UNIVERSITY OF BUCHAREST

FACULTY OF MATHEMATICS
AND COMPUTER SCIENCE

MASTER SECURITY AND APPLIED LOGIC

Database Security

PROJECT REPORT

Author
MATHIS DORY

Professor
LETITIA ANA MARIN

Bucharest, January 2024

Contents

1 Introduction

The following project involves securing an Oracle 21c database used for a fictitious vehicle rental application. The application is designed as follows: Customers can reserve a vehicle for a chosen period. They can also choose the vehicle, model and equipment they want. Vehicles have a daily price and a price per kilometre that varies according to model and equipment. In the event of late arrival, a fine must be paid and in the event of theft or breakage, a deposit must also be paid. An employee is then responsible for checking in the vehicle. When the vehicle is returned, the employee carries out a check-out and closes the reservation. What's more, in the event of damage to the vehicle, the employee has the option of immediately making the vehicle unavailable so that it can be sent for maintenance, but normally it's the role of the fleet manager to manage the vehicles. The fleet manager manages the vehicles, models and options. There is also an HR manager who manages the employees. Once the booking is closed, the employee calculates the invoice and sends it to the customer.

In order to secure the database the followings approaches were used:

- Management of the computational resources and users.
- Roles and privileges management.
- Encryption of critical data.
- Audit of tables.
- Data masking when exporting it.
- Application context and security.

2 Diagrams

The following figures are used to illustrate the final structure of the database. Note that the white tables in the ownership diagram 3 are owned by the admin user.

