-- Task 01

DROP PROCEDURE IF EXISTS rentFilm;

DELIMITER //

CREATE PROCEDURE rentFilm(

IN inventory\_id MEDIUMINT,

IN customer\_id SMALLINT,

IN staff\_id TINYINT,

OUT LID INT)

BEGIN

INSERT INTO rental(inventory\_id, customer\_id, rental\_date, last\_update, return\_date, staff\_id)

VALUES (inventory\_id, customer\_id, NOW(), NOW(), NULL, staff\_id);

SET LID = LAST\_INSERT\_ID();

END //

DELIMITER ;

-- Test 01

SELECT COUNT(\*) FROM rental;

-- 16044

CALL rentFilm(367, 130, 1, @LID);

-- OK

SELECT COUNT(\*) FROM rental;

-- 16045

-- Task 02

DROP PROCEDURE IF EXISTS makePayment;

DELIMITER //

CREATE PROCEDURE makePayment(

IN v\_rental\_id INT,

OUT LID INT

)

BEGIN

DECLARE v\_rental\_rate DECIMAL(4,2) DEFAULT 0;

DECLARE v\_customer\_id SMALLINT DEFAULT 0;

DECLARE v\_staff\_id TINYINT DEFAULT 0;

SELECT

film.rental\_rate,

rental.customer\_id,

rental.staff\_id

INTO

v\_rental\_rate,

v\_customer\_id,

v\_staff\_id

FROM rental

INNER JOIN inventory ON rental.inventory\_id = inventory.inventory\_id

INNER JOIN film ON inventory.film\_id = film.film\_id

WHERE rental\_id = v\_rental\_id;

INSERT INTO payment (customer\_id, staff\_id, rental\_id, amount, payment\_date, last\_update)

VALUES (v\_customer\_id, v\_staff\_id, v\_rental\_id, v\_rental\_rate, NOW(), NOW());

SET LID = LAST\_INSERT\_ID();

END //

DELIMITER ;

-- Test 02

SELECT COUNT(\*) FROM payment;

-- 16049

CALL makePayment(16, @LID);

-- OK

SELECT COUNT(\*) FROM payment;

-- 16050

Task 3: Create a stored procedure returnFilm that records when a customer returns a film to the store. The procedure should update the correct row in the rental table by setting the return\_date to NOW().

DROP PROCEDURE IF EXISTS returnFilm;

DELIMITER //

CREATE PROCEDURE returnFilm (IN rentID INT, IN customerID SMALLINT)

BEGIN

UPDATE rental

SET return\_date = NOW()

WHERE rental\_id = rentID

AND customer\_id = customerID;

END //

DELIMITER ;

-- Test 3

SELECT \* FROM rental

WHERE rental\_id = rental\_id

AND customer\_id = customer\_id

AND return\_date is null;

-- 181

CALL returnFilm (14098, 554);

--ok

SELECT \* FROM rental

WHERE rental\_id = rental\_id

AND customer\_id = customer\_id

AND return\_date is null;

--180

-- Task 4

DROP FUNCTION IF EXISTS daysOverdue;

DELIMITER $$

CREATE FUNCTION daysOverdue(

f\_id INT,

c\_id INT

)

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE daysOverdue INT;

SELECT (DATEDIFF(return\_date, rental\_date) - rental\_duration) INTO daysOverdue

FROM customer

JOIN rental USING (customer\_id)

JOIN inventory USING (inventory\_id)

JOIN film USING (film\_id)

WHERE film\_id=f\_id and c\_id=customer\_id and rental\_duration < DATEDIFF(return\_date, rental\_date);

RETURN daysOverdue;

END $$

DELIMITER ;

SELECT daysOverdue(1,518);

-- Task 05

DROP PROCEDURE IF EXISTS payFees;

DELIMITER //

CREATE PROCEDURE payFees(

IN overdue INT,

IN v\_rental\_id INT

)

BEGIN

UPDATE payment SET

amount = amount+(overdue\*1)

WHERE rental\_id = v\_rental\_id;

END //

DELIMITER ;

-- Test 05

SELECT \* FROM payment WHERE rental\_id = 1185;

-- amount: 5.99

CALL payFees(3, 1185);

SELECT \* FROM payment WHERE rental\_id = 1185;

-- amount: 8.99

-- Task 06

DROP TRIGGER IF EXISTS rental\_after\_insert;

DELIMITER \\

CREATE TRIGGER rental\_after\_insert

AFTER INSERT ON rental

FOR EACH ROW

BEGIN

CALL makePayment(NEW.rental\_id, @LID);

END \\

DELIMITER ;

--- Task 07

DROP TRIGGER IF EXISTS rental\_after\_update;

DELIMITER \\

CREATE TRIGGER rental\_after\_update

AFTER UPDATE ON rental

FOR EACH ROW

BEGIN

CALL payFees(UPDATE.rental\_id);

END \\

DELIMITER ;

-- Task 08

-- To create the table where the information will be stored

DROP TABLE IF EXISTS overdue;

CREATE TABLE overdue(

customer\_name varchar(45),

customer\_email varchar(45),

film\_name varchar(45),

days\_overdue int(255)

);

-- To make the event works:

-- First: Drop the event if exists.

-- Second: Set the global event\_scheduler to ON.

-- Third: Fire the event.

DROP EVENT IF EXISTS listOverdues;

SET GLOBAL event\_scheduler = ON;

DELIMITER $$

CREATE EVENT listOverdues

ON SCHEDULE

EVERY 1 DAY

DO

BEGIN

INSERT INTO overdue(customer\_name, customer\_email, film\_name, days\_overdue)

SELECT CONCAT(customer.first\_name,' ',customer.last\_name) as Name, customer.email, film.title,

CONCAT(DATEDIFF(return\_date, rental\_date) - rental\_duration) AS daysOverdue

FROM customer

JOIN rental USING (customer\_id)

JOIN inventory USING (inventory\_id)

JOIN film USING (film\_id)

WHERE rental\_duration < DATEDIFF(return\_date, rental\_date);

END $$

DELIMITER ;