user_role VARCHAR2(100);
  camping_id NUMERIC;

BEGIN
  --Retrieve role and camping id from the context
  user_role := SYS_CONTEXT('camping_ctx', 'user_role');
  camping_id := SYS_CONTEXT('camping_ctx', 'camping_id');

  IF user_role = 'Admin'
    THEN predicate := 'users.camping = ' || camping_id;

  END IF;

  RETURN predicate;
END admin_user_info;
```

Policies 2-4

Admins and officer employees can only access the rents of own camping.

```
CREATE OR REPLACE FUNCTION admin.camping_customer_info (schema_var
VARCHAR2, table_var VARCHAR2)
RETURN VARCHAR2 IS
  predicate VARCHAR2(4000);
  user_id NUMERIC;
  user_role VARCHAR2(100);
  user_department VARCHAR2(400);

BEGIN
  user_id := SYS_CONTEXT('camping_ctx', 'user_id');
  user_role := SYS_CONTEXT('camping_ctx', 'user_role');
  user_department := SYS_CONTEXT('camping_ctx','user_department');

  IF user_role = 'Admin' OR (user_role = 'Employee'
  AND user_department = 'Officer')
    THEN predicate := 'contracts.camping IN (
        SELECT c.name FROM Users u JOIN Camping c
        ON u.camping = c.campingID
        WHERE u.userID = ' || user_id || ')';
  END IF;

  RETURN predicate;
END camping_customer_info;
```

Policy 3

Officer employees see the list of customers and guests based on the camping they work (without accessing the username and password fields). Before implement the function, is necessary to create a view that collect all customers and guests of all campsites masking the sensitive attributes.

```
CREATE VIEW employee_customer_list AS
SELECT userID, email, name, surname, role, camping
FROM ADMIN.Users
WHERE role='Customer' OR role='Guest'
```

```
CREATE OR REPLACE FUNCTION admin.employee_user_info (schema_var
VARCHAR2, table_var VARCHAR2)
RETURN VARCHAR2 IS
  predicate VARCHAR2(4000);
  user_role VARCHAR2(100);
  user_department VARCHAR2(400);
  camping_id NUMERIC;

BEGIN
  -- Retrieve attributes based on context
  user_role := SYS_CONTEXT('camping_ctx', 'user_role');
  user_department := SYS_CONTEXT('camping_ctx', 'user_department');
  camping_id := SYS_CONTEXT('camping_ctx', 'camping_id');

  IF user_role = 'Employee' AND user_department = 'Officer'
  THEN predicate := 'employee_customer_list.camping = ' || camping_id;

  END IF;

  RETURN predicate;
END employee_user_info;
```

Policy 5

Maintenance employees manage the list of external workers based on the campsite (without accessing the username and password fields). A view is created to mask the specified fields.

```
CREATE VIEW employee_workers_list AS
SELECT userID, email, name, surname, role, department, camping
FROM admin.Users
WHERE role = 'Worker' OR role='Leader';
```

```
CREATE OR REPLACE FUNCTION admin.employee_workers_info (schema_var
VARCHAR2, table_var VARCHAR2)
RETURN VARCHAR2 IS
  predicate VARCHAR2(4000);
  user_role VARCHAR2(100);
  user_department VARCHAR2(400);
  camping_id NUMERIC;

BEGIN
  --Retrieve role, department and campingID of the connected user
  user_role := SYS_CONTEXT('camping_ctx', 'user_role');
  user_department := SYS_CONTEXT('camping_ctx', 'user_department');
  camping_id := SYS_CONTEXT('camping_ctx', 'camping_id');

  IF user_role = 'Employee' AND user_department = 'Maintenance'
  THEN predicate := 'employee_workers_list.camping = ' || camping_id;

  END IF;

  RETURN predicate;
END employee_workers_info;
```

Policies 2-6-7

Admins and Maintenance employees access the history of jobs done based on the camping where they work or manage, leaders access maintenance done based on their job.

