-- Demonstrates granting and revoking privileges

-- As Carter, try show databases
SHOW DATABASES;

-- As root, grant SELECT privileges on only the analysis view in rideshare
GRANT SELECT ON `rideshare`.`analysis` TO 'carter';

-- As Carter, try showing databases
SHOW DATABASES;
USE `rideshare`;
SELECT * FROM `analysis`;

-- As Carter, cannot view rides table
SELECT * FROM `rides`;

-- As Carter, try to create a new view
CREATE VIEW `destinations` AS
SELECT `destination` FROM `analysis`;

-- As root, grant create view privileges
GRANT CREATE VIEW ON `rideshare`.* TO 'carter';

-- Succeed in creating a view
CREATE VIEW `destinations` AS
SELECT `destination` FROM `analysis`;
CREATE DATABASE IF NOT EXISTS `rideshare`;
USE `rideshare`;

CREATE TABLE `rides` (  
  `id` INT AUTO_INCREMENT,  
  `origin` VARCHAR(64) NOT NULL,  
  `destination` VARCHAR(64) NOT NULL,  
  `rider` VARCHAR(16) NOT NULL,  
  PRIMARY KEY(`id`)  
);

INSERT INTO `rides` (`origin`, `destination`, `rider`)  
VALUES  
('Good Egg Galaxy', 'Honeyhive Galaxy', 'Peach'),  
('Castle Courtyard', 'Cascade Kingdom', 'Mario'),  
('Metro Kingdom', 'Mushroom Kingdom', 'Luigi'),  
('Seaside Kingdom', 'Deep Woods', 'Bowser');

CREATE VIEW `analysis` AS  
SELECT `id`, `origin`, `destination`  
FROM `rides`;
-- Demonstrates creating a new user
CREATE USER 'carter' IDENTIFIED BY 'password';
-- Demonstrates SQL injection attacks

-- Execute statement with non-malicious input
SELECT * FROM `accounts`
WHERE `id` = 1;

-- Execute statement with malicious input
SELECT * FROM `accounts`
WHERE `id` = 1 UNION SELECT * FROM `accounts`;

injection/injection.sql
-- Demonstrates prepared statements

-- Create a prepared statement
PREPARE `balance_check`
FROM 'SELECT * FROM `accounts`
WHERE `id` = ?';

-- Execute the prepared statement with non-malicious input
-- Set a user variable
SET @id = 1;
EXECUTE `balance_check` USING @id;

-- Execute the prepared statement with malicious input
-- Set a user variable
SET @id = '1 UNION SELECT * FROM `accounts`';
EXECUTE `balance_check` USING @id;
CREATE DATABASE IF NOT EXISTS 'bank';
USE 'bank';

CREATE TABLE 'accounts' (
    'id' INT AUTO_INCREMENT,
    'name' VARCHAR(16),
    'balance' INT,
    PRIMARY KEY('id')
);

INSERT INTO 'accounts' ('name', 'balance')
VALUES
('Alice', 10),
('Bob', 20),
('Charlie', 30);
-- Demonstrates expanded ALTER TABLE support in MySQL (vs. SQLite)
-- https://www.sqlite.org/omitted.html

ALTER TABLE `stations`
MODIFY `line` ENUM('blue', 'green', 'orange', 'red', 'silver') NOT NULL;
-- Demonstrates loading data from a CSV file

-- Load a CSV of riders into the riders table
LOAD DATA INFILE 'red.csv' INTO TABLE `stations`
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
-- Demonstrates types in MySQL

-- Create and use database
CREATE DATABASE IF NOT EXISTS 'mbta';
USE 'mbta';

-- Riders are represented by cards
CREATE TABLE `cards` (  
    `id` INT AUTO_INCREMENT,  
    PRIMARY KEY(`id`)  );

DESCRIBE `cards`;

