COMP23111
2016-2017
EX04
Alex-Radu Malan
9770386

```
SET linesize 250
SET pagesize 250

start create-Eclectic-Ecommerce-tables.sql
start populate-Eclectic-Ecommerce-tables.sql

-- Create a view showing the first and last names of customers
-- with shopping carts, then write a query that return its full
-- extent.

CREATE OR REPLACE VIEW shoppingCart_customers
AS SELECT DISTINCT firstName, lastName
FROM customerInfo
  RIGHT JOIN orderCartInfo
    ON customerInfo.loginName = orderCartInfo.customerID
WHERE otype = 'S';

SELECT * FROM shoppingCart_customers;

-- Create a view showing the code, item number, category id and
-- quantity in stock of inventory items that need to be reordered
-- (where an inventory item needs to be reordered if the quantity
-- in stock drops below 25), then write a
-- query that return its full extent.

CREATE OR REPLACE VIEW itemsToBeReordered
AS SELECT inventoryItem.code, inventoryItem.itemNum,
          inventoryItem.qtyInStock, category.categoryID
FROM inventoryItem
    JOIN itemType
      ON inventoryItem.itemNum = itemType.itemNum
    JOIN category
      ON itemType.belongsTo = category.categoryID
WHERE inventoryItem.qtyInStock < 25;

SELECT * FROM itemsToBeReordered;


-- Create a view showing the login name, first and last names,
-- order cart id and total price of each order, then write a query
-- that return its full extent
```

```
CREATE OR REPLACE VIEW infoOrderTotal
AS SELECT DISTINCT customerInfo.loginname, customerInfo.firstname,
                   customerInfo.lastName,
                    orderCartInfo.orderCartId,
                 Sum(lineItems.orderPrice * lineItems.qtyordered)
                  as theTotalPrice
FROM customerInfo
    JOIN orderCartInfo
      ON customerInfo.loginname = orderCartInfo.customerID
    JOIN lineItems
      ON orderCartInfo.orderCartId = lineItems.orderCartId
GROUP BY (customerInfo.loginname, customerInfo.firstname,
          customerInfo.lastName, orderCartInfo.orderCartId);

SELECT * FROM infoOrderTotal;

-- Create a view showing the login name, first and last names, and
-- total of all orders by a customer, then write a query that
-- return its full extent.

CREATE OR REPLACE VIEW OrderPerPerson
AS SELECT DISTINCT customerInfo.loginname,
                   customerInfo.firstname,
                      customerInfo.lastName,
                         Sum(lineItems.orderPrice *
lineItems.qtyordered) as theTotalPricePerPerson
FROM customerInfo
    JOIN orderCartInfo
      ON customerInfo.loginname = orderCartInfo.customerID
    JOIN lineItems
      ON orderCartInfo.orderCartId = lineItems.orderCartId
GROUP BY (customerInfo.loginname, customerInfo.firstname,
customerInfo.lastName);

SELECT * FROM OrderPerPerson;

-- Create a view to return the number of carts per customer, then
-- use this view in a query with a CASE statement in the SELECT
-- clause that, for each customer, returns the login name and an
-- outcome (represented as a string), which is either 'BR-1
-- satisfied' if that customer has no more than two carts in the
-- database, or 'BR-1 violated' otherwise.
```

```
CREATE OR REPLACE VIEW cartPerPerson
AS SELECT DISTINCT customerInfo.loginname,
                   customerInfo.firstname,
                      customerInfo.lastName,
                      Count(orderCartId) as numberOfCarts
FROM customerInfo
    JOIN orderCartInfo
      ON customerInfo.loginname = orderCartInfo.customerID
GROUP BY (customerInfo.loginname, customerInfo.firstname,
customerInfo.lastName);

select cartPerPerson.loginname,
CASE
    when numberOfCarts <= 2 THEN 'BR-1 Satisfied'
    else 'BR-1 Violated'
end
FROM cartPerPerson;


-- Now, rather than define a view, use query nesting in a similar
-- problem, as follows. Firstly, define a query (we'll refer to it
-- as 2) that returns item num, item size, item colour and acount
-- of how many items of a given color and size there are.
-- Secondly, define a query (we'll refer to it as 1) that nests 2
-- in its FROM clause and (somewhat similarly to the previous
-- task) has a CASE statement in the SELECT clause that returns as
-- outcome the string 'BR-2 satisfied' if the item number does not
-- occur more than once with the same color and size, or 'BR-2
-- violated' otherwise. Finally, define a (top-level, as it were)
-- query that uses 1 and when executed returns only the item
--number, color and size that violate the BR-2 business rule.

SELECT itemNum, itemSize, itemColor
FROM
(
    SELECT itemNum, itemSize, itemColor,
      CASE WHEN countSize = 1 AND countColor = 1
           THEN 'BR-2 Satisfied'
      ELSE 'BR-2 Violated'
    END AS result
    FROM
    (
        SELECT DISTINCT inventoryItem.itemNum,
                 inventoryItem.itemColor, inventoryItem.itemSize,
                 count(inventoryItem.itemColor) as countColor
```

```
                    count(inventoryItem.itemSize) as countSize
        FROM  inventoryItem, itemType
        WHERE itemType.itemNum = inventoryItem.itemNum
        GROUP BY (inventoryItem.itemNum, inventoryItem.itemColor,
                    inventoryItem.itemSize)
    )
)
WHERE result = 'BR-2 Violated';

SELECT * FROM itemType;


-- Code a SQL trigger that raises an error if the price of an item
-- is set to a value that is more than four times the value of the
-- least expensive item.

CREATE OR REPLACE TRIGGER ErrorPrice BEFORE
UPDATE OR INSERT OF price
ON itemType FOR EACH ROW DECLARE maxPrice NUMBER;

BEGIN
    SELECT min(price) * 4
      INTO maxPrice
    FROM itemType;

    IF :NEW.price > maxPrice THEN appError(-2000, ('The price is
bigger than the min by four times'), TRUE);
    END IF;
END;

INSERT INTO itemType VALUES('C8', 'Book', '****', 23.4, 'H');
INSERT INTO itemType VALUES('C8', 'Book', '****', 11.4, 'H');

UPDATE itemType
SET itemNum = 'C8', name = 'Book3', picture = '****', price =
23.4, belongsTo = 'H'
WHERE itemNum = 'A0';

UPDATE itemType
SET itemNum = 'C8', name = 'Book3', picture = '****', price =
11.4, belongsTo = 'H'
WHERE itemNum = 'A0';

-- Drop the table and the data from it
start drop-Eclectic-Ecommerce-tables.sql
```