Database Security Project

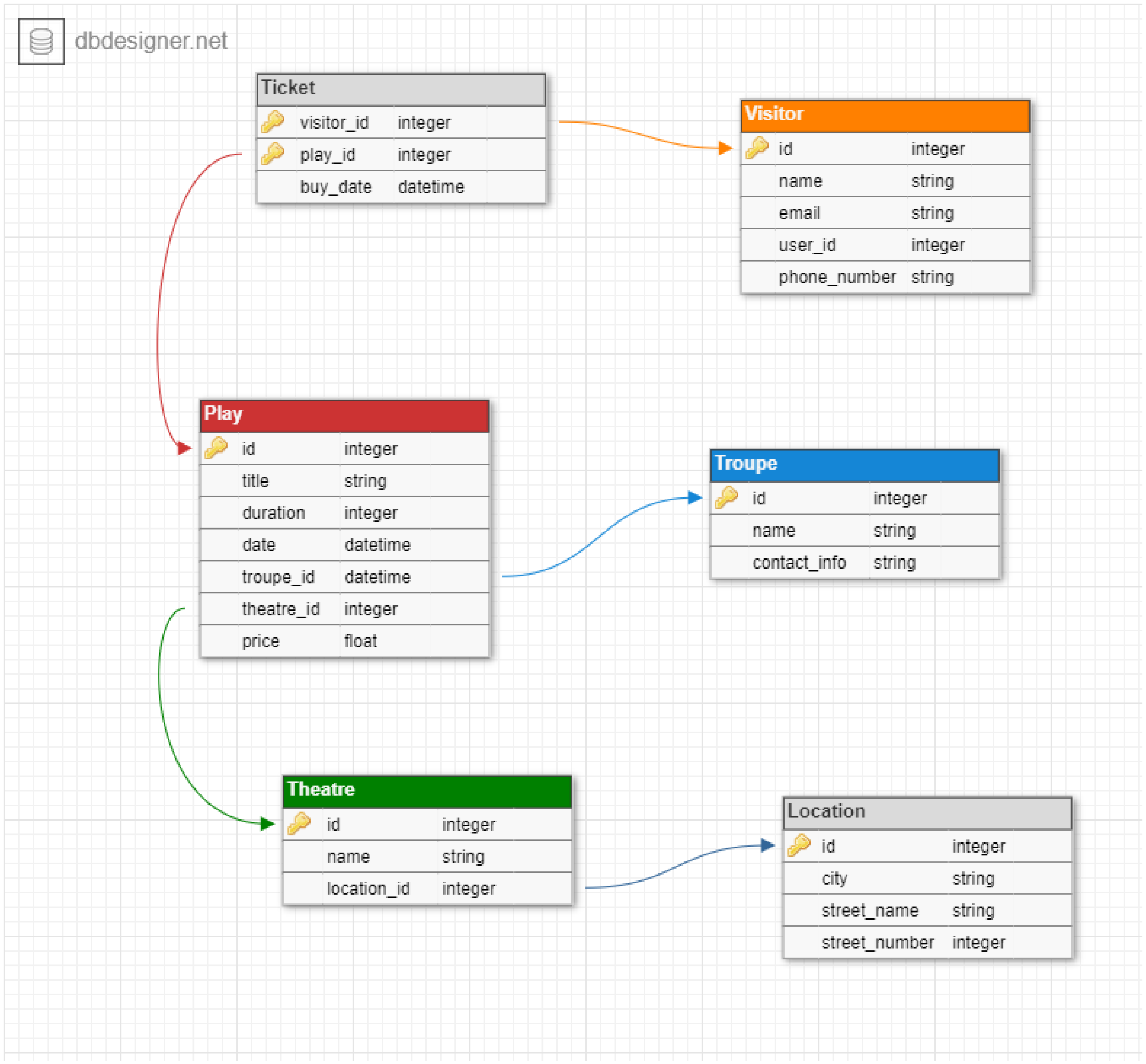
Pop Simona, 510

(1) The Conceptual, Logical and Physical Data Model

The relational model described in the current report illustrates how a fictional theatre franchise manages its business (consisting of ticket sales and organizing performances). The franchise's database is designed to store relevant information about the member theatres, the locations they are in, the collaborating troupes of actors, the plays they perform there, the theatergoers, and their purchased tickets.

Besides the database administrator, three different types of database users can be distinguished, as follows:

- the franchise customers, who are the theatergoers, referred to as "visitors", and use their accounts to purchase tickets to plays of interest. They can register themselves (which implies adding their personal information to the database) and further update their profile. They are invisible to other visitors and only have access to their own personal info, including the tickets they purchased. They can, of course, view all the theaters and plays, as well as buy or cancel tickets.
- one theater manager, who administers all the member theaters and decides which plays are going to be performed, when, and in which theatre. They have full access to the theaters, locations, and plays, along with power over them. Moreover, they are able to see all the data stored about the troupes in order to contact them for certain plays. They keep evidence of the purchased tickets and are able to cancel them.
- one troupe director, who takes care of all the troupes collaborating with the franchise. They have visibility over theaters and their locations, and they can schedule, modify, or cancel any play.



VISITOR(id#, name, email, user_id, phone_number)

PLAY(id#, title, duration, date, troupe_id, theatre_id, price)

THEATRE(id#, name, location_id)

LOCATION(id#, city, street_name, street_number)

TROUPE(id#, name, contact_info)

TICKET(visitor_id#, play_id#, buy_date)

The script run for creating and populating the database is saved under the name “Pop_Simona_510-creare_inserare.sql”.

Additionally, below can be found the security measures applied to the model that ensure the system’s security:

- Access privileges are granted according to the role of each user, as explained earlier.
- Protection against SQL Injection is provided by opting for parameterized queries.
- The GDPR regulation is respected due to the encryption of the registered visitors’ sensitive information (email address and phone number), such that no user other than the account titular has the right to view it in clear text. There are two different AES keys per user (one assigned to the email and one assigned to the phone number), which are stored inside table CRYPTO_KEYS(id#, email_key, phone_key, user_id).
- All actions on tables (select, insert, update, delete) are audited.
- An audit trigger gets activated each time the price of a play is modified in order for the database administrator to track unauthorized changes.
- An audit policy is enabled to monitor changes made to the date of a play performance.

