Anh Tran

CPSC-440

Triggers

Apr 14, 2021

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Trigger name** | **Event** | **Timing** | **Table** | **description** |
| Customer\_name\_to\_uppercase\_before\_insert | insert | before | Customer | This trigger will converse customer first and last name to upper case before insert to the Customer table. |
| Customer\_promotion\_update | insert | after | Ticket | This trigger will help user keep up with number of tickets each customer buy so user can offer promotion. Whoever buy more than 10 tickets may have some kind of promotion. |
| send\_message\_to\_messages\_table\_after\_insert | insert | after | Customer | This trigger inserts a message to the messages table after new customer inserted into Customer table. This message will help keep up with date and time when new customer saved to the database. |
| Customer\_backup\_before\_delete | delete | before | Customer | This trigger will store customer first and last name to customer\_backup table before user delete customer from customer table. In case they come back, we have some basic information about the customer. |
| Update\_promotion\_after\_delete\_ticket | delete | after | Customer\_promotion | This trigger will subtract ticket from current\_number\_of\_ticket from customer\_promotion table if customer return a ticket. |
| Customer\_payment\_method\_before\_update | update | before | Customer | This trigger will make sure that all payment method has to be either Cash or Credit Card before update to the Customer table. It will display error message if invalid data input. |
| Send\_message\_after\_update | update | after | Customer | This trigger will send a message to messages table display all information and date, time after that customer information updated in the data base. User can go back to the messages table to look for what they changed when needed. |

**The following tables need to create before creating all the triggers**

**Create Customer\_promotion table:**

DROP TABLE IF EXISTS Customer\_promotion;

CREATE TABLE Customer\_promotion (

promotionID INT PRIMARY KEY AUTO\_INCREMENT,

customer\_ID INT NOT NULL,

currentDate TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

current\_number\_of\_ticket int,

promotion\_approval VARCHAR(50));

**Create messages table:**

DROP TABLE IF EXIST ‘theater’.’messages’;

CREATE TABLE messages (

id int primary key auto\_increment,

message varchar(250) default null,

recorded\_time TIMESTAMP null default CURRENT\_TIMESTAMP on update CURRENT\_TIMESTAMP);

**Create customer\_backup:**

DROP TABLE IF EXIST ‘theater’.‘customer\_backup’;

CREATE TABLE customer\_backup (

id int primary key auto\_increment,

customerFirstName text(50),

customerLastName text(50));

**1/ This trigger help converse customer name to upper case before insert into the database**

DROP TRIGGER IF EXISTS `theater`.`Customer\_name\_to\_uppercase\_before\_insert`;

DELIMITER $$

USE `theater`$$

CREATE TRIGGER `theater`.`Customer\_name\_to\_uppercase\_before\_insert`

BEFORE INSERT ON `Customer` FOR EACH ROW

BEGIN

SET new.FirstName = UPPER(new.FirstName), new.LastName = UPPER(new.LastName);

END

$$

DELIMITER ;

**2/ This trigger send a notification to messages including date and time table after new customer inserted into Customer table.**

DROP TRIGGER IF EXISTS `theater`.`send\_message\_to\_messages\_table\_after\_insert`;

DELIMITER $$

USE `theater`$$

CREATE TRIGGER `theater`.`send\_message\_to\_messages\_table\_after\_insert`

AFTER INSERT ON `Customer` FOR EACH ROW

BEGIN

INSERT INTO messages(message) VALUES ('New data insert to the Customer table');

END

$$

DELIMITER ;

**3/ This trigger will create a record of customer before delete in case user needed in the future**

DROP TRIGGER IF EXISTS `theater`.`Customer\_backup\_before\_delete`;

DELIMITER $$

USE `theater`$$

CREATE TRIGGER `theater`.`Customer\_backup\_before\_delete`

BEFORE DELETE ON `Customer` FOR EACH ROW

BEGIN

INSERT INTO customer\_backup(customerFirstName, customerLastName)

VALUES (old.FirstName, old.LastName);

END

$$

DELIMITER ;

**4/ This trigger will insert a message to messages table to let user know that a customer was deleted. It also show time of deletion and remind user that they can go back to customer\_backup to look for that customer in case they needed based on the time of deletion.**

DROP TRIGGER IF EXISTS `theater`.`send\_notification\_after\_delete\_customer`;