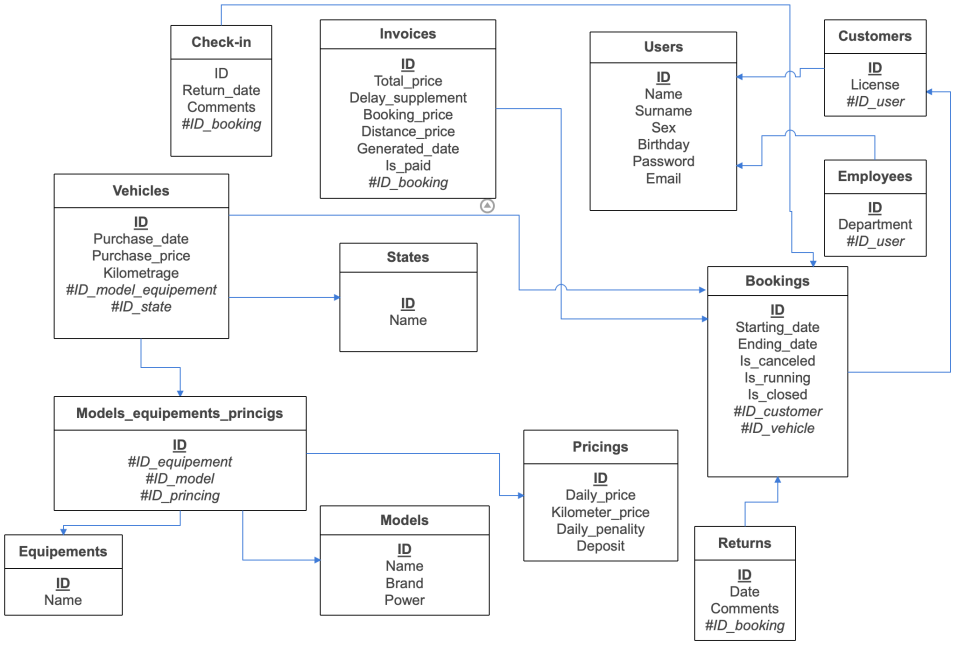


Figure 1: Relational Diagram

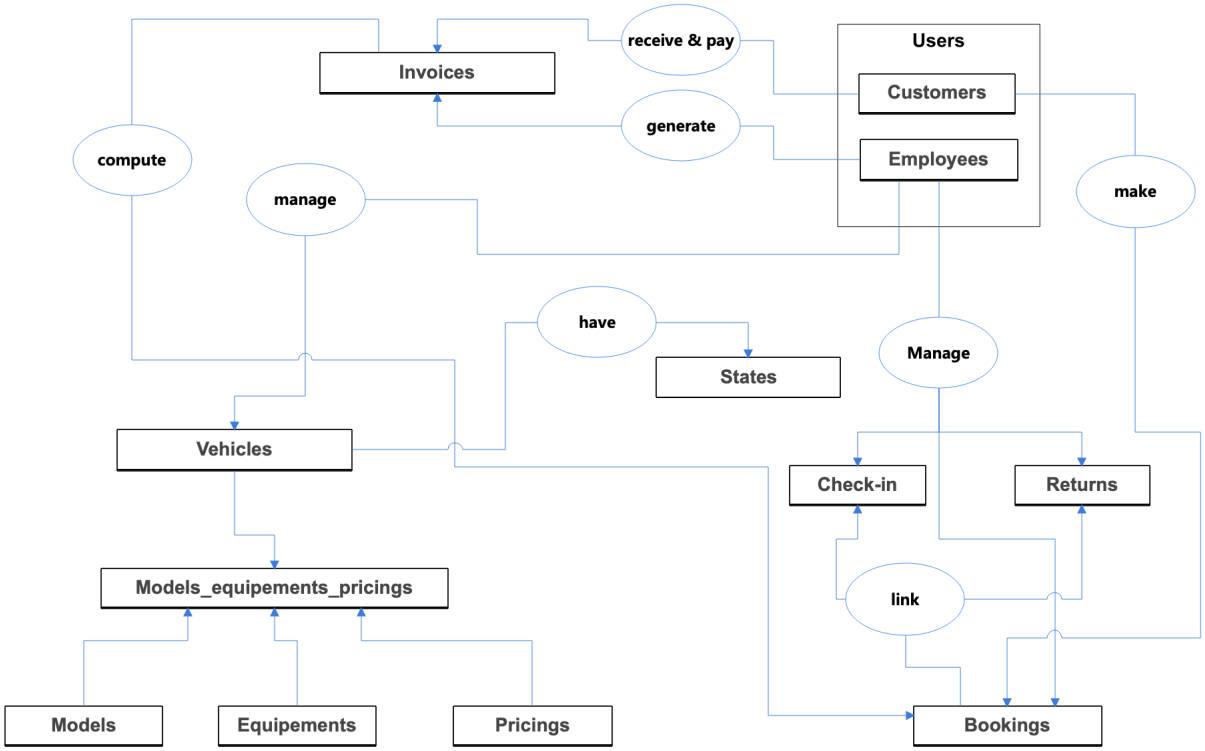


Figure 2: Conceptual Diagram

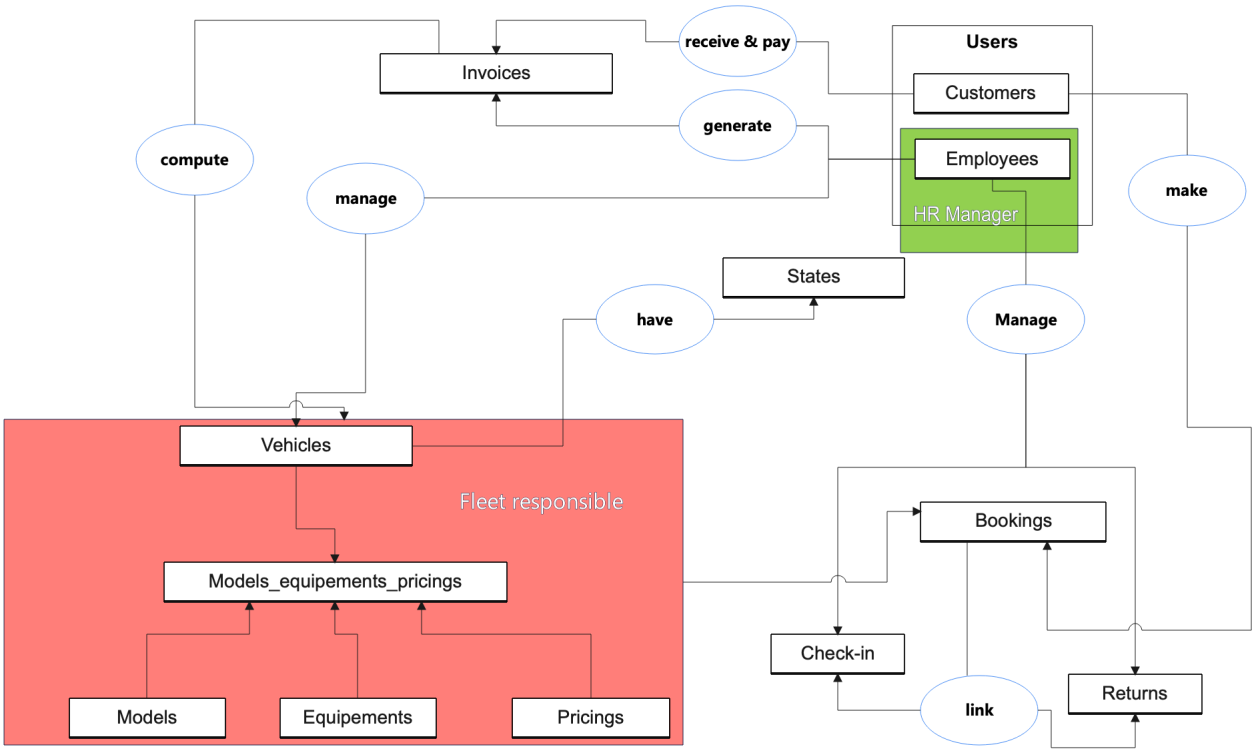


Figure 3: Ownership Diagram

3 Users and resources management

Before starting SQL scripts, different matrices are defined in order to assign the necessary privileges and create necessary users. Starting with the process-user matrix, it identify the users involved in the different processes of the application as shown on figure 4. The second matrix is the entity-process matrix and is used to identify which permission a process need on a specific table in order to be successful as shown on figure 4. By combining both matrices, the entity-user matrix is created on figure 6. This one is the final matrix which identify the privileges every users need to have.

Process-user matrix	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
Application administrator							X					
Employees		X		X				X	X	X	X	X
Customers	X	X	X		X		X					
HR manager						X						
Fleet reesponsible										X		

ID	Process
P1	Make a vehicle reservation
P2	Edit a vehicle reservation
P3	Pay invoice
P4	Generate invoice
P5	View invoice
P6	Manage employees
P7	Manage customers
P8	Create a vehicle check-in
P9	Create a vehicle return
P10	Manage vehicles
P11	Manage Check-in
P12	Manage Returns

For scalability reason customers are users at application level

Employees need this permission in order to call the procedure used to change a vehicule state

Figure 4: Process-user matrix

Entity-process matrix	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
Vehicles	S			S				S	S	I,U,D,S		
States										S		
Models	S									I,U,D,S		
Equipements	S			S						I,U,D,S		
Pricing	S			S						I,U,D,S		
Bookings	I	U,S		S				S	S			
Employee						I,U,D,S						
Invoices			S,U	I,U	S							
Returns				S					I		U,D	
Check-in				S				I				U,D
Users	S			S		I,U,D,S	I,U,D,S					
Customers	S			S			I,U,D,S					

Legend: I= Insert , U= update , D= delete, S= select

ID	Process
P1	Make a vehicle reservation
P2	Edit a vehicle reservation
P3	Pay invoice
P4	Generate invoice
P5	View invoice
P6	Manage employee
P7	Manage customers
P8	Create a vehicle check-in
P9	Create a vehicle return
P10	Manage vehicles
P11	Manage Check-in
P12	Manage Returns

Figure 5: Entity-process matrix

Entity-user matrix	Employee	Admin app	Fleet responsible	HR manager
Vehicles	S,U		I,U,D,S	
States	S	I,U,D	S	
Models			I,U,D,S	
Equipements			I,U,D,S	
Pricings	S		I,U,D,S	
Bookings	U,S		U,S	
Employees				I,U,D,S
Invoices	I,U			
Returns	S,I,U,D			
Check-in	S,I,U,D			
Customers	S	I,U,D,S		
Users	S	I,U,D,S		I,U,D,S