(2) The Encryption and Decryption of the Data

As specified in the previous section, the GDPR policy of this fictional franchise stipulates that the contact information of the theatregoers (email address and phone number) must only be visible to themselves since it is considered sensitive data. Thus, when a new visitor is registered, a pair of symmetrical keys is generated, and it is further used to encrypt the two aforementioned fields but only decrypt them for the corresponding database user.

The creation of the symmetrical keys table:

```
CREATE TABLE c##dba_admin.CRYPTO_KEYS(
    id NUMBER NOT NULL,
    email_key RAW(16),
    phone_key RAW(16),
    user_id NUMBER,
    CONSTRAINT crypto_keys_pk PRIMARY KEY(id)
);

CREATE SEQUENCE c##dba_admin.CRYPTO_KEYS_SEQ;

CREATE trigger c##dba_admin.bi_CRYPTO_KEYS_ID
  before insert on c##dba_admin.CRYPTO_KEYS
  for each row
begin
  select c##dba_admin.CRYPTO_KEYS_SEQ.nextval into :NEW.id from dual;
end;

/
```

The encryption procedure:

```
CREATE OR REPLACE PROCEDURE encrypt_email_phone(email in VARCHAR2,
phone_number in VARCHAR2, encrypted_email out RAW, encrypted_phone out RAW)
IS
   cuid VARCHAR2(50) := SYS_CONTEXT('USERENV', 'SESSION_USERID'); --current user id
   email_key RAW(16);
   raw_email RAW(1801);
   phone_key RAW(16);
   raw_phone RAW(100);
   operation_mode BINARY_INTEGER;
   symmetrical_keys c##dba_admin.CRYPTO_KEYS%ROWTYPE;
BEGIN
   BEGIN
```

```
        email_key := SYS_CONTEXT('visitor_context', 'email_key');
        phone_key := SYS_CONTEXT('visitor_context', 'phone_key');

        IF(email_key IS NULL OR phone_key IS NULL) THEN
            EXECUTE IMMEDIATE 'SELECT * FROM c##dba_admin.CRYPTO_KEYS WHERE
user_id = ' || cuid INTO symmetrical_keys;

            email_key := symmetrical_keys.email_key;
            phone_key := symmetrical_keys.phone_key;
        END IF;
    EXCEPTION WHEN NO_DATA_FOUND THEN
        email_key := dbms_crypto.randombytes(16);
        phone_key := dbms_crypto.randombytes(16);
        EXECUTE IMMEDIATE 'INSERT INTO c##dba_admin.CRYPTO_KEYS VALUES(-1, :1, :2, :3)'
USING email_key, phone_key, cuid;
        COMMIT;
    END;
    operation_mode := DBMS_CRYPTO.CHAIN_CBC + DBMS_CRYPTO.ENCRYPT_AES128 +
DBMS_CRYPTO.PAD_PKCS5;
    raw_email := UTL_I18N.STRING_TO_RAW(email, 'AL32UTF8');
    encrypted_email := DBMS_CRYPTO.ENCRYPT(raw_email, operation_mode, email_key);
    raw_phone := UTL_I18N.STRING_TO_RAW(phone_number, 'AL32UTF8');
    encrypted_phone := DBMS_CRYPTO.ENCRYPT(raw_phone, operation_mode, phone_key);
END;

/

GRANT EXECUTE ON encrypt_email_phone TO c##visitor_role;
/
```

The decryption procedure:

```
CREATE OR REPLACE PROCEDURE decrypt_email_phone(encrypted_email in RAW,
encrypted_phone in RAW, decrypted_email out VARCHAR2,

    decrypted_phone out VARCHAR2)

IS

    cuid VARCHAR2(50) := SYS_CONTEXT('USERENV', 'SESSION_USERID'); --current user id

    email_key RAW(16);

    phone_key RAW(16);

    operation_mode BINARY_INTEGER;

    symmetrical_keys c##dba_admin.CRYPTO_KEYS%ROWTYPE;

BEGIN

    email_key := SYS_CONTEXT('visitor_context', 'email_key');

    phone_key := SYS_CONTEXT('visitor_context', 'phone_key');


    IF(email_key IS NULL OR phone_key IS NULL) THEN

        EXECUTE IMMEDIATE 'SELECT * FROM c##dba_admin.CRYPTO_KEYS WHERE user_id =
' || cuid INTO symmetrical_keys;


        email_key := symmetrical_keys.email_key;

        phone_key := symmetrical_keys.phone_key;

    END IF;


    operation_mode := DBMS_CRYPTO.CHAIN_CBC + DBMS_CRYPTO.ENCRYPT_AES128 +
DBMS_CRYPTO.PAD_PKCS5;


    decrypted_email := UTL_I18N.RAW_TO_CHAR(DBMS_CRYPTO.DECRYPT(encrypted_email,
operation_mode, email_key), 'AL32UTF8');

    decrypted_phone :=
UTL_I18N.RAW_TO_CHAR(DBMS_CRYPTO.DECRYPT(encrypted_phone, operation_mode,
phone_key), 'AL32UTF8');
```

```
EXCEPTION WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20002, 'Nonexistent visitor.');
END;

/

GRANT EXECUTE ON decrypt_email_phone TO c##visitor_role;
/
```

