IS 456 IT Database Systems Management

HOP03A Working with Triggers

4/13/2021 Developed by Farzin Bahadori

5/13/2021 Developed by Smita Dutta

School of Technology & Computing @ City University of Seattle (CityU)

Before You Start

- The directory path shown in screenshots may be different from yours.
- Some steps are not explained in the tutorial**.** If you are not sure what to do:
    1. Consult the resources listed below.
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes

Students will be able to:

Students will be able to:

- Understand the SQLite queries.
- Run queries in SQLite.
- Work with triggers

```
-- 01 update triggers

-- test.db


CREATE TABLE widgetCustomer ( id INTEGER PRIMARY KEY, name TEXT, last_order_id
INT );

CREATE TABLE widgetSale ( id INTEGER PRIMARY KEY, item_id INT, customer_id INT,
quan INT, price INT );


INSERT INTO widgetCustomer (name) VALUES ('Bob');

INSERT INTO widgetCustomer (name) VALUES ('Sally');

INSERT INTO widgetCustomer (name) VALUES ('Fred');


SELECT * FROM widgetCustomer;


CREATE TRIGGER newWidgetSale AFTER INSERT ON widgetSale

   BEGIN

      UPDATE widgetCustomer SET last_order_id = NEW.id WHERE widgetCustomer.id
= NEW.customer_id;

   END

;


INSERT INTO widgetSale (item_id, customer_id, quan, price) VALUES (1, 3, 5, 1995);

INSERT INTO widgetSale (item_id, customer_id, quan, price) VALUES (2, 2, 3, 1495);

INSERT INTO widgetSale (item_id, customer_id, quan, price) VALUES (3, 1, 1, 2995);

SELECT * FROM widgetSale;

SELECT * FROM widgetCustomer;


-- 02 preventing updates
```

