```sql

-- create the database
DROP DATABASE IF EXISTS JungleBooks;
CREATE DATABASE JungleBooks;

USE JungleBooks;


-- create the tables
DROP TABLE IF EXISTS Customer;
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(32) NOT NULL,
    LastName VARCHAR(32) NOT NULL,
    BillingAddress VARCHAR(255) NOT NULL,
    ShippingAddress VARCHAR(255) NOT NULL,
    PhoneNumber CHAR(10) NOT NULL,
    EmailAddress VARCHAR(32) NOT NULL,
    Username VARCHAR(32) NOT NULL,
    Password VARCHAR(32) NOT NULL
);

DROP TABLE IF EXISTS CustomerOrder;
CREATE TABLE CustomerOrder(
    CustomerOrderID INT PRIMARY KEY,
    DateOrdered DATETIME NOT NULL,
    DateShipped DATETIME NULL,
    Status VARCHAR(32) NOT NULL,
    Subtotal FLOAT NOT NULL,
    Taxes FLOAT NOT NULL,
    TotalCost FLOAT NOT NULL,
    CustomerID INT NOT NULL
);

DROP TABLE IF EXISTS Book;
CREATE TABLE Book(
    BookID INT PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    Summary VARCHAR(255) NOT NULL,
    ISBN VARCHAR(32) NOT NULL,
    Price FLOAT NOT NULL
);

DROP TABLE IF EXISTS Author;
CREATE TABLE Author(
    AuthorID INT PRIMARY KEY,
    FirstName VARCHAR(32) NOT NULL,
    LastName VARCHAR(32) NOT NULL
);

DROP TABLE IF EXISTS Category;
CREATE TABLE Category(
    CategoryID INT PRIMARY KEY,
    Description VARCHAR(32) NOT NULL
);

DROP TABLE IF EXISTS Book_CustomerOrder;
CREATE TABLE Book_CustomerOrder(
    BookID INT NOT NULL,
    CustomerOrderID INT NOT NULL,
    Quantity INT NOT NULL,
    Subtotal FLOAT NOT NULL
);

DROP TABLE IF EXISTS Book_Author;
CREATE TABLE Book_Author(
    BookID INT NOT NULL,
    AuthorID INT NOT NULL
);

DROP TABLE IF EXISTS Book_Category;
CREATE TABLE Book_Category(
    BookID INT NOT NULL,
    CategoryID INT NOT NULL
);


-- create relationships
ALTER TABLE CustomerOrder
    DROP CONSTRAINT IF EXISTS FK__Customer__CustomerID;
ALTER TABLE CustomerOrder
    ADD CONSTRAINT FK__Customer__CustomerID FOREIGN KEY (
CustomerID)
        REFERENCES Customer (CustomerID);


ALTER TABLE Book_CustomerOrder
    DROP CONSTRAINT IF EXISTS PK_Book_CustomerOrder;
ALTER TABLE Book_CustomerOrder
    ADD CONSTRAINT  PK_Book_CustomerOrder
    PRIMARY KEY (BookID, CustomerOrderID);

```

```
ALTER TABLE Book_CustomerOrder
    DROP CONSTRAINT IF EXISTS FK_Book1_BookID;
ALTER TABLE Book_CustomerOrder
    ADD CONSTRAINT  FK_Book1_BookID
    FOREIGN KEY (BookID)
    REFERENCES Book (BookID);

ALTER TABLE Book_CustomerOrder
    DROP CONSTRAINT IF EXISTS
FK_CustomerOrder_CustomerOrderID;
ALTER TABLE Book_CustomerOrder
    ADD CONSTRAINT  FK_CustomerOrder_CustomerOrderID
    FOREIGN KEY (CustomerOrderID)
    REFERENCES CustomerOrder (CustomerOrderID);


ALTER TABLE Book_Author
    DROP CONSTRAINT IF EXISTS PK_Book_Author;
ALTER TABLE Book_Author
    ADD CONSTRAINT  PK_Book_Author
    PRIMARY KEY (BookID, AuthorID);

ALTER TABLE Book_Author
    DROP CONSTRAINT IF EXISTS FK_Book2_BookID;
ALTER TABLE Book_Author
    ADD CONSTRAINT  FK_Book2_BookID
    FOREIGN KEY (BookID)
    REFERENCES Book (BookID);

ALTER TABLE Book_Author
    DROP CONSTRAINT IF EXISTS FK_Author_AuthorID;
ALTER TABLE Book_Author
    ADD CONSTRAINT  FK_Author_AuthorID
    FOREIGN KEY (AuthorID)
    REFERENCES Author (AuthorID);


ALTER TABLE Book_Category
    DROP CONSTRAINT IF EXISTS PK_Book_Category;
ALTER TABLE Book_Category
    ADD CONSTRAINT  PK_Book_Category
    PRIMARY KEY (BookID, CategoryID);

ALTER TABLE Book_Category
    DROP CONSTRAINT IF EXISTS FK_Book3_BookID;
ALTER TABLE Book_Category
```

```sql
    ADD CONSTRAINT  FK_Book3_BookID
    FOREIGN KEY (BookID)
    REFERENCES Book (BookID);

ALTER TABLE Book_Category
    DROP CONSTRAINT IF EXISTS FK_Category_CategoryID;
ALTER TABLE Book_Category
    ADD CONSTRAINT  FK_Category_CategoryID
    FOREIGN KEY (CategoryID)
    REFERENCES Category (CategoryID);


-- INSERTing categories
INSERT INTO Category
    (CategoryID, Description)
VALUES
