**1. Write a stored procedure that accepts the month and year as inputs and prints the ordernumber, orderdate and status of the orders placed in that month. The month should be abbreviated to three characters.**

**Example:**

**Input: month -> 'Feb'**

**year -> 2003**

**Output:**

**+------------+---------+**

**| orderdate | status |**

**+------------+---------+**

**| 2003-02-11 | Shipped |**

**| 2003-02-17 | Shipped |**

**| 2003-02-24 | Shipped |**

**+------------+---------+**

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetOrdersByMonthYear`(

IN orderMonth CHAR(3),

IN orderYear INT

)

BEGIN

SELECT

orderdate,

`status`

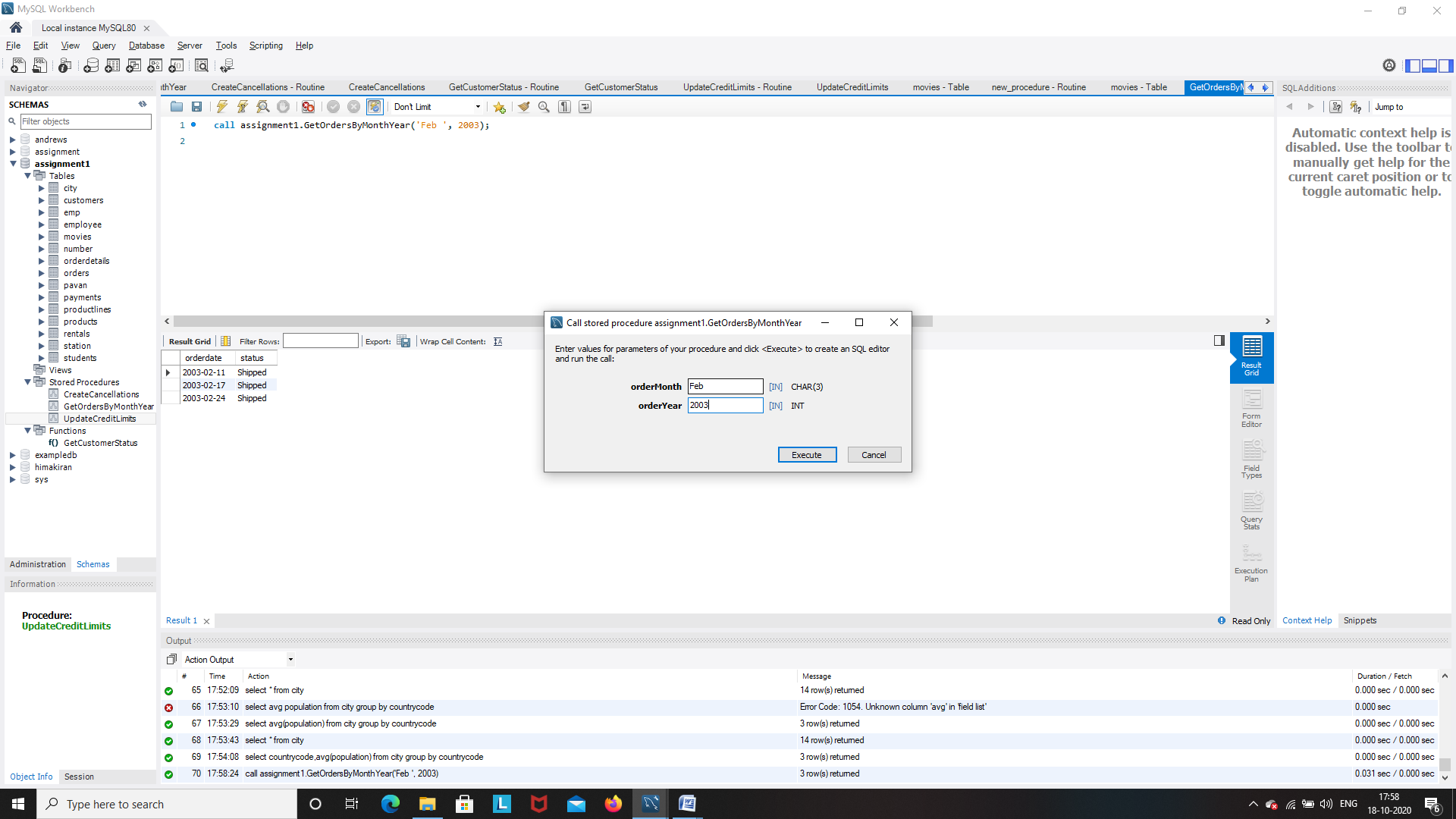
FROM

orders

WHERE

date\_format(orderDate, '%Y-%b') = concat(concat(orderYear, '-'), orderMonth);

END



|  |  |
| --- | --- |
| 2003-02-11 | Shipped |
| 2003-02-17 | Shipped |
| 2003-02-24 | Shipped |

