**Student Projects Chapter 8 - Implementing Security Features for Student Projects**

Read the sample project steps for this chapter and apply the same techniques to the student project that you are developing. Use the normalized purely relational database you created at the end of chapter 6.

Step 8.1 - Create a value-independent view that hides some private information.

The view will be of the Customer table joined to Zip table to show city and state attributes as well, but without the driver’s license number, reference name, or adSeen.

-- Create a value-independent view that hides some private information

CREATE VIEW customerview AS

SELECT

c.custId,

c.firstName,

c.lastName,

c.street,

z.city,

z.state,

c.zip,

c.phoneArea,

c.phoneNumber

FROM customer as c

JOIN zips as z

ON c.zip = z.zip;

-- Test view

SELECT \* FROM customerview;

Figure 1 - Creating the view

A screenshot of a social media post

Description automatically generated

Figure 2 – Testing the view

A screenshot of a cell phone

Description automatically generated

Step 8.2 - Create a user and authorize that person to read the view. Begin an authorization graph.

-- Create a user and authorize that person to read the view

CREATE USER 'U\_001' IDENTIFIED BY '12345';

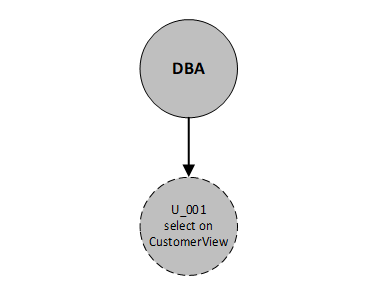
GRANT SELECT ON CustomerView TO 'U\_001';

Figure 3 – Creating user and grant privileges to select on customerview

A screenshot of a cell phone

Description automatically generated

Figure 4 - Beginning of the authorization graph



Note: U\_001 was granted just to select ON CustomerView view without grant option. So the arrow from DBA to U\_001 is a single arrow which means not granting a grant option, and the shape of U\_001 has a dashed outline which means it has not received a grant option.

Step 8.3 – Create and authorize four other users to access and/or modify various parts of the database and update the authorization graph.

-- Create and authorize four other users to access and/or modify various parts of the database

CREATE USER 'U\_002' IDENTIFIED BY '12345';

CREATE USER 'U\_003' IDENTIFIED BY '12345';

CREATE USER 'U\_004' IDENTIFIED BY '12345';

CREATE USER 'U\_005' IDENTIFIED BY '12345';

GRANT SELECT ON sale TO 'U\_002';

GRANT ALL PRIVILEGES ON financing TO 'U\_003';

GRANT SELECT, UPDATE, INSERT, DELETE ON newcar TO 'U\_004' WITH GRANT OPTION;

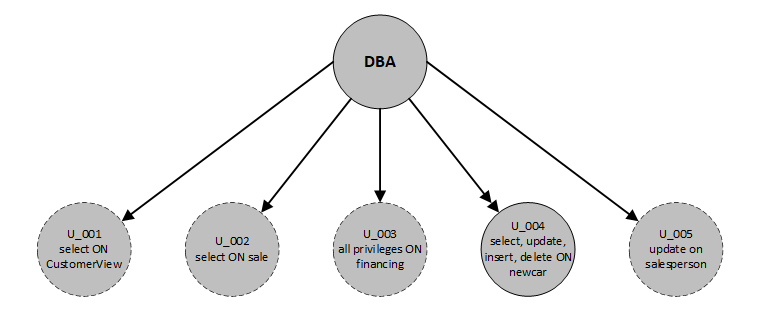
GRANT UPDATE ON salesperson TO 'U\_005';