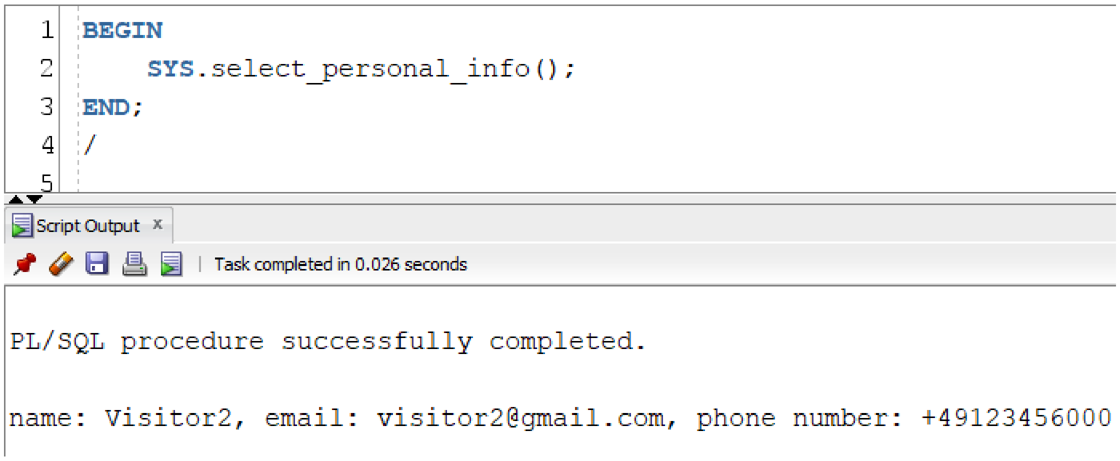
The procedure that displays the decrypted data to the corresponding user:

```
CREATE OR REPLACE PROCEDURE select_personal_info
IS
    cuid VARCHAR2(50) := SYS_CONTEXT('USERENV', 'SESSION_USERID');
    params_query  VARCHAR2(200);
    user_data c##dba_admin.VISITOR%ROWTYPE;
    email VARCHAR2(1024);
    phone_number VARCHAR2(1024);
BEGIN
    params_query := 'SELECT * FROM c##dba_admin.VISITOR WHERE "user_id" = ' ||  cuid;
    EXECUTE IMMEDIATE params_query INTO user_data;
    decrypt_email_phone(user_data."email", user_data."phone_number", email,
phone_number);
    dbms_output.put_line('name: ' || user_data."name" || ', email: ' || email || ', phone
number: ' || phone_number);
EXCEPTION WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20002, 'Nonexistent visitor.');
END;
/
```

```
GRANT EXECUTE ON select_personal_info TO c##visitor_role;
/
```

As a result, while the theatre visitor is viewing their personal data in clear [picture 1], any other user with access on the VISITOR table is seeing it encrypted [picture 2].

[picture 1]



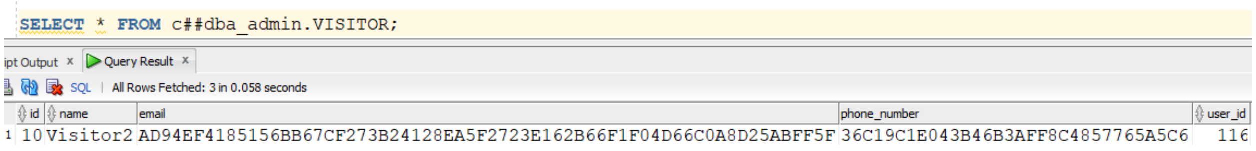
```
BEGIN
    SYS.select_personal_info();
END;
/
```

Script Output

Task completed in 0.026 seconds

```
PL/SQL procedure successfully completed.

name: Visitor2, email: visitor2@gmail.com, phone number: +49123456000
```

[picture 2]



```
SELECT * FROM c##dba_admin.VISITOR;
```

ipt Output | Query Result

SQL | All Rows Fetched: 3 in 0.058 seconds

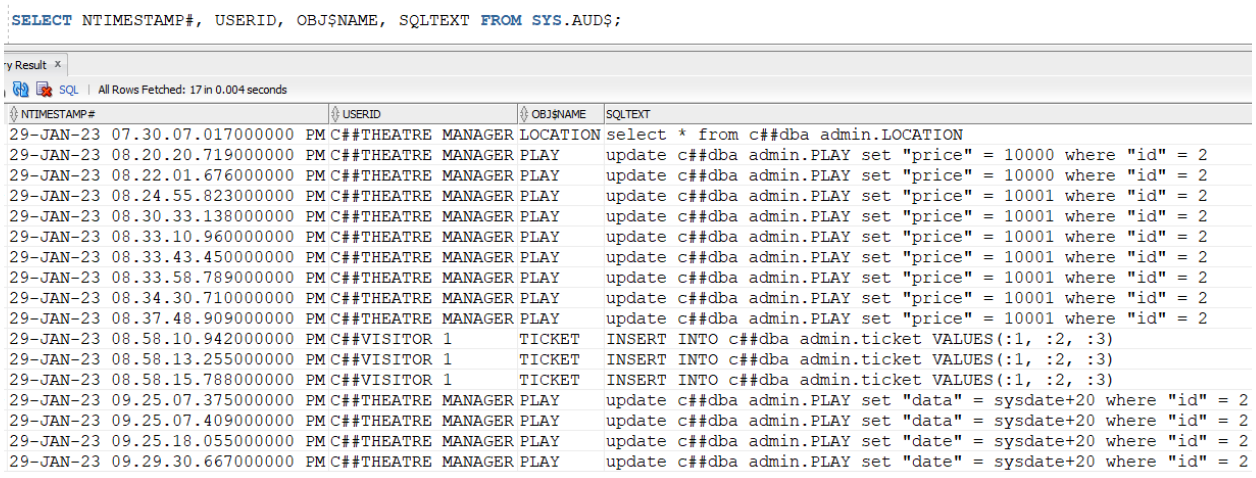
	id	name	email	phone_number	user_id
1	10	Visitor2	AD94EF4185156BB67CF273B24128EA5F2723E162B66F1F04D66C0A8D25ABFF5F	36C19C1E043B46B3AFF8C4857765A5C6	116