```
CREATE OR REPLACE FUNCTION admin.maintenance_info (schema_var VARCHAR2,
table_var VARCHAR2)
RETURN VARCHAR2 IS
  predicate VARCHAR2(4000);
  user_role VARCHAR2(100);
  user_department VARCHAR2(400);
  user_id NUMERIC;

BEGIN
   --Retrieve role,departmente and userID from the application context
  user_role := SYS_CONTEXT('camping_ctx', 'user_role');
  user_department := SYS_CONTEXT('camping_ctx', 'user_department');
  user_id := SYS_CONTEXT('camping_ctx', 'user_id');

  --Predicate to filter the results
  IF user_role = 'Admin' OR (user_role = 'Employee'
  AND user_department = 'Maintenance')
    THEN predicate := 'history.camping IN (
        SELECT c.name FROM Users u JOIN Camping c
        ON u.camping = c.campingID
        WHERE u.userID = ' || user_id || ')';

  ELSIF user_role = 'Leader'
    THEN predicate := 'history.department = ''' || user_department || '''';

  END IF;

  RETURN predicate;
END maintenance_info;
```

Policy 8

Workers can access based on department only to see and enter description and date of work performed. For this purpose is necessary to create the view, in which the dynamic behavior will be applied.

```
CREATE VIEW worker_maintenance AS
SELECT historyID, description, building, workdate, department
FROM ADMIN.History;

CREATE OR REPLACE FUNCTION admin.worker_maintenance_info (schema_var
VARCHAR2, table_var VARCHAR2)
RETURN VARCHAR2 IS
  predicate VARCHAR2(4000);
  user_role VARCHAR2(100);
  user_department VARCHAR2(400);

BEGIN
  --Role and department from context
  user_role := SYS_CONTEXT('camping_ctx', 'user_role');
  user_department := SYS_CONTEXT('camping_ctx', 'user_department');

  IF user_role = 'Worker'
    THEN predicate := 'worker_maintenance.department = ''' ||
    user_department || '''';

  END IF;

  RETURN predicate;
END worker_maintenance_info;
```

Policy 9

Customers can only access to their rentals (current and historical).

```
CREATE OR REPLACE FUNCTION admin.customer_info (schema_var VARCHAR2,
table_var VARCHAR2)
RETURN VARCHAR2 IS
  predicate VARCHAR2(4000);
  user_id NUMERIC;
  user_role VARCHAR2(100);
  user_department VARCHAR2(400);

BEGIN
  user_role := SYS_CONTEXT('camping_ctx', 'user_role');
  user_id := SYS_CONTEXT('camping_ctx', 'user_id');
  user_department := SYS_CONTEXT('camping_ctx','user_department');

  IF user_role = 'Admin' THEN predicate := '1=1';

  ELSIF user_role = 'Employee' AND user_department = 'Officer'
    THEN predicate := '1=1';

  ELSE predicate := 'contracts.customer = ''' || user_id || '''';

  END IF;

  RETURN predicate;
END customer_info;
```

New Policy

Users should only see contracts that started before the current date.

```
CREATE OR REPLACE FUNCTION contracts_data_policy (
  schema_var IN VARCHAR2, table_var IN VARCHAR2)
  RETURN VARCHAR2 AS
BEGIN
  RETURN 'startDate < SYSDATE';
END;
```

New Policy

Users can only access records in the History table for the past 5 years.

```
CREATE OR REPLACE FUNCTION history_date_policy (
  schema_name IN VARCHAR2, table_name IN VARCHAR2) RETURN VARCHAR2 AS
BEGIN
  RETURN 'workDate >= ADD_MONTHS(SYSDATE, -60)';
END;
```

5.2.3 VPD policy attaching

This last step specifies the table to protect, the policy to use and the types of SQL statements the policy applies.

Policy 1

```
BEGIN
    SYS.DBMS_RLS.ADD_POLICY(
        OBJECT_SCHEMA => 'ADMIN',
        OBJECT_NAME => 'USERS',
        POLICY_NAME => 'admin_user_info_policy',
        FUNCTION_SCHEMA => 'ADMIN',
        POLICY_FUNCTION => 'ADMIN_USER_INFO'
    );
END;
```

Policies 2-4

```
BEGIN
    SYS.DBMS_RLS.ADD_POLICY(
        OBJECT_SCHEMA => 'ADMIN',
        OBJECT_NAME => 'CONTRACTS',
        POLICY_NAME => 'camping_customer_info_policy',
        FUNCTION_SCHEMA => 'ADMIN',
        POLICY_FUNCTION => 'CAMPING_CUSTOMER_INFO'
    );
END;
```

Policy 3

```
BEGIN
    SYS.DBMS_RLS.ADD_POLICY(
        OBJECT_SCHEMA => 'ADMIN',
        OBJECT_NAME => 'employee_customer_list',
        POLICY_NAME => 'employee_user_info_policy',
        FUNCTION_SCHEMA => 'ADMIN',
        POLICY_FUNCTION => 'EMPLOYEE_USER_INFO',
        STATEMENT_TYPES => 'SELECT'
    );
END;
```

Policy 5