    (1, 'Classic Regency novel'), (2, 'Fantasy'), (3, 'Adventure'), (4, 'Science fiction'), (5, 'Comedy'), (6, 'Database');


-- INSERTing authors
INSERT INTO Author
    (AuthorID, FirstName , LastName)
VALUES
    (1, 'Jane', 'Austen'), (2, 'J. R. R.', 'Tolkien'), (3, 'Douglas', 'Adams');


-- INSERTing books
INSERT INTO Book
    (BookID, Title, Summary, ISBN, Price)
VALUES
    (1, 'Pride and Prejudice and Database', 'Pride and Prejudice has delighted generations of readers with its unforgettable cast of characters, carefully choreographed plot, and a hugely entertaining view of the world and its absurdities.', '9780199535569', 13.5),
    (2, 'Lord of the Rings: The Two Towers', 'Dispatched, from the UK, within 48 hours of ordering. This book is in good condition but will show signs of previous ownership. Please expect some creasing to the spine and/or minor damage to the cover.', '9780048231567', 9.9),
    (3, 'The Hitchhikers Guide to the Galaxy', 'Campus mission is to set online bookstore industry standards for savings, selection, convenience and customer service as
expressed in the companys slogan Textbooks Easy. Fast.
Cheap!', '9780345391803', 8.5);


-- INSERTing book categories
INSERT INTO Book_Category
    (CategoryID, BookID)
VALUES
    (1, 1), (2, 2), (3, 2), (4, 3), (5, 3), (6,3);


-- INSERTing book authors
INSERT INTO Book_Author
    (BookID, AuthorID)
VALUES
    (1, 1), (2, 2), (3, 3);


-- INSERTing customers
INSERT INTO Customer
    (CustomerID, FirstName, LastName, BillingAddress,
ShippingAddress, PhoneNumber, EmailAddress, Username,
Password)
VALUES
    (1, 'Reginaldo', 'Oliveira', 'Main Street 13 B3LN5C',
'Main Street 13 B3LN5C', '9021012002', 'ReginaldoOliveira@
gmail.com', 'Reginaldo121', 'R3g1nald0'),
    (2, 'Anderson', 'Medina', 'South Street 521 E2LG3D', '
South Street 521 E2LG3D', '9026035555', 'AndersonMedina@
gmail.com', 'AndersonMDN', '4nd3rs0n'),
    (3, 'Luiz', 'Souza', 'Young Street 1452 A3LN1C', '
Young Street 1452 A3LN1C', '9021100555', 'LuizSouza@gmail.
com', 'Luiz1987', 'Lu1z@2304'),
    (4, 'Maria', 'Silva', 'North Street 322 N3LW5C', '
North Street 322 N3LW5C', '9025023333', 'MariaSilva@gmail.
com', 'Maria_864', 'M4r14!1221');


-- INSERTing customer`s order
INSERT INTO CustomerOrder
    (CustomerOrderID, DateOrdered, DateShipped, Status,
Subtotal, Taxes, TotalCost, CustomerID)
VALUES
    (1, '2019-10-31T18:00:000', '2019-11-11T18:00:000', '
Completed', 31.9, 1.15, 36.685, 1),
    (2, '2019-11-20T22:00:000', '2019-11-26T18:00:000', '
Completed', 13.5, 1.15, 15.525, 2);

INSERT INTO CustomerOrder
    (CustomerOrderID, DateOrdered, Status, Subtotal, Taxes
, TotalCost, CustomerID)
VALUES
    (3, '2019-11-21T10:00:000', 'In progress', 172.3, 1.15
, 198.145, 3);


-- INSERTing books to a CustomerOrder
INSERT INTO Book_CustomerOrder
    (CustomerOrderID, BookID, Quantity, Subtotal)
VALUES
    (1, 1, 1, 13.50), (1, 2, 1, 9.90), (1, 3, 1, 8.50),
    (2, 1, 1, 13.50),
    (3, 1, 5, 67.5), (3, 2, 2, 19.8), (3, 3, 10, 85.0);


-- DELETEing the second order
DELETE FROM Book_CustomerOrder
WHERE CustomerOrderID = 2;

DELETE FROM CustomerOrder
WHERE CustomerOrderID = 2;


-- UPDATEing the status of the first order to complete
UPDATE CustomerOrder
SET Status = 'Complete'
WHERE CustomerOrderID = 1;


-- UPDATEing the third order to add another copy of the
second book
UPDATE Book_CustomerOrder BCO
RIGHT OUTER JOIN Book B
ON BCO.BookID = B.BookID
SET BCO.Quantity = (BCO.Quantity + 1), BCO.Subtotal = (BCO
.Subtotal + B.Price)
WHERE BCO.CustomerOrderID = 3 AND BCO.BookID = 2;

UPDATE CustomerOrder CO
INNER JOIN
    (
        SELECT BCO.CustomerOrderID, SUM(Subtotal) AS
```

```
SUBTOTAL_SUM
        FROM Book_CustomerOrder BCO
        WHERE BCO.CustomerOrderID = 3
        GROUP BY BCO.CustomerOrderID
    ) BCO
    ON BCO.CustomerOrderID = CO.CustomerOrderID
SET CO.Subtotal = BCO.SUBTOTAL_SUM, CO.TotalCost = (CO.