(3) Audit: Standard, Trigger, Policy

- The standard audit tracks all the select actions and configuration changes affecting any table in the schema.

```
ALTER SYSTEM SET audit_trail=db, EXTENDED SCOPE=spfile;
```

```
AUDIT SELECT, INSERT, UPDATE, DELETE ON c##dba_admin.VISITOR;

AUDIT SELECT, INSERT, UPDATE, DELETE ON c##dba_admin.LOCATION;

AUDIT SELECT, INSERT, UPDATE, DELETE ON c##dba_admin.THEATRE;

AUDIT SELECT, INSERT, UPDATE, DELETE ON c##dba_admin.TROUPE;

AUDIT SELECT, INSERT, UPDATE, DELETE ON c##dba_admin.TICKET;

AUDIT SELECT, INSERT, UPDATE, DELETE ON c##dba_admin.PLAY;
```



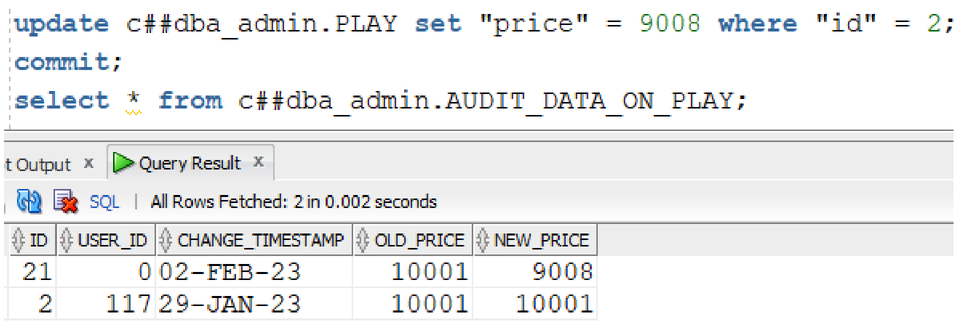
```
SELECT NTIMESTAMP#, USERID, OBJ$NAME, SQLTEXT FROM SYS.AUD$;
```

NTIMESTAMP#	USERID	OBJ$NAME	SQLTEXT
29-JAN-23 07.30.07.017000000 PM	C##THEATRE MANAGER	LOCATION	select * from c##dba admin.LOCATION
29-JAN-23 08.20.20.719000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "price" = 10000 where "id" = 2
29-JAN-23 08.22.01.676000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "price" = 10000 where "id" = 2
29-JAN-23 08.24.55.823000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "price" = 10001 where "id" = 2
29-JAN-23 08.30.33.138000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "price" = 10001 where "id" = 2
29-JAN-23 08.33.10.960000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "price" = 10001 where "id" = 2
29-JAN-23 08.33.43.450000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "price" = 10001 where "id" = 2
29-JAN-23 08.33.58.789000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "price" = 10001 where "id" = 2
29-JAN-23 08.34.30.710000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "price" = 10001 where "id" = 2
29-JAN-23 08.37.48.909000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "price" = 10001 where "id" = 2
29-JAN-23 08.58.10.942000000 PM	C##VISITOR 1	TICKET	INSERT INTO c##dba admin.ticket VALUES(:1, :2, :3)
29-JAN-23 08.58.13.255000000 PM	C##VISITOR 1	TICKET	INSERT INTO c##dba admin.ticket VALUES(:1, :2, :3)
29-JAN-23 08.58.15.788000000 PM	C##VISITOR 1	TICKET	INSERT INTO c##dba admin.ticket VALUES(:1, :2, :3)
29-JAN-23 09.25.07.375000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "data" = sysdate+20 where "id" = 2
29-JAN-23 09.25.07.409000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "data" = sysdate+20 where "id" = 2
29-JAN-23 09.25.18.055000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "date" = sysdate+20 where "id" = 2
29-JAN-23 09.29.30.667000000 PM	C##THEATRE MANAGER	PLAY	update c##dba admin.PLAY set "date" = sysdate+20 where "id" = 2

- An audit trigger was created to monitor the changes made to the prices of the plays.

```
CREATE TABLE c##dba_admin.AUDIT_DATA_ON_PLAY(

    id NUMBER NOT NULL,

    user_id NUMBER NOT NULL,

    change_timestamp DATE NOT NULL,

    old_price FLOAT,

    new_price FLOAT,

    CONSTRAINT audit_pk PRIMARY KEY(id)

);
```

```
CREATE SEQUENCE c##dba_admin.AUDIT_DATA_ON_PLAY_SEQ;

CREATE OR REPLACE TRIGGER c##dba_admin.BI_AUDIT_DATA_ON_PLAY_ID
  before insert on c##dba_admin.AUDIT_DATA_ON_PLAY
  for each row
begin
  select c##dba_admin.AUDIT_DATA_ON_PLAY_SEQ.nextval into :NEW.id from dual;
end;
/

CREATE OR REPLACE TRIGGER monitoring_price_change
AFTER UPDATE OF "price" ON c##dba_admin.PLAY
for each row
when (SYS_CONTEXT('USERENV', 'AUTHENTICATED_IDENTITY') != 'C##DBA_ADMIN')
begin
  insert into c##dba_admin.AUDIT_DATA_ON_PLAY values(-1, SYS_CONTEXT('USERENV',
'SESSION_USERID'), SYSDATE, :old."price", :new."price");
end;
/
```