DELIMITER $$

USE `theater`$$

CREATE TRIGGER `theater`.`send\_notification\_after\_delete\_customer`

AFTER DELETE ON `Customer` FOR EACH ROW

BEGIN

INSERT INTO messages(message) values ('Customer deleted. There is a record inserted into customer\_backup table.');

END

$$

DELIMITER ;

**5/ This trigger will check whether customer payment method is either Cash or Credit Card before update. If not it will display error message.**

DROP TRIGGER IF EXISTS `theater`.`Customer\_BEFORE\_UPDATE`;

DELIMITER $$

USE `theater`$$

CREATE TRIGGER `theater`.`Customer\_BEFORE\_UPDATE` BEFORE UPDATE ON `Customer` FOR EACH ROW

BEGIN

DECLARE errorMessage VARCHAR(200);

SET errorMessage = CONCAT('Payment method have to be either Cash or Credit Card');

IF new.PaymentMethod != 'Credit Card' and new.PaymentMethod != 'Cash' THEN

SIGNAL SQLSTATE'45000'

SET MESSAGE\_TEXT = errorMessage;

END IF;

END$$

DELIMITER ;

**6/ This trigger will send a message to messages table included all the old information, date, time of the customer after delete so user can go back to look for what were the previous information.**

DROP TRIGGER IF EXISTS `theater`.`send\_message\_after\_update`;

DELIMITER $$

USE `theater`$$

CREATE TRIGGER `theater`.`send\_message\_after\_update`

AFTER UPDATE ON `Customer` FOR EACH ROW

BEGIN

INSERT INTO messages (message) VALUES(concat('The following customer information updated in the database customer first name: from ',

'(',old.FirstName,')', ' to ', '(',new.FirstName,')',

', customer last name: from ', '(', old.LastName,')', ' to ', '(', new.LastName, ')',

', Payment method: from ','(',old.PaymentMethod,')',' to ','(', new.PaymentMethod, ')'));

END

$$

DELIMITER ;

**7/ This trigger will update customer tickets purchased in the customer promotion table. Any customer purchases 10 or more ticket will receive promotion.**

DROP TRIGGER IF EXISTS `theater`.`Customer\_Promotion\_Update`;

CREATE TRIGGER `theater`.`Customer\_Promotion\_Update`

AFTER INSERT ON `TICKET` FOR EACH ROW

BEGIN

IF NEW.CUSTOMER\_CUSTOMERID IN (SELECT CUSTOMER\_ID FROM CUSTOMER\_PROMOTION)

THEN

SET SQL\_SAFE\_UPDATES = 0;

UPDATE CUSTOMER\_PROMOTION

SET CURRENT\_NUMBER\_OF\_TICKET = CURRENT\_NUMBER\_OF\_TICKET + 1

WHERE NEW.CUSTOMER\_CUSTOMERID = CUSTOMER\_PROMOTION.CUSTOMER\_ID;

SET SQL\_SAFE\_UPDATES = 1;

ELSEIF NEW.CUSTOMER\_CUSTOMERID NOT IN (SELECT CUSTOMER\_ID FROM CUSTOMER\_PROMOTION)

THEN

INSERT INTO CUSTOMER\_PROMOTION(CUSTOMER\_ID, CURRENT\_NUMBER\_OF\_TICKET)

VALUES (NEW.CUSTOMER\_CUSTOMERID, 1);

END IF;

END

**8/ This trigger will subtract 1 from current\_number\_of\_ticket from Customer\_promotion table after a ticket deleted from ticket table.**

DROP TRIGGER IF EXISTS `theater`.`update\_promotion\_after\_delete\_ticket`;

DELIMITER $$

USE `theater`$$

CREATE TRIGGER `theater`.`update\_promotion\_after\_delete\_ticket`

AFTER DELETE ON `Ticket` FOR EACH ROW

BEGIN

IF OLD.CUSTOMER\_CUSTOMERID IN (SELECT CUSTOMER\_ID FROM CUSTOMER\_PROMOTION)

THEN

SET SQL\_SAFE\_UPDATES = 0;

UPDATE CUSTOMER\_PROMOTION

SET CURRENT\_NUMBER\_OF\_TICKET = CURRENT\_NUMBER\_OF\_TICKET - 1

WHERE OLD.CUSTOMER\_CUSTOMERID = CUSTOMER\_PROMOTION.CUSTOMER\_ID;

SET SQL\_SAFE\_UPDATES = 1;

END IF;

END$$

DELIMITER ;