# A simple database supporting an online book seller

# Tables about Books and Authors

CREATE TABLE Book (

 Isbn INTEGER,

 Title CHAR[120] NOT NULL,

 Synopsis CHAR[500],

 ListPrice CURRENCY NOT NULL,

 AmazonPrice CURRENCY NOT NULL,

 SavingsInPrice CURRENCY NOT NULL, /\* redundant

 AveShipLag INTEGER,

 AveCustRating REAL,

 SalesRank INTEGER,

 CoverArt FILE,

 Format CHAR[4] NOT NULL,

 CopiesInStock INTEGER,

 PublisherName CHAR[120] NOT NULL,

 PublicationDate DATE NOT NULL,

 PublisherComment CHAR[500],

 PublicationCommentDate DATE,

 PRIMARY KEY (Isbn)

)

CREATE TABLE Author (

 AuthorName CHAR[120],

 AuthorBirthDate DATE,

 AuthorAddress ADDRESS,

 AuthorBiography FILE,

 PRIMARY KEY (AuthorName, AuthorBirthDate)

)

CREATE TABLE WrittenBy (/\*Books are written by authors

 Isbn INTEGER,

 AuthorName CHAR[120],

 AuthorBirthDate DATE,

 OrderOfAuthorship INTEGER NOT NULL,

 AuthorComment FILE,

 AuthorCommentDate DATE,

 PRIMARY KEY (Isbn, AuthorName, AuthorBirthDate)

 )

CREATE TABLE Publisher (

 PublisherName CHAR[120],

 PublisherAddress ADDRESS,

 PRIMARY KEY (PublisherName)

)

**Tables about Customers and Customer Service**

CREATE TABLE Customers (/\* Customers identified by email address

 CustEmailAddr CHAR[120],

 CustName CHAR[120] NOT NULL,

 CustPassword CHAR[20] NOT NULL,

 PRIMARY KEY (CustEmailAddr)

)

CREATE TABLE AlertTo (/\* Customers can request notification about new books by an author

 CustEmailAddress CHAR[120],

 DateOfAlertRequest DATE NOT NULL,

 AuthorName CHAR[120],

 AuthorBirthDate DATE,

 PRIMARY KEY (UserEmailAddr, AuthorName, AuthorBirthDate)

)

CREATE TABLE Accounts (/**\*** Customers can have zero or more accounts

CustEmailAddr CHAR[120],

 CreditCardNumber INTEGER,

 ShippingAddr ADDRESS NOT NULL,

 DateOpened DATE NOT NULL,

 PRIMARY KEY (CustEmailAddr, CreditCardNumber)

)

**Tables about Purchases and Shipments**

CREATE TABLE Transactions (/\* Transactions (purchases) are made on a customer account

 TransNumber INTEGER,

 OrderDate DATE,

 PaymentClearanceDate DATE, /\* if NULL, then payment has not cleared \*/

 CustEmailAddr CHAR[120] NOT NULL,

 CreditCardNo INTEGER NOT NULL,

 PRIMARY KEY (TransNum),

 )

CREATE TABLE Shipment (/\* A record of purchases awaiting or when shipment

 ShipId INTEGER,

 ShipCost CURRENCY,

 ShipDate DATE, /\* if this is NULL, then not shipped yet \*/

 TransNumber INTEGER NOT NULL,

 PRIMARY KEY (ShipId)

)

CREATE TABLE Shipped (/\* A quantity of book associated with a shipment and therefore transaction

 Quantity INTEGER,

 ShipId INTEGER,

 Isbn INTEGER,

 PRIMARY KEY (ShipId, Isbn),

)

**Upload Answers to these to Oak Discussion Board**

***Study Group members in attendance*:**

***Special Note: You have not yet (officially) learned about foreign key constraints and other types of constraints that are critical in establishing unambiguous requisite relationships between tables, so for right now use same-named attributes (in different tables) as implicit associations that you can exploit in (natural) joins, for example.***

**Questions about the Database (Yes/No, short explanation)**

Does it appear that a Book be associated with more than one publisher?

Does it appear that a Book can be associated with more than one author?

Does it appear that a Book can be associated with zero (known) authors?

Can two of a customer’s accounts be associated with the same credit card ?

Does it appear that a shipment can be associated with more than one Transaction?

**Queries** *Each group write the query – suggestion is that each person spend a few minutes working on it, then discuss and reach consensus, or not (Groups can upload more than one answer per query)*

You will refer to the table definitions of the Book retailer DB for this assignment.

1. An online book retailer undoubtedly uses a more sophisticated resource planning policy than this, but certainly if the number of copies of a given book (Isbn) awaiting shipment exceeds the number of copies in stock, then ordering more books of the given Isbn is appropriate. Write a query that lists the Isbns of all books in which the number of copies awaiting shipment (Shipment.ShipDate IS NULL) to all customers exceeds the number of copies in stock (Books.CopiesInStock) . The query result should also list the difference between the total number of copies awaiting shipment and the number of copies in stock for each qualifying Isbn. Thus, your query should produce a table with two fields (Isbn, NumberofCopiesShort), and there will be only one entry per Isbn that is short.

2. Write a query that lists the CustEmailAddresses of all customers that have purchased books (Transactions.PaymentClearenceDate IS NOT NULL) that have not been shipped (Shipment.ShipDate IS NULL). The query should also list the Isbns and quantities of each unshipped book for each customer. Thus, your query should produce a table with three fields (CustEmailAddr, Isbn, #UnshippedBooks), where there will be only one entry for each (CustEmailAddr, Isbn) pair.

3. An online book retailer would like to know how effective its Alert service is at promoting sales. Each alert is entered on a particular date for a particular author. A customer that buys a book that was published on a date after the alert date by the given author, has arguably responded to the alert in a positive way as far as our retailer is concerned. Write a query that lists the email addresses of all customers (CustEmailAddr) that have purchased books (Books.Isbn, Transactions.PaymentClearenceDate IS NOT NULL) by authors (AuthorName, AuthorBirthDate) specified in the alert that were published after that customer (AlertTo.CustEmailAddr = Customers.CustEmailAddr) initiated a relevant alert request. The table produced will have two fields (CustEmailAddr, TotalNumberofPostAlertPurchases). The query result should order the customers (CustEmailAddresses) from those that have purchased the most total quantity of books (including copies of the same book) to those that have purchased the least quantity.

Since our tables assume a field type DATE, you can assume that if D1 and D2 are DATEs,

Then D1 > D2, D1 < D2, D1 = D2, etc are legal comparisons with the obvious meaning.