Customers are not DB users, they are users at the application level

Admin can also manage the list of states

Figure 6: Entity-user matrix

From now, the different database users are identified and the accounts can be created

using the **sys** account in the portable database as follows:

```
--+++++++ Create the users +++++++--

-- Create the admin user of the application, the password expire is used to force the user to change the password at the first login
-- This admin is a local user for the pluggable database orclpdb
CREATE USER appcar_admin_app IDENTIFIED BY admin1234 password expire;


-- Create an account for the responsible of the fleet
CREATE USER appcar_fleet_responsible IDENTIFIED BY test1234 password expire;

-- Create an account for the HR responsible
CREATE USER appcar_hr_manager IDENTIFIED BY test1234 password expire;

-- Create an account for the 5 employees of the commercial department
CREATE USER appcar_employee_1 IDENTIFIED BY test1234 password expire;
CREATE USER appcar_employee_2 IDENTIFIED BY test1234 password expire;
CREATE USER appcar_employee_3 IDENTIFIED BY test1234 password expire;
CREATE USER appcar_employee_4 IDENTIFIED BY test1234 password expire;
CREATE USER appcar_employee_5 IDENTIFIED BY test1234 password expire;


SELECT username, AUTHENTICATION_TYPE, ACCOUNT_STATUS, to_char(EXPIRY_DATE, 'dd/mm/yyyy hh24:mi:ss') AS expiry_date_time,
       to_char(CREATED, 'dd/mm/yyyy hh24:mi:ss') created_date_time,PROFILE FROM dba_users
WHERE lower(username) LIKE 'appcar%';

```

Figure 7: Users script

	USERNAME	AUTHENTICATION_TYPE	ACCOUNT_STATUS	EXPIRY_DATE_TIME	CREATED_DATE_TIME	PROFILE
1	APPCAR_ADMIN_APP	PASSWORD	EXPIRED	17/01/2024 14:08:45	17/01/2024 14:08:45	DEFAULT
2	APPCAR_FLEET_RESPONSIBLE	PASSWORD	EXPIRED	17/01/2024 14:08:45	17/01/2024 14:08:45	DEFAULT
3	APPCAR_EMPLOYEE_5	PASSWORD	EXPIRED	17/01/2024 14:08:45	17/01/2024 14:08:45	DEFAULT
4	APPCAR_HR_MANAGER	PASSWORD	EXPIRED	17/01/2024 14:08:45	17/01/2024 14:08:45	DEFAULT
5	APPCAR_EMPLOYEE_3	PASSWORD	EXPIRED	17/01/2024 14:08:45	17/01/2024 14:08:45	DEFAULT
6	APPCAR_EMPLOYEE_1	PASSWORD	EXPIRED	17/01/2024 14:08:45	17/01/2024 14:08:45	DEFAULT
7	APPCAR_EMPLOYEE_2	PASSWORD	EXPIRED	17/01/2024 14:08:45	17/01/2024 14:08:45	DEFAULT
8	APPCAR_EMPLOYEE_4	PASSWORD	EXPIRED	17/01/2024 14:08:45	17/01/2024 14:08:45	DEFAULT

Figure 8: Users list

After the accounts creation, it is important to set the storage limit for the users. The administrator of the application receives unlimited storage, the fleet responsible receives 20MB, the HR manager receives 10 MB and the employees 3MB.

```
--+++++++ Set storage limit for users +++++++--

-- unlimited storage for the admin
alter user appcar_admin_app quota unlimited on users;
-- 20 MB for the fleet responsible
alter user appcar_fleet_responsible quota 20M on users;
-- 10 MB for the HR manager
alter user appcar_hr_manager quota 10M on users;
-- 3 MB storage for the employees
alter user appcar_employee_1 quota 3M on users;
alter user appcar_employee_2 quota 3M on users;
alter user appcar_employee_3 quota 3M on users;
alter user appcar_employee_4 quota 3M on users;
alter user appcar_employee_5 quota 3M on users;

-- Check the storage limits
SELECT * FROM dba_ts_quotas WHERE tablespace_name LIKE 'USERS';
```

Figure 9: Storage script

	TABLESPACE_NAME	USERNAME	BYTES	MAX_BYTES	BLOCKS	MAX_BLOCKS	DROPPED
1	USERS	APPCAR_FLEET_RESPONSIBLE	0	20971520	0	2560	NO
2	USERS	APPCAR_EMPLOYEE_2	0	3145728	0	384	NO
3	USERS	APPCAR_EMPLOYEE_1	0	3145728	0	384	NO
4	USERS	APPCAR_EMPLOYEE_5	0	3145728	0	384	NO
5	USERS	APPCAR_EMPLOYEE_4	0	3145728	0	384	NO
6	USERS	APPCAR_ADMIN_APP	0	-1	0	-1	NO
7	USERS	APPCAR_HR_MANAGER	0	10485760	0	1280	NO
8	USERS	APPCAR_EMPLOYEE_3	0	3145728	0	384	NO

Figure 10: Storage list

In order to monitor the CPU resources and other settings for the different users, profiles can be used. Only 2 profiles will be used: one is used for all kind of employees (HR manager and fleet responsible included) and one for the admin account.

```
68  --+++++++ Create profiles +++++++--
69
70  -- Create the profile for all employees
71  CREATE PROFILE appcar_profile_employee LIMIT
72      SESSIONS_PER_USER 3 -- 3 sessions
73      CPU_PER_CALL 3000 -- 30 seconds threshold for the CPU
74      IDLE_TIME 5 -- 5 minutes
75      CONNECT_TIME 30 -- 30 minutes
76      PASSWORD_LIFE_TIME 90 -- 3 months
77      PASSWORD_REUSE_TIME UNLIMITED -- no reuse of the password
78      PASSWORD_LOCK_TIME 1/24 -- 1 hour
79      FAILED_LOGIN_ATTEMPTS 5; -- 5 attempts
80
81  -- Set the profile for the users
82  ALTER USER appcar_employee_1 profile appcar_profile_employee;
83  ALTER USER appcar_employee_2 profile appcar_profile_employee;
84  ALTER USER appcar_employee_3 profile appcar_profile_employee;
85  ALTER USER appcar_employee_4 profile appcar_profile_employee;
86  ALTER USER appcar_employee_5 profile appcar_profile_employee;
87  ALTER USER appcar_fleet_responsible profile appcar_profile_employee;
88  ALTER USER appcar_hr_manager profile appcar_profile_employee;
89
90  -- Create the profile for the admin
91  CREATE PROFILE appcar_profile_admin LIMIT
92      SESSIONS_PER_USER 3 -- 3 sessions
93      CPU_PER_CALL UNLIMITED -- had to use unlimited for the export of data
94      IDLE_TIME 3 -- 3 minutes
95      CONNECT_TIME 30 -- 30 minutes
96      PASSWORD_LIFE_TIME 30 -- 1 months
97      PASSWORD_REUSE_TIME UNLIMITED -- no reuse of the password
98      PASSWORD_LOCK_TIME 1 -- 1 day
99      FAILED_LOGIN_ATTEMPTS 3; -- 3 attempts
100
```