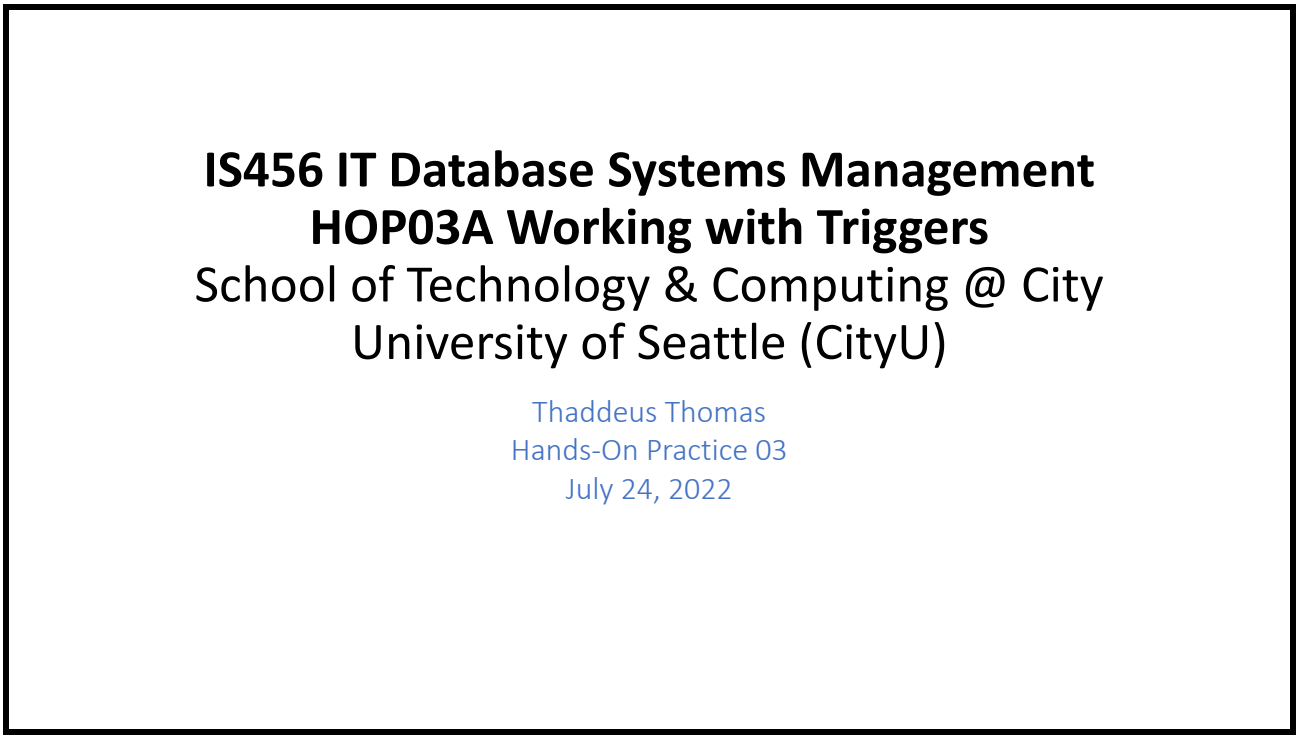
```
-- test.db


DROP TABLE IF EXISTS widgetSale;


CREATE TABLE widgetSale ( id integer primary key, item_id INT, customer_id INTEGER, quan INT, price INT,
   reconciled INT );
INSERT INTO widgetSale (item_id, customer_id, quan, price, reconciled) VALUES (1, 3, 5, 1995, 0);
INSERT INTO widgetSale (item_id, customer_id, quan, price, reconciled) VALUES (2, 2, 3, 1495, 1);
INSERT INTO widgetSale (item_id, customer_id, quan, price, reconciled) VALUES (3, 1, 1, 2995, 0);
SELECT * FROM widgetSale;


CREATE TRIGGER updateWidgetSale BEFORE UPDATE ON widgetSale
   BEGIN
      SELECT RAISE(ROLLBACK, 'cannot update table "widgetSale"') FROM widgetSale
         WHERE id = NEW.id AND reconciled = 1;
   END
;


BEGIN TRANSACTION;
UPDATE widgetSale SET quan = 9 WHERE id = 2;
END TRANSACTION;


SELECT * FROM widgetSale;


-- 03 timestamps
-- test.db
```

```
DROP TABLE IF EXISTS widgetSale;

DROP TABLE IF EXISTS widgetCustomer;


CREATE TABLE widgetCustomer ( id integer primary key, name TEXT, last_order_id
INT, stamp TEXT );

CREATE TABLE widgetSale ( id integer primary key, item_id INT, customer_id INTEGER,
quan INT, price INT, stamp TEXT );

CREATE TABLE widgetLog ( id integer primary key, stamp TEXT, event TEXT, username
TEXT, tablename TEXT, table_id INT);


INSERT INTO widgetCustomer (name) VALUES ('Bob');

INSERT INTO widgetCustomer (name) VALUES ('Sally');

INSERT INTO widgetCustomer (name) VALUES ('Fred');

SELECT * FROM widgetCustomer;


CREATE TRIGGER stampSale AFTER INSERT ON widgetSale
   BEGIN
      UPDATE widgetSale SET stamp = DATETIME('now') WHERE id = NEW.id;
      UPDATE widgetCustomer SET last_order_id = NEW.id, stamp = DATETIME('now')
         WHERE widgetCustomer.id = NEW.customer_id;
      INSERT INTO widgetLog (stamp, event, username, tablename, table_id)
         VALUES (DATETIME('now'), 'INSERT', 'TRIGGER', 'widgetSale', NEW.id);
   END
;


INSERT INTO widgetSale (item_id, customer_id, quan, price) VALUES (1, 3, 5, 1995);

INSERT INTO widgetSale (item_id, customer_id, quan, price) VALUES (2, 2, 3, 1495);

INSERT INTO widgetSale (item_id, customer_id, quan, price) VALUES (3, 1, 1, 2995);
```

```
SELECT * FROM widgetSale;

SELECT * FROM widgetCustomer;

SELECT * FROM widgetLog;


-- restore database

DROP TRIGGER IF EXISTS newWidgetSale;

DROP TRIGGER IF EXISTS updateWidgetSale;

DROP TRIGGER IF EXISTS stampSale;


DROP TABLE IF EXISTS widgetCustomer;

DROP TABLE IF EXISTS widgetSale;

DROP TABLE IF EXISTS widgetLog;
```

Screenshots

Provide at least 3 screenshots as part of HOP submission.



Summary

Write a 150-word summary to explain your understandings and findings from this lab assignment.