**2. Write a stored procedure to insert a record into the cancellations table for all cancelled orders.**

**STEPS: a. Create a table called cancellations with the following fields**

**id (primary key), custumernumber (foreign key), ordernumber (foreign key), comments**

**All values except id should be taken from the order table.**

**b. Read through the orders table . If an order is cancelled, then put an entry in the cancellations table.**

CREATE TABLE cancellations (

id int auto\_increment,

customernumber int ,

ordernumber int,

PRIMARY KEY (id),

FOREIGN KEY (customernumber) REFERENCES customers(customernumber),

FOREIGN KEY (ordernumber) REFERENCES orders(orderNumber)

);

select \* from cancellations;

|  |  |  |
| --- | --- | --- |
| 1 | 448 | 10167 |
| 2 | 496 | 10179 |
| 3 | 131 | 10248 |
| 4 | 201 | 10253 |
| 5 | 357 | 10260 |
| 6 | 141 | 10262 |
| 8 | 448 | 10167 |
| 9 | 496 | 10179 |
| 10 | 131 | 10248 |
| 11 | 201 | 10253 |
| 12 | 357 | 10260 |
| 13 | 141 | 10262 |

CREATE DEFINER=`root`@`localhost` PROCEDURE `CreateCancellations`()

BEGIN

INSERT INTO cancellations (customernumber, ordernumber)

SELECT

customerNumber,

ordernumber

FROM orders

WHERE `status` = 'Cancelled';

END

**3. a. Write function that takes the customernumber as input and returns the purchase\_status based on the following criteria . [table:Payments]**

**if the total purchase amount for the customer is < 25000 status = Silver, amount between 25000 and 50000, status = Gold**

**if amount > 50000 Platinum**

CREATE FUNCTION `GetCustomerStatus`(customerNum INT) RETURNS varchar(20)

DETERMINISTIC

BEGIN

DECLARE customer\_status VARCHAR(20);

DECLARE total\_amount DECIMAL(19, 2);

SELECT sum(amount) into total\_amount

FROM

payments

WHERE

customerNumber = customerNum;

IF total\_amount > 50000 THEN

SET customer\_status = 'Platinum';

ELSEIF total\_amount > 25000 THEN

SET customer\_status = 'Gold';

ELSE

SET customer\_status = 'Silver';

END IF;

RETURN (customer\_status);

END

**assignment1.GetCustomerStatus(448)**

Platinum

**b. Write a query that displays customerid, customername and purchase\_status**

SELECT

customerNumber,

customername,

GetCustomerStatus(customerNumber)

FROM

customers;

**4. Write a stored procedure that checks the creditlimit and the purchase status of the customers.**

**If a platinum customer has crediltlimit less than 100,000 raise an exception. In the exception handler update the crediltlimit to 100000.**

**If a silver customer has creditlimit greater than 60,000 raise an exception. In the exception handler update the crediltlimit to 60000.**

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateCreditLimits`()

BEGIN

DECLARE cur\_customer\_id INT;

DECLARE cur\_credit\_limit DOUBLE(19,2);

DECLARE customer\_status VARCHAR(20);

DECLARE done INT DEFAULT false;

DECLARE raise\_platinum CONDITION FOR SQLSTATE '45000';

DECLARE reduce\_silver CONDITION FOR SQLSTATE '45001';

DECLARE cur\_customers CURSOR FOR SELECT customernumber, creditlimit FROM customers;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

DECLARE CONTINUE HANDLER FOR raise\_platinum

BEGIN

UPDATE customers SET creditlimit = 100000 WHERE customernumber = cur\_customer\_id ;

END;

DECLARE CONTINUE HANDLER FOR reduce\_silver

BEGIN

UPDATE customers SET creditlimit = 60000 WHERE customernumber = cur\_customer\_id;

END;

OPEN cur\_customers;

read\_loop: LOOP

FETCH cur\_customers INTO cur\_customer\_id, cur\_credit\_limit;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT

GETCUSTOMERSTATUS(cur\_customer\_id)

INTO customer\_status FROM

customers

WHERE customerNumber = cur\_customer\_id ;

IF customer\_status = 'Platinum' AND cur\_credit\_limit < 100000 THEN

SIGNAL SQLSTATE '45000';

ELSEIF customer\_status = 'Silver' AND cur\_credit\_limit > 60000 THEN

SIGNAL SQLSTATE '45001';

END IF;

END LOOP;

CLOSE cur\_customers;

END

**5. Replicate the functionality of 'on delete cascade' and 'on update cascade' using triggers on movies and rentals tables. Note: Both tables - movies and rentals - don't have primary or foreign keys. Use only triggers to implement the above.**

CREATE DEFINER=`root`@`localhost` TRIGGER `movies\_BEFORE\_DELETE` BEFORE DELETE ON `movies` FOR EACH ROW BEGIN

UPDATE rentals SET movieId = NULL WHERE movieId = OLD.id;

END

CREATE DEFINER=`root`@`localhost` TRIGGER `movies\_BEFORE\_UPDATE` BEFORE UPDATE ON `movies` FOR EACH ROW BEGIN

UPDATE rentals SET movieId = NEW.id WHERE movieId = OLD.id;

END