Figure 11: Profiles script

	USERNAME	PROFILE
1	APPCAR_FLEET_RESPONSIBLE	APPCAR_PROFILE_EMPLOYEE
2	APPCAR_EMPLOYEE_5	APPCAR_PROFILE_EMPLOYEE
3	APPCAR_HR_MANAGER	APPCAR_PROFILE_EMPLOYEE
4	APPCAR_EMPLOYEE_3	APPCAR_PROFILE_EMPLOYEE
5	APPCAR_EMPLOYEE_1	APPCAR_PROFILE_EMPLOYEE
6	APPCAR_EMPLOYEE_2	APPCAR_PROFILE_EMPLOYEE
7	APPCAR_EMPLOYEE_4	APPCAR_PROFILE_EMPLOYEE
8	APPCAR_ADMIN_APP	APPCAR_PROFILE_ADMIN

Figure 12: Profiles list

The last step is the resource plan of the application. This plan includes directives for each consumer group, dictating how resources should be allocated among them. This

step is essential for maintaining performance and preventing resource hogging by any single user or group. The *mgmt* group might be allocated 30% of resources, the *admin* group 25%, the *employee* group 35%, and *OTHER_GROUPS* 10%. The full script of the resource plan can be found in the **1-management.sql** file.

	CONSUMER_GROUP_ID	CONSUMER_GROUP	CPU_METHOD	MGMT_METHOD	INTERNAL_USE	COMMENTS	CATEGORY	STATUS	MANDATORY
1	20970	OTHER_GROUPS	ROUND-ROBIN	ROUND-ROBIN	NO	Consumer group for users not included in any consumer group with a …	OTHER	<null>	YES
2	20971	DEFAULT_CONSUMER_GROUP	ROUND-ROBIN	ROUND-ROBIN	NO	Consumer group for users not assigned to any consumer group	OTHER	<null>	YES
3	20973	LOW_GROUP	ROUND-ROBIN	ROUND-ROBIN	NO	Consumer group for low-priority sessions	OTHER	<null>	NO
4	75776	MGMT	ROUND-ROBIN	ROUND-ROBIN	NO	Groups of sessions of the HR manager and the fleet responsible	OTHER	<null>	NO
5	75777	ADMIN	ROUND-ROBIN	ROUND-ROBIN	NO	Groups of sessions of the admin	OTHER	<null>	NO
6	75778	EMPLOYEE	ROUND-ROBIN	ROUND-ROBIN	NO	Groups of sessions of the employees	OTHER	<null>	NO

Figure 13: Consumer groups of the resource plan

4 Privileges and roles

Now that accounts are created, it is necessary to grant them the required privileges. The first thing to do is to authorize the account to connect to the database by using the *GRANT CREATE SESSION TO* syntax. Because the application administrator will own most of the tables, so it is important to give him the necessary privileges so that he can create tables, indexes and sequences. After that, the first part of the tables can be created with the **create_tables.sql** script and by using the administrator account. The HR manager needs the permissions to create foreign keys on tables that are not in its schema. To do this, the *GRANT REFERENCES* instruction is given to the HR manager on the following table: **APPCAR_ADMIN_APP.USERS**. Then, the administrator is granted the references permission to the **APPCAR_FLEET_RESPONSIBLE.VEHICLES**. The user responsible for the vehicle fleet must be able to reference **APPCAR_ADMIN_APP.STATES**, which belongs to the admin.

	OWNER	TABLE_NAME
1	APPCAR_ADMIN_APP	BOOKINGS
2	APPCAR_ADMIN_APP	CHECK_IN
3	APPCAR_ADMIN_APP	CUSTOMERS
4	APPCAR_ADMIN_APP	INVOICES
5	APPCAR_ADMIN_APP	RETURNS
6	APPCAR_ADMIN_APP	STATES
7	APPCAR_ADMIN_APP	USERS
8	APPCAR_FLEET_RESPONSIBLE	EQUIPMENTS
9	APPCAR_FLEET_RESPONSIBLE	MODELS
10	APPCAR_FLEET_RESPONSIBLE	MODELS_EQUIPMENTS_PRICINGS
11	APPCAR_FLEET_RESPONSIBLE	PRICINGS
12	APPCAR_FLEET_RESPONSIBLE	VEHICLES
13	APPCAR_HR_MANAGER	EMPLOYEES

Figure 14: Created tables

After the tables creation, roles can help assign permissions more quickly, so it's important to create a few. Four different roles are used, **appcar_employee_role**, **app-**

car_fleet_role,**appcar_hr_role** and **appcar_admin_role**. The entity user matrix created above is used to assign the necessary permissions to the roles. In addition, a special view will be used to allow employees to SELECT on the user table without giving them access to the password column. This view is owned by the administrator and is called **APPCAR_ADMIN_APP.USERS_MGMT_VIEW**. The last step is to assign the roles to the users.