Figure 5 - Creating users and granting privileges

A screenshot of a social media post

Description automatically generated

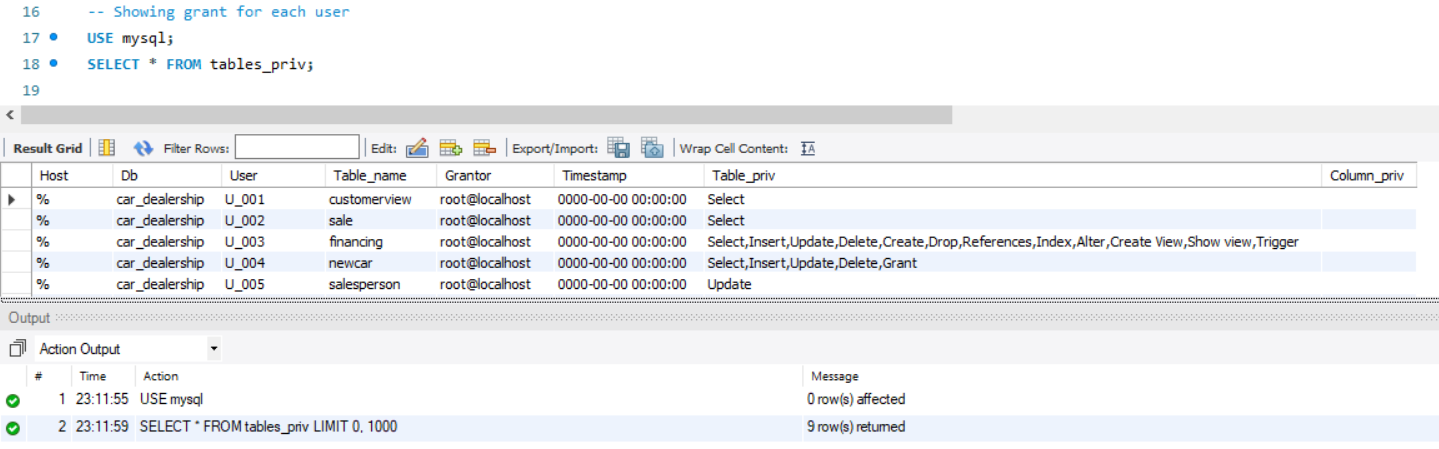
Figure 6 - Updating the authorization graph



Note: U\_001 was granted just to select ON CustomerView view, U\_002 was granted just select ON sale table, U\_003 was granted all privileges ON financing table, and U\_005 was granted just update ON salesperson. All of them without grant option. So the arrows from DBA to U\_001, U\_002, U\_003 and U\_005 are single arrows which mean not granting a grant option, and the shape of them has a dashed outline which means it has not received a grant option.

U\_004 was granted select, update, insert and delete ON newcar with grant option. So the arrow from DBA to U\_004 is double arrow which mean granting a grant option, and the shape of it has a solid outline which means it has received a grant option.

Figure 7 – Showing grant for each user:



Step 8.4 - Set up an audit trail trigger for updates to a sensitive item that users can update.

-- Set up an audit trail trigger for updates to a sensitive item that users can update

SET @username := user();

-- Create the table for the audit trail

CREATE TABLE newcar\_price\_audit (

dateofChange DATE,

userChanged VARCHAR(50),

IDofNewCar INT(6),

oldPrice DECIMAL(10,2),

newPrice DECIMAL(10,2));

-- Create trigger

DELIMITER %

CREATE TRIGGER NewCarPriceAuditTrail

BEFORE UPDATE ON newcar

FOR EACH ROW

BEGIN

INSERT INTO newcar\_price\_audit VALUES (

NOW(),

@username,

OLD.newCarId,

OLD.listPrice,

NEW.listPrice);

END%

DELIMITER ;

-- Testing the audit trail trigger

SELECT \* FROM newcar WHERE newCarId = 3;

SELECT \* FROM newcar\_price\_audit;

UPDATE car\_dealership.newcar SET listPrice = '24300.00' WHERE (newCarId = 3);

SELECT \* FROM newcar\_price\_audit;

Figure 8 - Creating a table to use in the audit trail trigger and setting up the audit trail trigger

A screenshot of a social media post

Description automatically generated

Note: Every time an user updates the price of a new car (listPrice), one roll is created in the table newcar\_price\_audit with the date of change, the user who changed, the id of the new car, the old price and the new price.

Step 8.5 – Write and execute a SQL statement to demonstrate that the trigger is working as expected.

1. Selecting the row that we will change to show its values, and showing the created table which is empty and which will be affected when we use the audit trail trigger
2. Updating value of the selected row and showing its new value, and showing the row created in the new table after updating the value of selected row

-- Testing the audit trail trigger

-- 1)

SELECT \* FROM newcar WHERE newCarId = 3;

SELECT \* FROM newcar\_price\_audit;

-- 2)

UPDATE car\_dealership.newcar SET listPrice = '24300.00' WHERE (newCarId = 3);

SELECT \* FROM newcar WHERE newCarId = 3;

SELECT \* FROM newcar\_price\_audit;

Figure 9:

a) Selecting the row that we will change to show its values (listPrice = 22300.00)

b) Selecting the table newcar\_price\_audit to show that it is empty

A picture containing screenshot

Description automatically generated

Figure 10:

a) Updating the row that we have shown in Figure 1 to new price = 24300.00

b) Selecting the row that we have changed to show its values (new listPrice = 24300.00)

c) Then, selecting the table newcar\_price\_audit to show that it was created a roll with the date of price change, the user who changed it, the id of the new car, the old price and the new price

A screenshot of a social media post

Description automatically generated