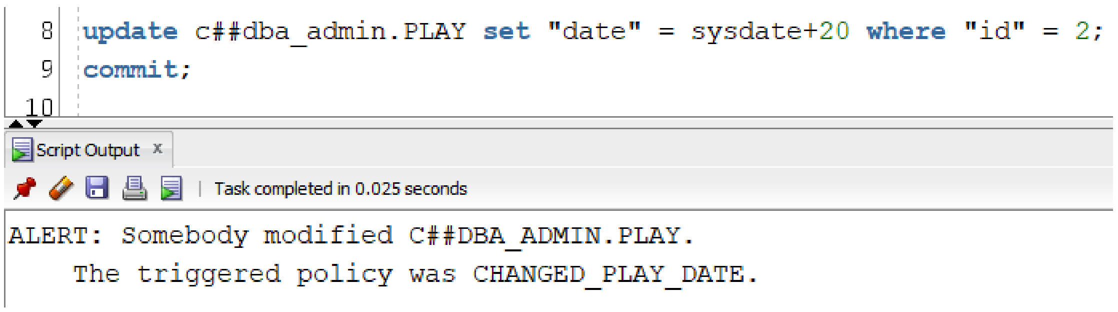
```
update c##dba_admin.PLAY set "price" = 9008 where "id" = 2;
commit;
select * from c##dba_admin.AUDIT_DATA_ON_PLAY;
```

All Rows Fetched: 2 in 0.002 seconds

ID	USER_ID	CHANGE_TIMESTAMP	OLD_PRICE	NEW_PRICE
21	0	02-FEB-23	10001	9008
2	117	29-JAN-23	10001	10001

- Finally, an audit policy is configured to issue an alert each time the date of a performance is modified.

```
CREATE OR REPLACE PROCEDURE audit_alert (object_schema VARCHAR2, object_name
VARCHAR2, policy_name VARCHAR2)

AS

BEGIN

   dbms_output.put_line('ALERT: Somebody modified ' || object_schema || '.' ||
object_name || '.

   The triggered policy was ' || policy_name || '.');

END;

/

CREATE OR REPLACE PROCEDURE audit_policy_on_play AS

BEGIN

   dbms_fga.add_policy (

      object_schema => 'C##DBA_ADMIN',

      object_name => 'PLAY',

      policy_name => 'CHANGED_PLAY_DATE',

      statement_types => 'UPDATE',

      audit_column  => '"date"',

      ENABLE => TRUE,

      handler_module => 'audit_alert');

END;

/


EXECUTE audit_policy_on_play;

/
```



```
update c##dba_admin.PLAY set "date" = sysdate+20 where "id" = 2;
commit;
```

```
ALERT: Somebody modified C##DBA_ADMIN.PLAY.
    The triggered policy was CHANGED_PLAY_DATE.
```

(4) Database Users and Computational Resources

The following processes are defined:

A – Register or configure visitor

B – View visitor info

C – View and cancel personal ticket

D – View THEATRE

E – Configure THEATRE

F – View TROUPE

G – Configure TROUPE

H – View LOCATION

I – Configure LOCATION

J – View PLAY

K – Configure PLAY

L – View and cancel TICKET

M – Modify TICKET

N – Buy ticket

[user – process matrix]

Process	DBA Admin	Troupe Director	Theatre Manager	Visitor
A	X			X
B	X			X
C	X			X
D	X	X	X	X
E	X		X	
F	X	X	X	X
G	X	X		
H	X	X	X	X
I	X		X	
J	X	X	X	X
K	X	X	X	
L	X		X	
M	X			
N	X			X

[entity – process matrix]

Process	VISITOR	THEATRE	TROUPE	LOCATION	PLAY	TICKET
A	I,U,D,S					
B	S					
C						D,S
D		S				
E		I,U,D				
F			S			
G			I,U,D			
H				S		
I				I,U,D		
J					S	
K					I,U,D	
L						D,S
M						U
N						I

[entity – user matrix]

	DBA Admin	Troupe Director	Theatre Manager	Visitor

VISITOR	I,U,D,S		S	I,U,D,S
THEATRE	I,U,D,S	S	I,U,D,S	S
TROUPE	I,U,D,S	I,U,D,S	S	S
LOCATION	I,U,D,S	S	I,U,D,S	S
PLAY	I,U,D,S	I,U,D,S	I,U,D,S	S
TICKET	I,U,D,S		D,S	I,D,S

The implementation of identity configuration management in the database:

```
CREATE PROFILE c##visitor_profile LIMIT
   SESSIONS_PER_USER 2
   PASSWORD_LIFE_TIME 30
   FAILED_LOGIN_ATTEMPTS 10;

CREATE PROFILE c##manager_profile LIMIT
   SESSIONS_PER_USER 5
   PASSWORD_LIFE_TIME 30
   FAILED_LOGIN_ATTEMPTS 10;

CREATE PROFILE c##admin_profile LIMIT
   SESSIONS_PER_USER 20
   FAILED_LOGIN_ATTEMPTS 5;

CREATE USER c##dba_admin IDENTIFIED BY dba_admin QUOTA UNLIMITED ON USERS
PROFILE c##admin_profile;
CREATE USER c##theatre_manager IDENTIFIED BY theatre_manager QUOTA UNLIMITED ON
USERS PROFILE c##manager_profile;
CREATE USER c##troupe_director IDENTIFIED BY troupe_director QUOTA UNLIMITED ON
USERS PROFILE c##manager_profile;
CREATE USER c##visitor_1 IDENTIFIED BY visitor_1 QUOTA UNLIMITED ON USERS PROFILE
c##visitor_profile;
```

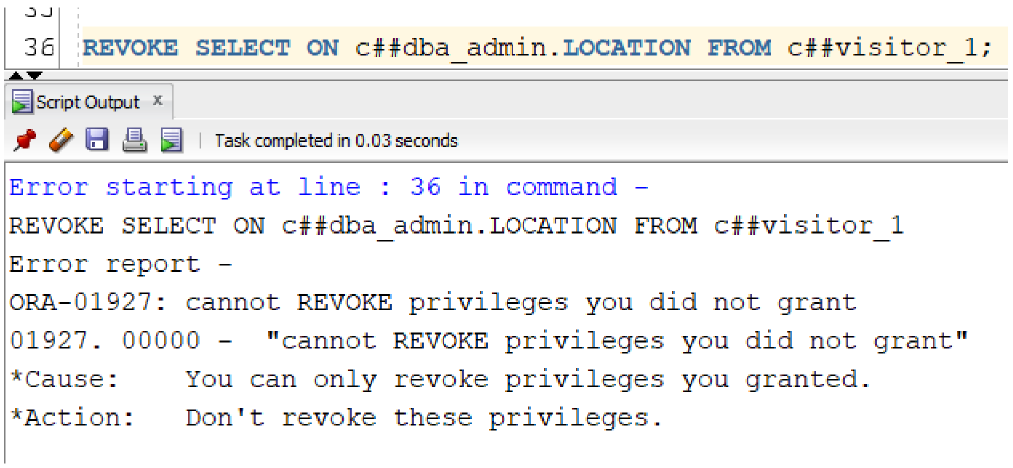
```
CREATE USER c##visitor_2 IDENTIFIED BY visitor_2 QUOTA UNLIMITED ON USERS PROFILE
c##visitor_profile;
```