	USERNAME	GRANTED_ROLE
1	APPCAR_ADMIN_APP	APPCAR_ADMIN_ROLE
2	APPCAR_EMPLOYEE_1	APPCAR_EMPLOYEE_ROLE
3	APPCAR_EMPLOYEE_2	APPCAR_EMPLOYEE_ROLE
4	APPCAR_EMPLOYEE_3	APPCAR_EMPLOYEE_ROLE
5	APPCAR_EMPLOYEE_4	APPCAR_EMPLOYEE_ROLE
6	APPCAR_EMPLOYEE_5	APPCAR_EMPLOYEE_ROLE
7	APPCAR_FLEET_RESPONSIBLE	APPCAR_FLEET_ROLE
8	APPCAR_HR_MANAGER	APPCAR_HR_ROLE

Figure 15: Roles assigned

Before moving on to the "encryption" section, we need to insert some test data into the tables. The **insert_samples.sql** file can be used for this. In addition, the procedure allowing employees to modify the status of a vehicle is created and belongs to the administrator. The following screens demonstrate that the procedure works and that the roles and permissions are correctly configured.

	ID	PURCHASE_DATE	PURCHASE_PRICE	KILOMETRAGE	ID_MODEL_EQUIPMENT	ID_STATE
1	1	2020-01-01	40000.00	10000	1	1

Figure 16: State of vehicle 1 before calling the procedure with the employee account

	ID	PURCHASE_DATE	PURCHASE_PRICE	KILOMETRAGE	ID_MODEL_EQUIPMENT	ID_STATE
1	1	2020-01-01	40000.00	10000	1	3

Figure 17: State of vehicle 1 after calling the procedure with the employee account

5 Encryption

In order to put some additional security, it is important to encrypt critical data such as the users password in this case. The file **3-encryption.sql** is used to implement it. Firstly a new table is created in order to store the encryption keys. Then, encryption and decryption functions are used to compute the cipher value of passwords. The next step is creating procedures to encrypt / decrypt the password by calling those functions. Furthermore, a trigger is added to detect when a new line is inserted in the users table. This trigger calls the encryption procedure to prevent plain text password to be stored.

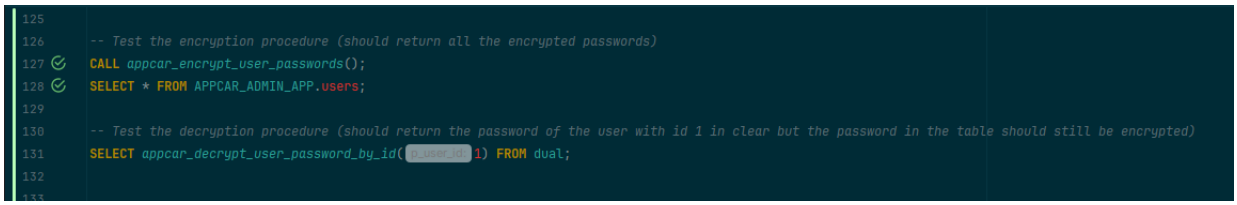


Figure 18: Calling encryption and decryption procedures

	ID	NAME	SURNAME	SEX	BIRTHDATE	PASSWORD	EMAIL
1	1	John	Doe	M	1980-01-01	5253EFA383BDF247D185A3912F52DA49	john.doe@example.com
2	2	Jane	Smith	F	1985-02-02	A99BA11EBB73CBE0CA98E816282D4CD5	jane.smith@example.com
3	3	Luck	Cena	M	1966-01-12	635E54F41F088F7982B0F9E121892657	luck123h@example.com
4	4	Mario	Bross	M	1999-12-01	33F8B1D00E644D34443298E8A2138055	mario.bross@example.com

Figure 19: Encrypted passwords

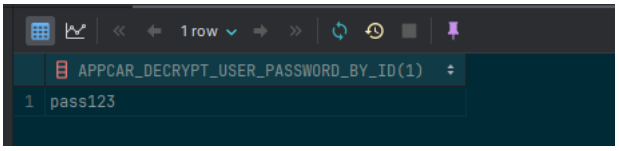


Figure 20: Uncrypted password of user 1

6 Audit

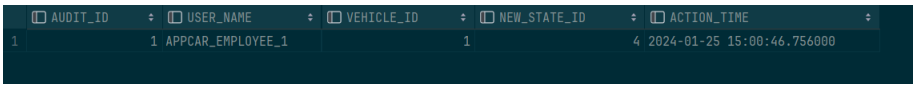
Audit is used in order to record some events that occurred in the database such as selecting or inserting data in a specific table. The **4-audit.sql** file is used in order to create standard audit, trigger audit and audit policy. The standard audit is used to log all **INSERT**, **DELETE** and **UPDATE** on the **EMPLOYEES** table. It also record all **INSERT** and **DELETE** on the **VEHICLES** table. The standard audit is also enable on the **DELETE** query of the **USERS** table and the **UPDATE** query of the **INVOICES** table. When any query fail on the **BOOKINGS** table, it is also recorded. To test the standard audit, a new employee record is added by the HR manager user, and then it is immediately deleted. The HR manager also attempted to select the booking with ID 1, resulting in a failure because he does not have the required privileges. Figure 21 shows the audit of the queries by using the query *SELECT OBJ$NAME, SQLTEXT, NTIMESTAMP# FROM aud$;*.

	OBJ$NAME	ACTION#	SQLTEXT	NTIMESTAMP#
1	EMPLOYEES	2	INSERT INTO APPCAR_HR_MANAGER.EMPLOYEES (department, id_user) VALUES ('administrative', 5)	2024-01-25 12:26:44.432000
2	EMPLOYEES	7	DELETE FROM APPCAR_HR_MANAGER.EMPLOYEES WHERE id_user = 5	2024-01-25 12:31:23.954000
3	BOOKINGS	3	SELECT * FROM APPCAR_ADMIN_APP.BOOKINGS WHERE id = 1	2024-01-25 12:35:51.205000

Figure 21: Standard audit result

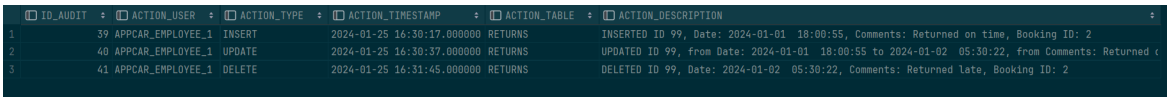
Concerning, trigger audit, a new table is created (**APPCAR_AUDIT_LOG_PROC_STATES**) for auditing when employees call the **appcar_proc_state** pro-

cedure to modify the status of a vehicle. The trigger **appcar_audit_trigger_proc_states** is added to store the name of the employee calling the procedure, the timestamp, the id of the impacted vehicle and the id of the new state into the new audit table. As figure 22 shows, employee 1 called the procedure by giving state 4 (In maintenance) to vehicle 1. Another trigger audit is used in order to audit the **RETURNS** table (except when using the **SELECT** query on it). The audit table created is called **APPCAR_AUDIT_LOG_RETURNS**. Figure 23 show the logs of the **INSERT**, **UPDATE** and **DELETE** queries on this table.