-- Stations have a name and a line
CREATE TABLE `stations` (  
    `id` INT AUTO_INCREMENT,  
    `name` VARCHAR(32) NOT NULL UNIQUE,  
    `line` ENUM('blue', 'green', 'orange', 'red') NOT NULL,  
    PRIMARY KEY(`id`)  );

DESCRIBE `stations`;

-- Cards can swipe at a station, in order to enter, exit, or deposit funds
CREATE TABLE `swipes` (  
    `id` INT AUTO_INCREMENT,  
    `card_id` INT,  
    `station_id` INT,  
    `type` ENUM('enter', 'exit', 'deposit') NOT NULL,  
    `datetime` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,  
    `amount` DECIMAL(5,2) NOT NULL CHECK(`amount` != 0),  
    PRIMARY KEY(`id`),  
    FOREIGN KEY(`station_id`) REFERENCES `stations`(`id`),  
    FOREIGN KEY(`card_id`) REFERENCES `cards`(`id`)  );

DESCRIBE `swipes`;

SHOW TABLES;
-- Demonstrates stored procedures

-- Set up database
USE `mfa`;

-- Add deleted column to columns
ALTER TABLE `collections`
ADD COLUMN `deleted` TINYINT DEFAULT 0;

-- Create a stored procedure to view all (non-deleted) items in collections
delimiter //
CREATE PROCEDURE `current_collections`()
BEGIN
SELECT `title`, `accession_number`, `acquired` FROM `collections` WHERE `deleted` = 0;
END//
delimiter ;

-- Call the stored procedure
CALL `current_collections`();

-- Set an item to be deleted
UPDATE `collections` SET `deleted` = 1
WHERE `title` = 'Farmers working at dawn';

-- Call stored procedure again
CALL `current_collections`();

-- Create a table to log artwork transactions
CREATE TABLE `transactions`
(`id` INT AUTO_INCREMENT,
`title` VARCHAR(64) NOT NULL,
`action` ENUM('bought', 'sold') NOT NULL,
PRIMARY KEY(`id`))
;

-- Create a stored procedure with a parameter to mark artwork sold
delimiter //
CREATE PROCEDURE `sell`(IN `sold_id` INT)
BEGIN
UPDATE `collections` SET `deleted` = 1 WHERE `id` = `sold_id`;
INSERT INTO `transactions`(`title`, `action`) VALUES ((SELECT `title` FROM `collections` WHERE `id` = `sold_id`), 'sold');
END;
delimiter

-- Sell a piece of artwork
CALL `sell`((
  SELECT `id` FROM `collections`
  WHERE `title` = 'Farmers working at dawn'
));

-- Show results
SELECT * FROM `transactions`;
SELECT * FROM `collections`;

-- Improve procedure to handle corner cases
DROP PROCEDURE `sell`;

-- Create a stored procedure to handle case where item is already deleted
delimiter /
CREATE PROCEDURE `sell`(`IN` `sold_id` INT)
BEGIN
  IF `sold_id` IN (SELECT `id` FROM `collections` WHERE `deleted` = 0 ) THEN
    UPDATE `collections` SET `deleted` = 1 WHERE `id` = `sold_id`;
    INSERT INTO `transactions` (`title`, `action`) VALUES ((SELECT `title` FROM `collections` WHERE `id` = `sold_id`), 'sold');
  END IF;
END;
delimiter /

-- Show using status code
delimiter /
CREATE PROCEDURE `sell`(`IN` `sold_id` INT)
BEGIN
  IF `sold_id` NOT IN (SELECT `id` FROM `collections` WHERE `deleted` = 0 ) THEN
    SIGNAL SQLSTATE '45000';
  END IF;
  UPDATE `collections` SET `deleted` = 1 WHERE `id` = `sold_id`;
  INSERT INTO `transactions` (`title`, `action`) VALUES ((SELECT `title` FROM `collections` WHERE `id` = `sold_id`), 'sold');
85  END//
86  delimiter ;
-- Schema for MFA db