Subtotal * CO.Taxes)
WHERE CO.CustomerOrderID = 3;


-- SELECTing all customer information for customers that
have no orders. (i.e. the count of their order ids is zero
)
SELECT C.*, COUNT(CO.CustomerID) AS CUSTOMER_COUNT
FROM Customer C
LEFT OUTER JOIN CustomerOrder CO
ON C.CustomerID = CO.CustomerID
GROUP BY CustomerID
HAVING CUSTOMER_COUNT = 0;


-- SELECTing the title, author, ISBN and price of all
books related to databases. (i.e. contain the word “
database” or are in the “database” category)
SELECT B.Title, A.FirstName, A.LastName, B.ISBN, B.Price
FROM Book B
LEFT OUTER JOIN Book_Author BA
ON B.BookID = BA.BookID
LEFT OUTER JOIN Author A
ON BA.AuthorID = A.AuthorID
RIGHT OUTER JOIN Book_Category BC
ON B.BookID = BC.BookID
RIGHT OUTER JOIN Category C
ON BC.CategoryID = C.CategoryID
WHERE B.Title LIKE '%database%' OR C.Description = '
database'
GROUP BY B.Title;


-- SELECTing the email addresses of customers who have
outstanding orders. (i.e. orders that have no ship date
yet)
SELECT C.EmailAddress, COUNT(CO.DateShipped) AS
DATESHIPPED_COUNT, COUNT(CO.CustomerID) AS CUSTOMER_COUNT
FROM Customer C
```

```sql
LEFT OUTER JOIN CustomerOrder CO
ON C.CustomerID = CO.CustomerID
GROUP BY C.EmailAddress
HAVING DATESHIPPED_COUNT = 0 AND CUSTOMER_COUNT > 0;


-- SELECTing just the order information (i.e. not the order details) on all orders that have purchased more than one copy of any of the books. (e.g. two copies of book one, three copies of book three, etc)
SELECT CO.CustomerOrderID, SUM(BCO.Quantity) AS QUANTITY_SUM, COUNT(BCO.Quantity) AS QUANTITY_COUNT
FROM CustomerOrder CO
LEFT OUTER JOIN Book_CustomerOrder BCO
ON CO.CustomerOrderID = BCO.CustomerOrderID
GROUP BY CO.CustomerOrderID
HAVING QUANTITY_SUM > QUANTITY_COUNT;
-- I'm sorry, but I didn't understand exactly what should be shown when you say "Display just the order information (ie not the order details)". I understood that it was to show only the order number


-- SELECTing the order number and the total number of books in each order.
SELECT CO.CustomerOrderID, SUM(BCO.Quantity) AS QUANTITY_SUM
FROM CustomerOrder CO
LEFT OUTER JOIN Book_CustomerOrder BCO
ON CO.CustomerOrderID = BCO.CustomerOrderID
GROUP BY CO.CustomerOrderID;


-- SELECTing each order number, customer name and the total cost of their order by adding the cost of all the items. (Determine each cost by using the quantity and price and adding 15% tax).
SELECT CO.CustomerOrderID, C.FirstName, C.LastName, SUM(B.Price * BCO.Quantity * CO.Taxes) AS TotalCost_SUM
FROM CustomerOrder CO
LEFT OUTER JOIN Customer C
ON CO.CustomerID = C.CustomerID
RIGHT OUTER JOIN Book_CustomerOrder BCO
ON CO.CustomerOrderID = BCO.CustomerOrderID
RIGHT OUTER JOIN Book B
ON BCO.BookID = B.BookID
```

```sql
GROUP BY CO.CustomerOrderID;


```