

Rutgers University, Business School/Undergraduate New Brunswick
Operations Management (33:623:370)
 Instructor: Jonathan Eckstein

Solutions to Practice Material from a Prior Second Midterm Exam

Q1. Access Query Construction

(a)

Field:	Title	Category	YearPublished	Language
Table:	BOOK	BOOK	BOOK	BOOK
Total:				
Sort:				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Criteria:			>= 2002	"Spanish"

Note that the text said “*YearPublished*, *BirthYear*, and *DeathYear* are integer fields that hold only a year, not a more detailed date.” That means that the correct format for the year criterion is just “>= 2002”, not “>= #1/1/2002#”. If *YearPublished* had the date/time datatype instead of the integer datatype, then it would be “>= #1/1/2002#”.

(b)

Field:	Title	Name	YearPublished	NumPages	AuthorPosition
Table:	BOOK	AUTHOR	BOOK	BOOK	WROTE
Total:					
Sort:				Descending	
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Criteria:				>= 1000	1

Note that to make sure that you only show the names of first authors, you need the criterion *AuthorPosition* = 1 on the WROTE table.

(c)

Field:	Title	BarCode	ISBN
Table:	BOOK	COPYOFBOOK	BOOK
Total:	Group By	Count	Group By
Sort:			
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Criteria:			

To make sure books with the same title but different ISBN’s are not lumped together when counting copies, you need to group by ISBN, but since you are not supposed to show that field, you should leave the “show” box unchecked. Note that any field of the COPY table would do just as well as *BarCode* in this query, since the “Count” function just counts rows.

(d) Here is a solution that uses an aggregation query:

Field:	Title	YearPublished	NumPages	Category	AuthorID	ISBN
Table:	BOOK	BOOK	BOOK	BOOK	WROTE	BOOK
Total:	Group by	Group by	Group by	Group by*	Count	Group By
Sort:						
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteria:				"Fiction"	> 1	

Note that here we are applying a criterion (more than one author) *after* grouping to count the number of authors, so we put a criterion in the same column as "Count". Note that you could also use "Where" instead of "Group by" in the *Category* column (hence the "*"). Technically, we should still group by but not show ISBN, in order to guard against the remote possibility of two different books have the same title, publication year, number of pages, and category.

There is also another solution, which comes from noting that if the WROTE table is set up as indicated, then any book with more than one author should have a second author, and therefore an entry in WROTE with *AuthorPosition* = 2. Thus we could also use the query

Field:	Title	YearPublished	NumPages	Category	AuthorPosition
Table:	BOOK	BOOK	BOOK	BOOK	WROTE
Total:					
Sort:					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteria:				"Fiction"	2

Note that a criterion of "> 1" instead of "2" in the *AuthorPosition* column would not be quite right, since you would then get multiple identical lines of output for books with more than two authors. For example, a book with three authors would appear twice in the output.

(e)

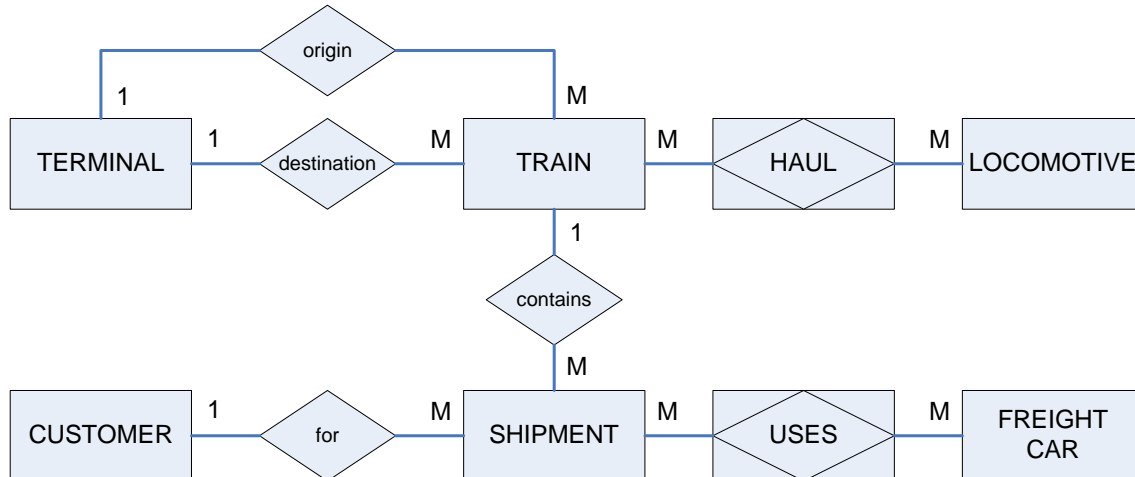
Field:	Name	Nationality	YearPublished	ISBN	NumPages	AuthorID	AuthorPosition
Table:	AUTHOR	AUTHOR	BOOK	BOOK	BOOK	AUTHOR	WROTE
Total:	Group by	Group by*	Where	Count	Avg	Group by	Group by*
Sort:							
Show:	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteria:		"USA"	< 1939				1
Or:		"USA"	> 1945				1

Note that we need a non-shown "Group by" for *AuthorID* to guard against the chance of authors with the same name and nationality being lumped together. The condition on *AuthorPosition* is needed to make sure we only look at first authors. Using "Where" is critical for *YearPublished* – otherwise, books with different publication years would not be grouped together. For *Nationality* and *AuthorPosition*, you could also use "Where" instead of "Group by".