CREATE DATABASE IF NOT EXISTS `mfa`;
USE `mfa`;

CREATE TABLE `collections` (
    `id` INT AUTO_INCREMENT,
    `title` VARCHAR(64) NOT NULL,
    `accession_number` VARCHAR(9) NOT NULL UNIQUE,
    `acquired` DATE,
    PRIMARY KEY(`id`)
);

INSERT INTO `collections` (`title`, `accession_number`, `acquired`) VALUES
('Farmers working at dawn', '11.6152', '1911-08-03'),
('Imaginative landscape', '56.496', NULL),
('Profusion of flowers', '56.257', '1956-04-12'),
('Spring outing', '14.76', '1914-01-08');

CREATE TABLE `artists` (
    `id` INT AUTO_INCREMENT,
    `name` VARCHAR(64) NOT NULL,
    PRIMARY KEY(`id`)
);

INSERT INTO `artists` (`name`) VALUES
('Li Yin'),
('Qian Weicheng'),
('Unidentified artist'),
('Zhou Chen');

CREATE TABLE `created` (
    `artist_id` INT,
    `collection_id` INT,
    PRIMARY KEY(`artist_id`, `collection_id`),
    FOREIGN KEY(`artist_id`) REFERENCES `artists`(`id`),
    FOREIGN KEY(`collection_id`) REFERENCES `collections`(`id`)
);

INSERT INTO `created` (`artist_id`, `collection_id`)
VALUES

(SELECT 'id' FROM 'artists' WHERE 'name' = 'Li Yin'), (SELECT 'id' FROM 'collections' WHERE 'title' = 'Imaginative landscape'),

(SELECT 'id' FROM 'artists' WHERE 'name' = 'Qian Weicheng'), (SELECT 'id' FROM 'collections' WHERE 'title' = 'Profusion of flowers'),

(SELECT 'id' FROM 'artists' WHERE 'name' = 'Unidentified artist'), (SELECT 'id' FROM 'collections' WHERE 'title' = 'Farmers working at dawn'),

(SELECT 'id' FROM 'artists' WHERE 'name' = 'Zhou Chen'), (SELECT 'id' FROM 'collections' WHERE 'title' = 'Spring outing');
-- Demonstrates navigating a MySQL database

-- Start docker container
docker start mysql

-- Log in
mysql -u root -h 127.0.0.1 -P 3306 -p

-- List all databases on server
SHOW DATABASES;
-- Demonstrates navigating a PostgreSQL database

-- Start docker container
docker start postgres

-- Log in
psql postgresql://postgres@127.0.0.1:5432/postgres

-- List all databases
\l
\list

-- Create a database
CREATE DATABASE "mbta";

-- Connect to a particular database
\c
\connect

-- List all tables
\dt

-- Describe a particular table
\d "cards"

-- Quit
\q
--- Demonstrate PostgreSQL syntax

CREATE TABLE "cards" (  
  "id" SERIAL,  
  PRIMARY KEY("id") ) ;

CREATE TABLE "stations" (  
  "id" SERIAL,  
  "name" VARCHAR(32) NOT NULL UNIQUE,  
  "line" VARCHAR(32) NOT NULL,  
  PRIMARY KEY("id") ) ;

CREATE TYPE "swipe_type" AS ENUM('enter', 'exit', 'deposit');

CREATE TABLE "swipes" (  
  "id" SERIAL,  
  "card_id" INT,  
  "station_id" INT,  
  "type" "swipe_type" NOT NULL,  
  "datetime" TIMESTAMP NOT NULL DEFAULT now(),  
  "amount" NUMERIC(5,2) NOT NULL CHECK("amount" != 0),  
  PRIMARY KEY("id"),  
  FOREIGN KEY("station_id") REFERENCES "stations"("id"),  
  FOREIGN KEY("card_id") REFERENCES "cards"("id") ) ;