```
BEGIN
    SYS.DBMS_RLS.ADD_POLICY(
        OBJECT_SCHEMA => 'ADMIN',
        OBJECT_NAME => 'employee_workers_list',
        POLICY_NAME => 'employee_workers_info_policy',
        FUNCTION_SCHEMA => 'ADMIN',
        POLICY_FUNCTION => 'EMPLOYEE_WORKERS_INFO'
    );
END;
```

Policy 2-6-7

```
BEGIN
    SYS.DBMS_RLS.ADD_POLICY(
        OBJECT_SCHEMA => 'ADMIN',
        OBJECT_NAME => 'HISTORY',
        POLICY_NAME => 'maintenance_info_policy',
        FUNCTION_SCHEMA => 'ADMIN',
        POLICY_FUNCTION => 'MAINTENANCE_INFO'
    );
END;
```

Policy 8

```
BEGIN
    SYS.DBMS_RLS.ADD_POLICY(
        OBJECT_SCHEMA => 'ADMIN',
        OBJECT_NAME => 'worker_maintenance',
        POLICY_NAME => 'worker_maintenance_info_policy',
        FUNCTION_SCHEMA => 'ADMIN',
        POLICY_FUNCTION => 'WORKER_MAINTENANCE_INFO'
    );
END;
```

Policy 9

```
BEGIN
    SYS.DBMS_RLS.ADD_POLICY(
        OBJECT_SCHEMA => 'ADMIN',
        OBJECT_NAME => 'CONTRACTS',
        POLICY_NAME => 'customer_info_policy',
        FUNCTION_SCHEMA => 'ADMIN',
        POLICY_FUNCTION => 'CUSTOMER_INFO',
        STATEMENT_TYPES => 'SELECT'
    );
END;
```

New Policy

```
BEGIN
  DBMS_RLS.ADD_POLICY(
    object_schema   => 'ADMIN',
    object_name     => 'CONTRACTS',
    policy_name     => 'contracts_data_policy',
    function_schema => 'ADMIN',
    policy_function => 'CONTRACTS_DATA',
  );
END;
```

New Policy

```
BEGIN
  DBMS_RLS.ADD_POLICY(
    object_schema   => 'ADMIN',
    object_name     => 'HISTORY',
    policy_name     => 'history_date_policy',
    function_schema => 'ADMIN',
    policy_function => 'HISTORY_DATE',
  );
END;
```

5.2.4 Additional operations

Before implementing and testing oracle VPD inside the scenario, it's important to add grant operations in order to assign the correct privileges to users on other objects inside the system .

```
--Privileges for policy 1
GRANT select,insert,update,delete ON USERS TO jdoe;
GRANT select,insert,update,delete ON USERS TO gstone;

--Privileges for policy 2-4
--Admin
GRANT select,insert,update,delete ON CONTRACTS TO gstone;
GRANT select,insert,update,delete ON CONTRACTS TO jdoe;
--Officer Employee
GRANT select ON CONTRACTS TO lgreen;
GRANT select ON CONTRACTS TO vred;

--Privileges for policy 3
GRANT SELECT ON employee_customer_list TO lgreen;
GRANT SELECT ON employee_customer_list TO vred;

--Privileges for policy 5
GRANT select,insert,update,delete ON employee_workers_list TO pblack;
GRANT select,insert,update,delete ON employee_workers_list TO xcode;

-- Privileges for policy 2-6-7
--Admin
GRANT sselect,insert,update,delete ON HISTORY TO jdoe;
GRANT select,insert,update,delete ON HISTORY TO gstone;
--Maintenance employees
GRANT select ON HISTORY TO xcode;
GRANT select ON HISTORY TO pblack;
```

```
--Workers/leaders
GRANT select,insert,update ON HISTORY TO jparker;

--Privilege for policy 8
GRANT select,insert,update ON worker_maintenance TO asmith;