Note that Access first combines the cells in each criterion line by "and", and then combines the lines by "or". Therefore, we have to repeat "USA" in the *Nationality* column and "1" in the

AuthorPosition column. I'm not sure whether it is legal syntax to put "< 1939 or > 1945" or "not between 1939 and 1945" into a single cell, so it is safer to express the "or" condition using two different lines.

Q2. Database Design: the Jersey Pacific Railroad



TERMINAL(FourLetterCode, Name, Description, Longitude, Latitude, TrackSpace)

TRAIN(TrainID, OriginTerminal, DestinationTerminal,
 DepartureDateTime, ArrivalDateTime)
 OriginTerminal foreign key to TERMINAL
 DestinationTerminal foreign key to TERMINAL

LOCOMOTIVE(PlacardNumber, Manufacturer, Model, Horsepower, Weight, YearBuilt)

HAUL(PlacardNumber, TrainID)
 PlacardNumber foreign key to LOCOMOTIVE
 TrainID foreign key to TRAIN

CUSTOMER(CustomerID, Name, Address, City, State, Zip, Phone)

SHIPMENT(ShipmentID, FreightDescription, Weight, Volume, Price,
 CustomerID, TrainID)
 CustomerID foreign key to CUSTOMER
 TrainID foreign key to TRAIN

FREIGHTCAR(RegistrationNumber, Type, EmptyWeight, CargoCapacity, Length,
 DateBuilt)

USES(ShipmentID, RegistrationNumber)
 ShipmentID foreign key to SHIPMENT
 RegistrationNumber foreign key to FREIGHTCAR

The dual one-to-many relationship between TRAIN and TERMINAL is very similar to the commuter airline problem we covered in class. Instead, you could also have a many-to-many setup like

TRAIN(TrainID, DepartureDateTime, ArrivalDateTime)

TRAINTERMINAL(TrainID, FourLetterCode, OriginOrDestination)

TrainID foreign key to TRAIN

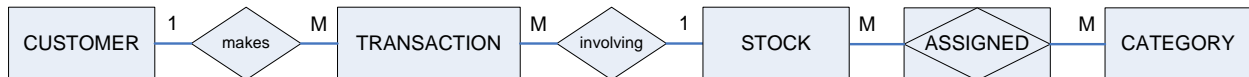
FourLetterCode foreign key to TERMINAL,

where *OriginOrDestination* contains (for example) “O” to indicate the origin terminal, and “D” to indicate the destination terminal.

Note that the TRAIN-LOCOMOTIVE relationship has to be many-to-many. The reason is that, although a locomotive only hauls one train at a time, the problem specifies that the system must track past history (see the first paragraph), and, as the last sentence points out, “once a locomotive has finished with one train, it of course becomes free to haul a different train”. The SHIPMENT-FREIGHTCAR relationship must also be many-to-many for very similar reasons.

There is no need for a direct relationship between TRAIN and FREIGHTCAR. Such a relationship is redundant because one can determine which freight cars are in a train by passing through the SHIPMENT table.

Q3. Normalization: the Piscataway Stock Exchange



CUSTOMER(CustID, CustName)

STOCK(TickerSymbol, StockName)

TRANSACTION(TransID, DateAndTime, TransType, CustID, TickerSymbol,
NumShares, UnitPrice)

CustID foreign key to CUSTOMER

TickerSymbol foreign key to STOCK

CATEGORY(CategCode, CategoryDescrip)

ASSIGNED(TickerSymbol, CategCode)

TickerSymbol foreign key to STOCK

CategCode foreign key to CATEGORY

It may be tempting to put the *UnitPrice* field to the STOCK table, but that will not work since the price can obviously vary over time. You could also try to have a PRICE table indexed by

stock and time, but that also would not work in this case because of the “spread” between buying and selling prices, which is visible in the data and is how stock exchanges typically support themselves. With the assumption that all “buys” at the same time have the same price, and all “sells” at the same time have the same price, one could have a PRICE table of the form PRICE(TickerSymbol, DateAndTime, TransType, Price). But one could also imagine this assumption being violated; so I just show the simpler solution of having *Price* in the TRANSACTION table.

Note that each transaction here involves only one stock, so there’s no need to use the ORDER-ORDERDETAIL-PRODUCT pattern that is common in typical retail databases.