	AUDIT_ID	USER_NAME	VEHICLE_ID	NEW_STATE_ID	ACTION_TIME
1	1	APPCAR_EMPLOYEE_1		1	4 2024-01-25 15:00:46.756000

Figure 22: Trigger audit of the APPCAR_AUDIT_LOG_PROC_STATES procedure



	ID_AUDIT	ACTION_USER	ACTION_TYPE	ACTION_TIMESTAMP	ACTION_TABLE	ACTION_DESCRIPTION
1	39	APPCAR_EMPLOYEE_1	INSERT	2024-01-25 16:30:17.000000	RETURNS	INSERTED ID 99, Date: 2024-01-01 18:00:55, Comments: Returned on time, Booking ID: 2
2	40	APPCAR_EMPLOYEE_1	UPDATE	2024-01-25 16:30:37.000000	RETURNS	UPDATED ID 99, from Date: 2024-01-01 18:00:55 to 2024-01-02 05:30:22, from Comments: Returned
3	41	APPCAR_EMPLOYEE_1	DELETE	2024-01-25 16:31:45.000000	RETURNS	DELETED ID 99, Date: 2024-01-02 05:30:22, Comments: Returned late, Booking ID: 2

Figure 23: Trigger audit of the RETURNS table

Finally, an audit policy is created in order to audit the **UPDATE** query of the **COMMENTS** column from the **CHECK_IN** table. The figure 24 shows the audit result when an employee update the **COMMENTS** column.

Output SYS.DBA_FGA_AUDIT_TRAIL

	DB_USER	USERHOST	OS_USER	TIMESTAMP	OBJECT_NAME	SQL_TEXT	STATEMENT_TYPE
1	APPCAR_EMPLOYEE_1	DESKTOP-HEJLMNP	Thomas Vanhaeren	2024-01-25 19:36:29	CHECK_IN	UPDATE APPCAR_ADMIN_APP.CHECK_IN SET comments.	UPDATE

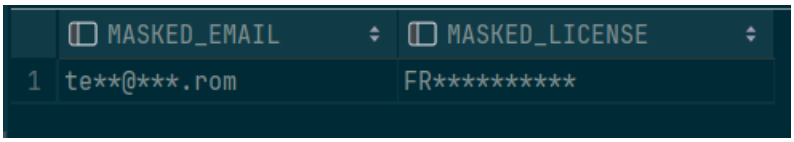
Figure 24: Audit policy result

7 Data masking

The **5-masking.sql** file is useful because it can be used in order to mask sensitive data when exporting it such as the **EMAIL** column of the **USERS** table. The pattern used only keeps the two first character and the domain of the emails while the rest is replaced by the "*" character. The same logic is applied for the **LICENSE** column, only the first two character are kept.

```
SELECT
    appcar_masking_pkg.mask_email( email: 'test@test.rom') AS masked_email,
    appcar_masking_pkg.mask_license( license: 'FR4578961123') AS masked_license
FROM dual;
```

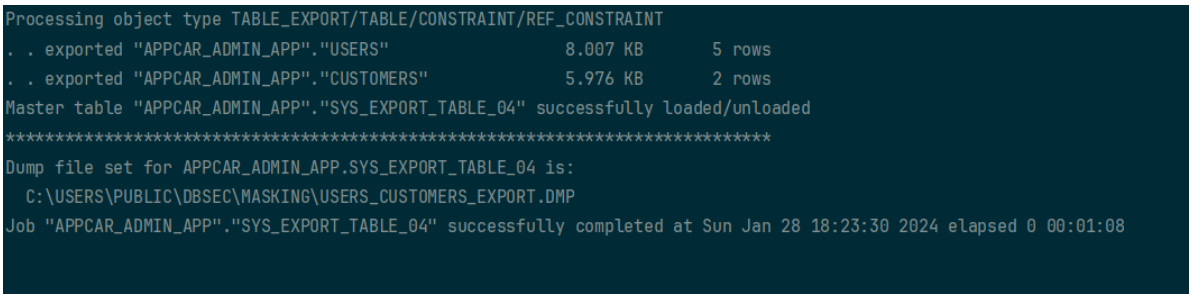
Figure 25: Query in order to call masking procedures