(5) Roles and Privileges

- At this point, roles are defined, privileges are granted to them, and then they are assigned to the users previously created.

```
GRANT CREATE SESSION TO c##dba_admin;

GRANT CREATE SESSION TO c##theatre_manager;

GRANT CREATE SESSION TO c##troupe_director;

GRANT CREATE SESSION TO c##visitor_1;

GRANT CREATE SESSION TO c##visitor_2;


CREATE ROLE c##dba_admin_role;

CREATE ROLE c##theatre_manager_role;

CREATE ROLE c##troupe_director_role;

CREATE ROLE c##visitor_role;


GRANT SELECT ON c##dba_admin.THEATRE TO c##visitor_role;

GRANT SELECT ON c##dba_admin.LOCATION TO c##visitor_role;

GRANT SELECT ON c##dba_admin.PLAY TO c##visitor_role;

GRANT SELECT ON c##dba_admin.TROUPE TO c##visitor_role;


GRANT SELECT, UPDATE, DELETE, INSERT ON c##dba_admin.TROUPE TO
c##troupe_director_role;

GRANT SELECT, UPDATE, DELETE, INSERT ON c##dba_admin.THEATRE TO
c##theatre_manager_role;

GRANT SELECT ON c##dba_admin.TROUPE TO c##theatre_manager_role;
GRANT SELECT ON c##dba_admin.THEATRE TO c##troupe_director_role;


GRANT SELECT, UPDATE, DELETE, INSERT ON c##dba_admin.PLAY TO
c##troupe_director_role;

GRANT SELECT, UPDATE, DELETE, INSERT ON c##dba_admin.PLAY TO
c##theatre_manager_role;


GRANT SELECT, DELETE ON c##dba_admin.TICKET TO c##theatre_manager_role;

GRANT SELECT, UPDATE, DELETE, INSERT ON c##dba_admin.TICKET TO
c##dba_admin_role;

GRANT SELECT, UPDATE, DELETE, INSERT ON c##dba_admin.LOCATION TO
c##theatre_manager;


GRANT c##dba_admin_role TO c##dba_admin;

GRANT c##theatre_manager_role TO c##theatre_manager;

GRANT c##troupe_director_role TO c##troupe_director;

GRANT c##visitor_role TO c##visitor_1;

GRANT c##visitor_role TO c##visitor_2;
```

- With regards to the hierarchy of privileges, it should be noted that, within this particular model, privileges are granted only through roles and not directly to the users (since the schema was conceived such that users under the same role are allowed to perform exactly the same actions). Therefore, no privilege on the schema can be revoked directly from a user (example below).



```
REVOKE SELECT ON c##dba_admin.LOCATION FROM c##visitor_1;
```

Script Output ×

Task completed in 0.03 seconds

```
Error starting at line : 36 in command -
REVOKE SELECT ON c##dba_admin.LOCATION FROM c##visitor_1
Error report -
ORA-01927: cannot REVOKE privileges you did not grant
01927. 00000 -  "cannot REVOKE privileges you did not grant"
*Cause:    You can only revoke privileges you granted.
*Action:   Don't revoke these privileges.
```

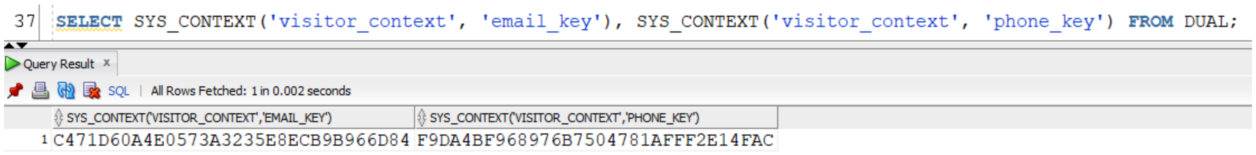
- On the privileges being granted over dependent objects, it can be noted that, even though the visitor type users don't have access on the CRYPTO_KEYS table, they can make use of the data inside this table by calling the encryption related procedures (register_visitor and select_personal_info).

(6) Data Applications and Data Security

- To optimize the sensitive data encryption process, a new application's context is built: the need for querying the CRYPTO_KEYS table each time a user interacts with their data is replaced with storing the user corresponding symmetrical keys within the application's context and always having them at hand for data encryption/ decryption.

```
CREATE CONTEXT visitor_context USING get_crypto_keys;

CREATE OR REPLACE PROCEDURE get_crypto_keys
IS
   cuid VARCHAR2(50) := SYS_CONTEXT('USERENV', 'SESSION_USERID'); --current user id
   symmetrical_keys c##dba_admin.crypto_keys%ROWTYPE;
```

```
BEGIN
   EXECUTE IMMEDIATE 'SELECT * FROM c##dba_admin.crypto_keys WHERE user_id = ' || cuid INTO symmetrical_keys;
   DBMS_SESSION.SET_CONTEXT('visitor_context', 'email_key', symmetrical_keys.email_key);
   DBMS_SESSION.SET_CONTEXT('visitor_context', 'phone_key', symmetrical_keys.phone_key);
EXCEPTION WHEN NO_DATA_FOUND THEN
   DBMS_SESSION.SET_CONTEXT('visitor_context', 'email_key', NULL);
   DBMS_SESSION.SET_CONTEXT('visitor_context', 'phone_key', NULL);
END;
/

CREATE OR REPLACE TRIGGER set_visitor_context_trigger
AFTER INSERT OR UPDATE ON c##dba_admin.crypto_keys
BEGIN
   get_crypto_keys();
END;
/
```



```
SELECT SYS_CONTEXT('visitor_context', 'email_key'), SYS_CONTEXT('visitor_context', 'phone_key') FROM DUAL;
```