--Privilege for policy 9
GRANT select ON CONTRACTS TO rjones;
```

Please, not that privileges on objects are granted directly to the user. If RBAC is implemented, privileges can be assigned directly to roles instead of individual users, saving memory and improving system efficiency.

Chapter 6

OLS access policies

Oracle Label Security provides the support for MAC within the Oracle DBMS. It controls access to the content of a row by comparing the row's label with a user's session label. Only if certain condition is met the access in granted. Let's integrate it inside our scenario.

6.1 High-level policy definition

New policies must be defined in order to implement OLS enforcement. Then, user/data labels are be properly modeled, considering:

- **Security level**;
- **Compartments**;
- **Groups**

6.1.1 Maintenance importance

Campings include the "maintenance manager" with the function to monitor the maintenance operations. Based on the price of the work, they need to be monitored by "senior maintenance manager" for high price operations or "regular maintenance manager" for lower ones.

The policy is implemented with two **security levels** to manage the operations:

- **Sensitive**: enables access to maintenance history that costs less or equal than 1400 euro.
- **Highly_sensitive**: enables access to maintenance history that costs more than 1400.

Concerning the **compartments** we consider them according to the department of campings:

- **Electrical**;
- **Plumbing**;
- **Surveillance**;

Concerning the **groups** we consider them according to the possible location of the campsites, which we can assume to be in

- **Europe**;
- **Asia**;
- **US**;
- **UK**

So a possible user could have the following associated label:

"SENSITIVE:ELECTRICAL:EUROPE"

6.1.2 Premium buildings

Campings have premium buildings for VIP customers. Based on the building the profile could be "vip", "advanced", "regular" and therefore they needs to be managed by different personnel.

The policy is implemented with three **security levels** to manage the building:

- **Public**: enables access to building with id = 1 (for example).
- **Sensitive**: enables access to building with id = 2 (for example).
- **Highly_sensitive**: enables access to building with id = 2 (for example).

Concerning the **compartments** we assume them according to the their positions:

- **Lake**;
- **Sea**;
- **Plains**;
- **Hills**

Concerning the **groups** we consider them according to the possible location of the campsites, which we can assume to be:

- **Europe**;
- **Asia**;
- **US**;
- **UK**

So a possible user could have the following associated label:

"PUBLIC:SEA:UK"

6.1.3 Users information

Campings distinguish users information by role. Based on the role of the users their information could be "private", "restricted", "public".

The policy is implemented with three **security levels** to manage the users:

- **Public**: enables access to guests information.
- **Restricted**: enables access to workers information.
- **Private**: enables access to customers information.

Concerning the **compartments** we consider them according to the possible location of the campsites, which we can assume to be:

- **Europe**;
- **Asia**;
- **US**;
- **UK**

In this particular case there are not groups, since they are not mandatory and there is no necessity to implement them.

So a possible user could have the following associated label:

"RESTRICTED:ASIA:"

6.1.4 Contracts level

Contracts can now identify different kind of users, we can assume them to be "premium", "regular" so that the officer employees can create personalized offers and so on. They can be identified, for example, querying the contracts table for contracts that lasted for "x" days or users who signed "x" contracts in the last year or so.

The policy could be implemented with two **security levels** to manage the contracts:

- **Public**: enables access to regular users information.
- **Restricted**: enables access to premium users information.

Concerning the compartments and groups we can consider a different approach, setting them to be equal to departments and roles of the camping, respectively for **compartments**:

- **Officer**;
- **Maintenance**

And for **groups**:

- **Admin**;
- **Employee**;
- **Worker**

So a possible user could have the following associated label:

"RESTRICTED:OFFICER:EMPLOYEE"

6.2 Example of implementation

In order to completely understand the OLS functionality, let's implement in oracle the first policy: "Maintenance importance".

Assume we create two more users "doroty" and "james" in order to test the new policy.

```
-- 1) Create the Oracle Label Security Policy Container
BEGIN
    SA_SYSDBA.CREATE_POLICY(
        policy_name => 'Camping_OLS_POL',
        column_name => 'OLS_CONTAINER_COLUMN',
        default_option => 'read_control'
    );
END;

-- 2) Enable the Policy
EXEC SA_SYSDBA.ENABLE_POLICY ('Camping_OLS_POL');

-- 3) Create Data Labels for the Label Security Policy
BEGIN
    SA_COMPONENTS.CREATE_LEVEL (
                policy_name => 'Camping_OLS_POL',
                level_num => 3000,
                short_name => 'HS',
                long_name => 'HIGHLY_SENSITIVE');
END;

BEGIN
    SA_COMPONENTS.CREATE_LEVEL (
                Policy_name => 'Camping_OLS_POL',
                level_num => 2000,
                short_name => 'S',
                long_name => 'SENSITIVE');
END;
```