	MASKED_EMAIL	MASKED_LICENSE
1	te**@***.rom	FR**********

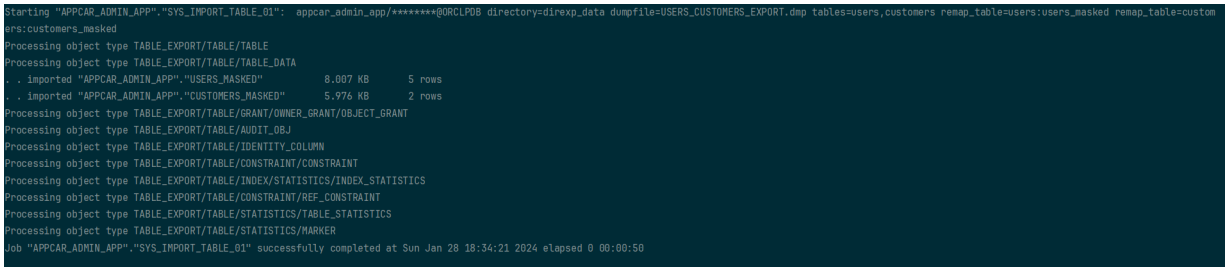
Figure 26: Result of the masked query

In order to export the data the **expdp** command can be used in a regular terminal (see the file for to see the parameters used). In the same way the **impdp** is used to import the masked data into the database. Figure 27 shows the successful export of the data using the masking functions. Figures 28 and 29 show the result of the import command.



```
Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/REF_CONSTRAINT
. . exported "APPCAR_ADMIN_APP"."USERS"                  8.007 KB       5 rows
. . exported "APPCAR_ADMIN_APP"."CUSTOMERS"              5.976 KB       2 rows
Master table "APPCAR_ADMIN_APP"."SYS_EXPORT_TABLE_04" successfully loaded/unloaded
******************************************************************************
Dump file set for APPCAR_ADMIN_APP.SYS_EXPORT_TABLE_04 is:
  C:\USERS\PUBLIC\DBSEC\MASKING\USERS_CUSTOMERS_EXPORT.DMP
Job "APPCAR_ADMIN_APP"."SYS_EXPORT_TABLE_04" successfully completed at Sun Jan 28 18:23:30 2024 elapsed 0 00:01:08
```

Figure 27: Export result



```
Starting "APPCAR_ADMIN_APP"."SYS_IMPORT_TABLE_01":  appcar_admin_app/********@ORCLPDB directory=direxp_data dumpfile=USERS_CUSTOMERS_EXPORT.dmp tables=users,customers remap_table=users:users_masked remap_table=custom
ers:customers_masked
Processing object type TABLE_EXPORT/TABLE/TABLE
Processing object type TABLE_EXPORT/TABLE/TABLE_DATA
. . imported "APPCAR_ADMIN_APP"."USERS_MASKED"          8.007 KB       5 rows
. . imported "APPCAR_ADMIN_APP"."CUSTOMERS_MASKED"      5.976 KB       2 rows
Processing object type TABLE_EXPORT/TABLE/GRANT/OWNER_GRANT/OBJECT_GRANT
Processing object type TABLE_EXPORT/TABLE/AUDIT_OBJ
Processing object type TABLE_EXPORT/TABLE/IDENTITY_COLUMN
Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
Processing object type TABLE_EXPORT/TABLE/INDEX/STATISTICS/INDEX_STATISTICS
Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/REF_CONSTRAINT
Processing object type TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
Processing object type TABLE_EXPORT/TABLE/STATISTICS/MARKER
Job "APPCAR_ADMIN_APP"."SYS_IMPORT_TABLE_01" successfully completed at Sun Jan 28 18:34:21 2024 elapsed 0 00:00:50
```

Figure 28: Import result

Figure 29: Imported masked data

8 Database application and context

The last part of the project is the security and the context of the application using the file **6-security_context.sql**. A context is set to prevent the insertion, deletion or modification of data outside opening hours for the check in and returns table. As figure 30 shows, when an employee tries to insert a new row after 10pm, he receives an error message, but if he tries again between 8am and 10pm, the error message disappears and the new data is added to the table as shown on figure 31.