The slides were combined for both Hands on Practices

I learned some faster techniques for querying the data by doing this hands-on practice. I know if I go from output to output it will allow me to group 10 statements together until I get some feedback from the server. The Triggers were kind of tricky and were probably the only part of this assignment that required following steps exactly otherwise theyd just throw errors. Luckily when a drop and create supersede any SQL statement it prevents these types of errors. The feasibility of this on a live server seem impossible as it would cause far too many errors to be occurring.

IS 456 IT Database Systems Management

HOP03B Working with Joins

4/13/2021 Developed by Farzin Bahadori

5/13/2021 Developed by Smita Dutta

School of Technology & Computing @ City University of Seattle (CityU)

Before You Start

- The directory path shown in screenshots may be different from yours.
- Some steps are not explained in the tutorial**.** If you are not sure what to do:
    1. Consult the resources listed below.
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes

Students will be able to:

Students will be able to:

- Understand the SQLite queries.
- Run queries in SQLite.
- Understand join queries

```
Execute the following query
```

-- 02 JOIN -- test.db

```
-- join example tables, left and right

CREATE TABLE left ( id INTEGER, description TEXT );
CREATE TABLE right ( id INTEGER, description TEXT );

INSERT INTO left VALUES ( 1, 'left 01' );
INSERT INTO left VALUES ( 2, 'left 02' );
INSERT INTO left VALUES ( 3, 'left 03' );
INSERT INTO left VALUES ( 4, 'left 04' );
INSERT INTO left VALUES ( 5, 'left 05' );
INSERT INTO left VALUES ( 6, 'left 06' );
INSERT INTO left VALUES ( 7, 'left 07' );
INSERT INTO left VALUES ( 8, 'left 08' );
INSERT INTO left VALUES ( 9, 'left 09' );

INSERT INTO right VALUES ( 6, 'right 06' );
INSERT INTO right VALUES ( 7, 'right 07' );
INSERT INTO right VALUES ( 8, 'right 08' );
INSERT INTO right VALUES ( 9, 'right 09' );
INSERT INTO right VALUES ( 10, 'right 10' );
INSERT INTO right VALUES ( 11, 'right 11' );
INSERT INTO right VALUES ( 11, 'right 12' );
INSERT INTO right VALUES ( 11, 'right 13' );
INSERT INTO right VALUES ( 11, 'right 14' );

SELECT * FROM left;
SELECT * FROM right;

SELECT l.description AS left, r.description AS right
  FROM left AS l
  JOIN right AS r ON l.id = r.id;

-- restore database
DROP TABLE left;
DROP TABLE right;

-- sale example
SELECT * FROM sale;
SELECT * FROM item;

SELECT s.id AS sale, i.name, s.price
  FROM sale AS s
  JOIN item AS i ON s.item_id = i.id;

SELECT s.id AS sale, s.date, i.name, i.description, s.price
  FROM sale AS s
  JOIN item AS i ON s.item_id = i.id;
```

-- 03 Junction Table -- test.db

```
SELECT * FROM customer;
SELECT * FROM item;
SELECT * FROM sale;

SELECT c.name AS Cust, c.zip, i.name AS Item, i.description, s.quantity AS Quan, s.price
AS Price
  FROM sale AS s
  JOIN item AS i ON s.item_id = i.id
  JOIN customer AS c ON s.customer_id = c.id
  ORDER BY Cust, Item;

-- a customer without sales
INSERT INTO customer ( name ) VALUES ( 'Jane Smith' );
SELECT * FROM customer;

-- left joins
SELECT c.name AS Cust, c.zip, i.name AS Item, i.description, s.quantity AS Quan, s.price
AS Price
  FROM customer AS c
  LEFT JOIN sale AS s ON s.customer_id = c.id
  LEFT JOIN item AS i ON s.item_id = i.id
  ORDER BY Cust, Item;

-- restore database
DELETE FROM customer WHERE id = 4;
```

Summary

Write a 150-word summary to explain your understandings and findings from this lab assignment.

I didn't get anything out of this lesson really. I do like having sample code though to use as a template if I need to make a join table. Those are very convenient for interrelated data like the one we have for our team project. I created the views to behave very similar to the Junction table. That way you could have an HR schema with a certain view of employee records across tables and nearly the same view with constraints to fulfill a more secure function.