Query Result

All Rows Fetched: 1 in 0.002 seconds

	SYS_CONTEXT('VISITOR_CONTEXT','EMAIL_KEY')	SYS_CONTEXT('VISITOR_CONTEXT','PHONE_KEY')
1	C471D60A4E0573A3235E8ECB9B966D84	F9DA4BF968976B7504781AFFF2E14FAC

- As brought up in the beginning, the use of parametrized queries annihilates the risk of SQL Injection attacks. It can be observed below that every procedure called by the users relies on the current user id parameter (which cannot be controlled by users themselves).

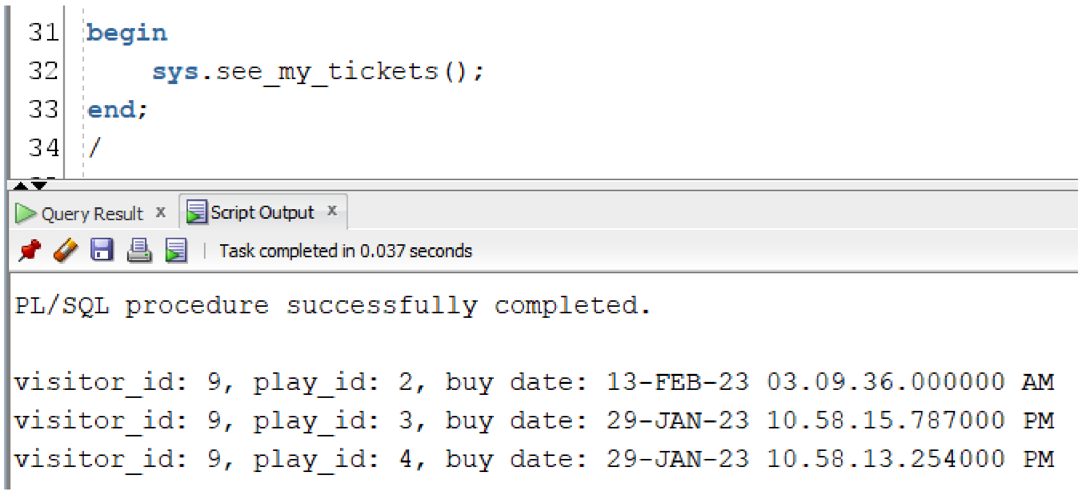
The procedure that allows theatregoers to register themselves:

```
CREATE OR REPLACE PROCEDURE register_visitor(username in VARCHAR2,
    email in VARCHAR2, phone_number in VARCHAR2)
IS
    cuid VARCHAR2(50) := SYS_CONTEXT('USERENV', 'SESSION_USERID'); --current user id
    already_found VARCHAR2(2);
    params_query  VARCHAR2(200);
    encrypted_email RAW(1896);
    encrypted_phone RAW(128);
BEGIN
    EXECUTE IMMEDIATE 'SELECT DECODE(COUNT(*), 0, :1, :2) FROM c##dba_admin.visitor
WHERE "user_id" = :3' INTO already_found
        USING 'N', 'Y', cuid;

    IF (already_found = 'N') THEN
        params_query := 'INSERT INTO c##dba_admin.visitor VALUES(-1, :1, :2, :3, :4)';
    ELSE
        params_query := 'UPDATE c##dba_admin.visitor SET "name" = :1, "email" = :2,
"phone_number" = :3 WHERE "user_id" = :4';
    END IF;
    encrypt_email_phone(email, phone_number, encrypted_email, encrypted_phone);
    EXECUTE IMMEDIATE params_query USING username, encrypted_email,
encrypted_phone, cuid;
    COMMIT;
END;
/

GRANT EXECUTE ON register_visitor TO c##visitor_role;
/
```

The procedures that enable a theatregoer to buy, cancel, respectively see their purchased tickets:

```
CREATE OR REPLACE PROCEDURE buy_ticket(play_id in NUMBER)
IS
   current_user_id VARCHAR2(50) := SYS_CONTEXT('USERENV', 'SESSION_USERID');
   params_query  VARCHAR2(200);
   user_pk NUMBER;
BEGIN
   EXECUTE IMMEDIATE 'SELECT "id" from c##dba_admin.visitor WHERE "user_id" = ' ||
current_user_id INTO user_pk;
   params_query := 'INSERT INTO c##dba_admin.ticket VALUES(:1, :2, :3)';
   EXECUTE IMMEDIATE params_query USING user_pk, play_id, current_timestamp;
   COMMIT;
END;
/

GRANT EXECUTE ON buy_ticket TO c##visitor_role;
/

CREATE OR REPLACE PROCEDURE cancel_ticket(play_id in NUMBER)
IS
   current_user_id VARCHAR2(50) := SYS_CONTEXT('USERENV', 'SESSION_USERID');
   params_query  VARCHAR2(200);
BEGIN
   params_query := 'DELETE FROM c##dba_admin.ticket WHERE "visitor_id" = (SELECT "id"
from c##dba_admin.visitor WHERE "user_id" = :1)
      and "play_id" = :2';
   EXECUTE IMMEDIATE params_query USING current_user_id, play_id;
   COMMIT;
END;
/
```

```
GRANT EXECUTE ON cancel_ticket TO c##visitor_role;
/

CREATE OR REPLACE PROCEDURE see_my_tickets
IS
   current_user_id VARCHAR2(50) := SYS_CONTEXT('USERENV', 'SESSION_USERID');
BEGIN
   FOR ticket IN (SELECT * FROM c##dba_admin.TICKET WHERE "visitor_id" = (SELECT "id"
from c##dba_admin.VISITOR WHERE "user_id" = current_user_id))
   LOOP
      DBMS_OUTPUT.PUT_LINE('visitor_id: ' || ticket."visitor_id" || ', play_id: ' ||
ticket."play_id" || ', buy date: ' ||
         to_char(ticket."buy_date"));
   END LOOP;
END;
/

GRANT EXECUTE ON see_my_tickets TO c##visitor_role;
/
```



```
begin
    sys.see_my_tickets();
end;
/
```

```
PL/SQL procedure successfully completed.

visitor_id: 9, play_id: 2, buy date: 13-FEB-23 03.09.36.000000 AM
visitor_id: 9, play_id: 3, buy date: 29-JAN-23 10.58.15.787000 PM
visitor_id: 9, play_id: 4, buy date: 29-JAN-23 10.58.13.254000 PM
```