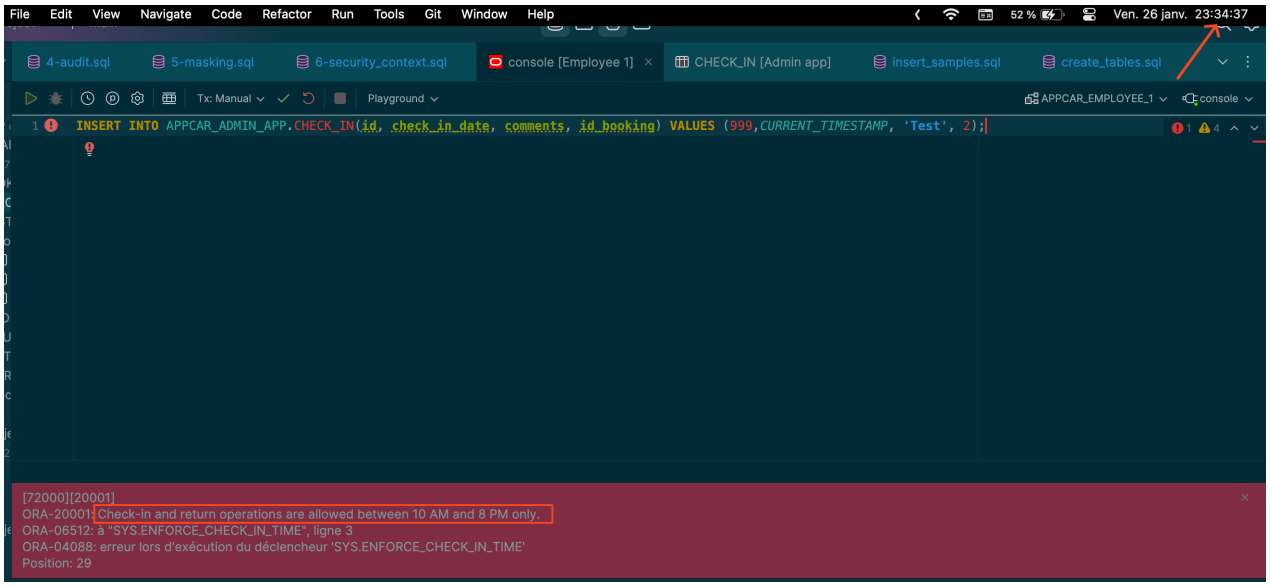


Figure 30: Failed insert due to bad timing

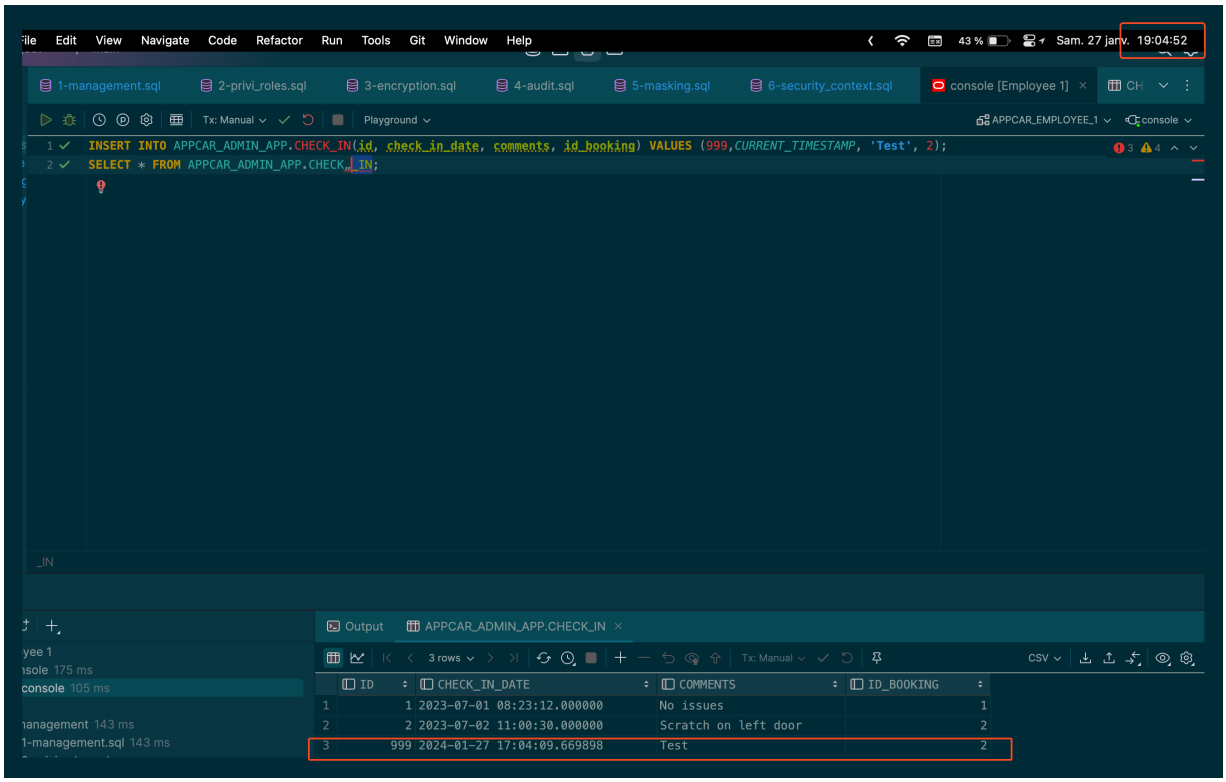


Figure 31: Success insert

Another important aspect of security is protection against SQL injections, which can occur when a user has direct access to the query. To illustrate the principle of sql injection, a vulnerable procedure has been created to take a customer's name as a parameter and search for reservations and invoice amount. Figure 32 shows the result of the procedure when it is called with a valid input. Figure 33 shows a successful exploitation of the vulnerable procedure by fetching all the user passwords with the following payload *'Nonexistent' UNION SELECT 'EMAIL: ', email , 'PASSWORD: ', password, 0, DATE '1970-01-01', DATE '1970-01-01', 0.0 FROM APPCAR_ADMIN_APP.USERS –'*. In order to fix the procedure, a bind variable is used with the **USING** clause. As shown on figure 34, by using the same malicious payload, the attacker is not able to get any password because the procedure does not render anything while figure 35 shows the normal behaviour of the fixed procedure.

```
APPCAR_ADMIN_APP> CALL APPCAR_ADMIN_APP.get_customer_bookings('Luck')
[2024-01-28 15:38:26] completed in 25 ms
Luck Cena luck123@example.com ABC123 - Booking ID: 1 - Starting date: 01/07/23 - Ending date: 10/07/23 - Total price: 5500
Luck Cena luck123@example.com ABC123 - Booking ID: 3 - Starting date: 13/10/23 - Ending date: 11/01/24 - Total price:
```

Figure 32: Normal execution of the vulnerable procedure

```
APPCAR_ADMIN_APP> CALL APPCAR_ADMIN_APP.get_customer_bookings('Nonexistent'' UNION SELECT ''EMAIL: '', email , ''PASSWORD: '', password, 0, DATE ''1970-01-01'', DATE ''19
[2024-01-28 15:39:54] completed in 75 ms
EMAIL:  john.doe@example.com PASSWORD:  C15068EFE8F059972A13AC6E4160E1EC - Booking ID: 0 - Starting date: 01/01/70 - Ending date: 01/01/70 - Total price: 0
EMAIL:  jane.smith@example.com PASSWORD:  3C0377E566D32F67419DC8A2EB3A7867 - Booking ID: 0 - Starting date: 01/01/70 - Ending date: 01/01/70 - Total price: 0
EMAIL:  luck123@example.com PASSWORD:  A36CCA2306F9AB1B7590B60D91371509 - Booking ID: 0 - Starting date: 01/01/70 - Ending date: 01/01/70 - Total price: 0
EMAIL:  mario.bross@example.com PASSWORD:  621F498FE16AFAA7D1ACA052C2BC686D - Booking ID: 0 - Starting date: 01/01/70 - Ending date: 01/01/70 - Total price: 0
```

Figure 33: Exploited procedure

```
APPCAR_ADMIN_APP> CALL APPCAR_ADMIN_APP.get_customer_bookings_mitigate('Nonexistent'' UNION SELECT ''EMAIL: '', email , ''PASSWORD: '', password, 0, DATE ''1970-01-01'',
[2024-01-28 15:47:43] completed in 22 ms
```

Figure 34: Result of the fixed procedure when using SQLi

```
APPCAR_ADMIN_APP> CALL APPCAR_ADMIN_APP.get_customer_bookings_mitigate('Luck')
[2024-01-28 15:48:30] completed in 27 ms
Luck Cena luck123@example.com ABC123 - Booking ID: 1 - Starting date: 01/07/23 - Ending date: 10/07/23 - Total price: 5500
Luck Cena luck123@example.com ABC123 - Booking ID: 3 - Starting date: 13/10/23 - Ending date: 11/01/24 - Total price:
```

Figure 35: Normal behaviour of the fixed procedure