```
-- 3.1) Create Data Compartments
BEGIN
  SA_COMPONENTS.CREATE_COMPARTMENT (
    policy_name       => 'Camping_OLS_POL',
    long_name         => 'ELECTRICAL',
    short_name        => 'EL',
    comp_num          =>  1000);

  SA_COMPONENTS.CREATE_COMPARTMENT (
    policy_name       => 'Camping_OLS_POL',
    long_name         => 'PLUMBING',
    short_name        => 'PLB',
    comp_num          =>  2000);

    SA_COMPONENTS.CREATE_COMPARTMENT (
      policy_name       => 'Camping_OLS_POL',
      long_name         => 'SURVEILLANCE',
      short_name        => 'SRV',
      comp_num          =>  1000);
END;
-- 3.2) Create Data Groups
BEGIN
  SA_COMPONENTS.CREATE_GROUP (
   policy_name      => 'Camping_OLS_POL',
   group_num        => 4000,
   short_name       => 'EU',
   long_name        => 'EUROPE');

   SA_COMPONENTS.CREATE_GROUP (
   policy_name      => 'Camping_OLS_POL',
   group_num        => 3000,
   short_name       => 'UK',
   long_name        => 'UNITED_KINGDOM');

   SA_COMPONENTS.CREATE_GROUP (
   policy_name      => 'Camping_OLS_POL',
   group_num        => 2000,
   short_name       => 'US',
   long_name        => 'UNITED_STATES');
```

```
SA_COMPONENTS.CREATE_GROUP (
   policy_name      => 'Camping_OLS_POL',
   group_num        => 1000,
   short_name       => 'AS',
   long_name        => 'ASIA');
END;

BEGIN
 SA_USER_ADMIN.SET_GROUPS (
  policy_name     => 'Camping_OLS_POL',
  user_name       => 'DOROTY',
  read_groups     => 'EU');

 SA_USER_ADMIN.SET_GROUPS (
  policy_name     => 'Camping_OLS_POL',
  user_name       => 'JAMES',
  read_groups     => 'US');
END;

-- 4) Associating the Policy Components with a Named Data Label
BEGIN
        SA_LABEL_ADMIN.CREATE_LABEL (
                        policy_name => 'Camping_OLS_POL',
                        label_tag => 3100,
                        label_value => 'HS:ELECTRICAL',
                        data_label => TRUE);
END;

BEGIN
        SA_LABEL_ADMIN.CREATE_LABEL (
                        policy_name => 'Camping_OLS_POL',
                        label_tag => 2100,
                        label_value => 'S:PLUMBING',
                        data_label => TRUE);
END;
```

```
BEGIN
   SA_USER_ADMIN.SET_USER_LABELS (
      policy_name    => 'Camping_OLS_POL',
      user_name      => 'DOROTY',
      max_read_label => 'S:PLUMBING:');

   SA_USER_ADMIN.SET_USER_LABELS (
      policy_name    => 'Camping_OLS_POL',
      user_name      => 'JAMES',
      max_read_label => 'HS:ELECTRICAL:');
END;

-- 5) Authorizing Users for Label Security Policies
BEGIN SA_USER_ADMIN.SET_LEVELS(
        policy_name => 'Camping_OLS_POL',
        user_name => 'DOROTY',
        max_level => 'HS',
        min_level => 'S',
        def_level => 'HS',
        row_level => 'HS');
END;

BEGIN SA_USER_ADMIN.SET_LEVELS(
        policy_name => 'Camping_OLS_POL',
        user_name => 'JAMES',
        max_level => 'S',
        min_level => 'S',
        def_level => 'S',
        row_level => 'S');
END;

-- 6) Apply the Policy to a Database Table
BEGIN SA_POLICY_ADMIN.APPLY_TABLE_POLICY (
        policy_name => 'Camping_OLS_POL',
        schema_name => 'ADMIN',
        table_name => 'HISTORY');
END;
```

```
--7) Adding a Policy Label to a Table Row
BEGIN
SA_USER_ADMIN.SET_USER_PRIVS (
        policy_name => 'Camping_OLS_POL',
        user_name => 'ADMIN',
        privileges => 'FULL');
END;


UPDATE admin.history SET OLS_COL = CHAR_TO_LABEL(
    'OLS_CONTAINER_COLUMN',
    'HS')
WHERE UPPER(COST) > 1400;

UPDATE admin.history SET OLS_COL = CHAR_TO_LABEL(
    'OLS_CONTAINER_COLUMN',
    'S')
WHERE UPPER(COSt) <= 1400;
```