(7) Data Masking

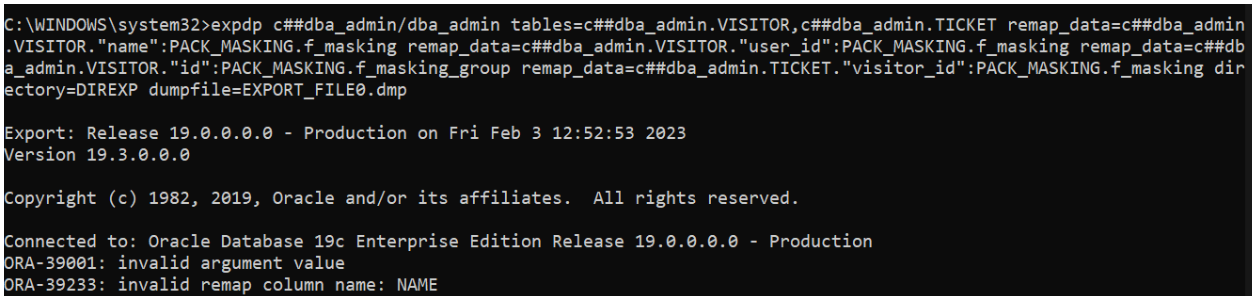
Visitors’ data is aimed to be masked as follows:

```
CREATE OR REPLACE PACKAGE PACK_MASKING IS
   FUNCTION f_masking(str VARCHAR2) RETURN VARCHAR2;
   FUNCTION f_masking(nb NUMBER) RETURN NUMBER;
   FUNCTION f_masking_group(nb NUMBER) RETURN NUMBER;
END;
/

CREATE OR REPLACE PACKAGE BODY PACK_MASKING IS
   TYPE t_tab_ind IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
   v_tab_ind t_tab_ind;

   FUNCTION f_masking(str VARCHAR2) RETURN VARCHAR2 IS
      v_str VARCHAR2(255);
      v_len NUMBER;
   BEGIN
      v_str := SUBSTR(str,1,1);
      SELECT LENGTH(str) INTO v_len FROM DUAL;
      v_str := RPAD(v_str, v_len, '#');
      RETURN v_str;
   END f_masking;
```

```
  FUNCTION f_masking(nb NUMBER) RETURN NUMBER IS
    v_len NUMBER;
    v_min NUMBER;
    v_max NUMBER;
    v_seed VARCHAR2(100);
    v_new_nb NUMBER;
  BEGIN
    IF v_tab_ind.EXISTS(nb) THEN
      RETURN v_tab_ind(nb);
    ELSE
      v_len := LENGTH(to_char(nb));
      v_min := to_number(RPAD(SUBSTR(to_char(nb),1,1),v_len,'0'));
      v_max := to_number(rpad(substr(to_char(nb),1,1),v_len,'9'));
      v_seed := TO_CHAR(SYSTIMESTAMP,'YYYYDDMMHH24MISSFFFF');
      v_new_nb := round(DBMS_RANDOM.VALUE(LOW => v_min, HIGH =>
v_max),0);
      v_tab_ind(nb):=v_new_nb;
      RETURN v_new_nb;
    END IF;
  END f_masking;

  FUNCTION f_masking_group(nb NUMBER) RETURN NUMBER IS
    v_new_nb NUMBER;
    v_len NUMBER;
  BEGIN
    v_len:=LENGTH(to_char(nb));
```

```
        v_new_nb:=to_number(RPAD(SUBSTR(to_char(nb),1,1),v_len,'0'));
        RETURN v_new_nb;
    END f_masking_group;
END;
/

CREATE OR REPLACE DIRECTORY DIREXP AS 'C:\SECBD';
/



declare
  f utl_file.file_type;
begin
  f := utl_file.fopen ('C:\SECBD', 'test.txt', 'w');
  utl_file.put_line(f, 'test');
  utl_file.fclose(f);
end;
/

GRANT ALL PRIVILEGES TO c##dba_admin;
GRANT READ, WRITE ON DIRECTORY DIREXP TO c##dba_admin;
/
```

However, an error is being issued when trying to export the masked data.



```
C:\WINDOWS\system32>expdp c##dba_admin/dba_admin tables=c##dba_admin.VISITOR,c##dba_admin.TICKET remap_data=c##dba_admin
.VISITOR."name":PACK_MASKING.f_masking remap_data=c##dba_admin.VISITOR."user_id":PACK_MASKING.f_masking remap_data=c##db
a_admin.VISITOR."id":PACK_MASKING.f_masking_group remap_data=c##dba_admin.TICKET."visitor_id":PACK_MASKING.f_masking dir
ectory=DIREXP dumpfile=EXPORT_FILE0.dmp

Export: Release 19.0.0.0.0 - Production on Fri Feb 3 12:52:53 2023
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle and/or its affiliates.  All rights reserved.

Connected to: Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
ORA-39001: invalid argument value
ORA-39233: invalid remap column name: NAME
```