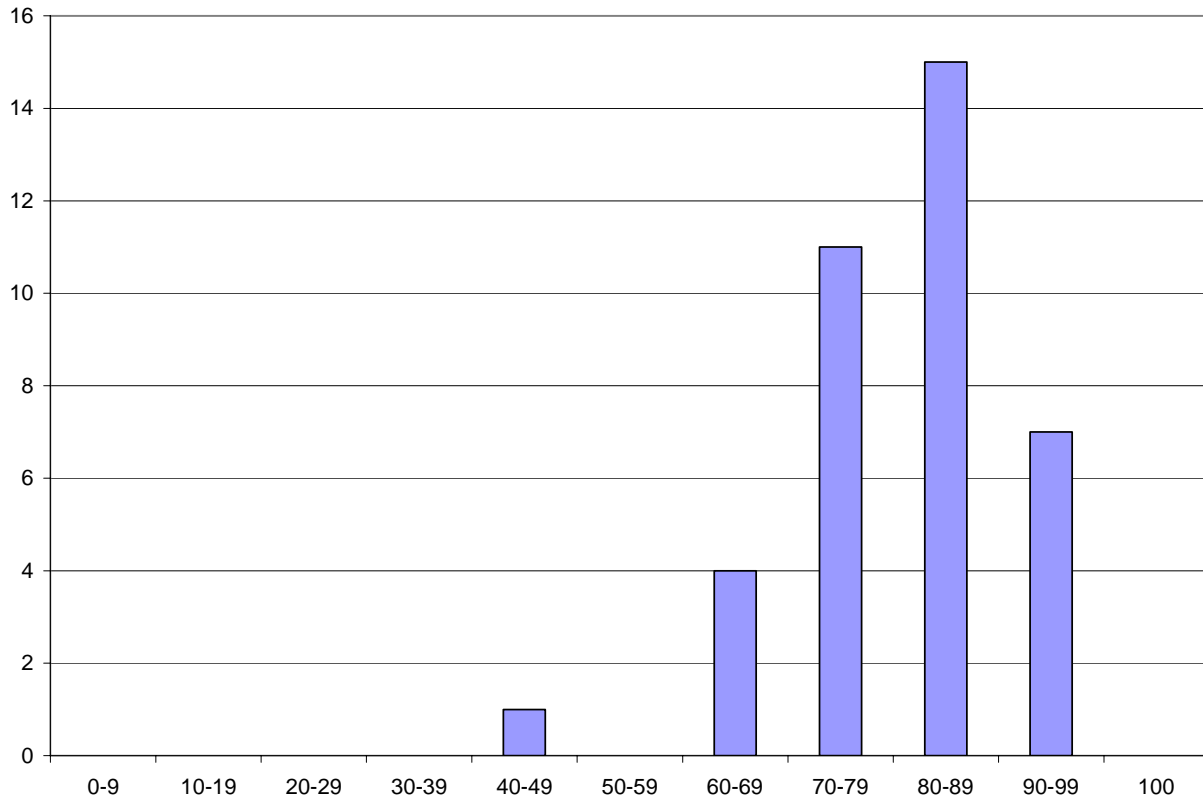


Rutgers University, Business School/Undergraduate New Brunswick
Operations Management (33:623:370:05)
 Fall 2008; Instructor: Jonathan Eckstein

Solutions to Midterm Exam — Friday, November 21, 2008

	Q1	Q2	Q3	Total
Points Allocated	30	40	30	100
Highest Score	29	40	30	97
Mean	25.6	32.9	21.6	80.2
Median	25.0	32.0	23.0	82.0
Lowest Score	22	17	0	49
Standard Deviation	1.9	5.6	6.6	10.5
Mean as %	85.4%	82.3%	72.1%	80.2%
Coefficient of Variation	0.07	0.17	0.30	0.13



Typically, the second midterm is the hardest exam in this course, with an average score between 2 and 6 points lower than the first midterm. The final exam average score is usually between the two midterm averages. This term was no different: compared to the first midterm, the average score dropped 3.8 points and the median fell 3.5 points. Your performance on questions 1 and 2 was comparable to the first midterm, but the normalization question proved much more difficult than I anticipated. Still, I like my exams to have an average in the low- or mid-80's, and the mean was just over 80, so I consider the results to be satisfactory.

Q1. Access Query Construction: Car Rental

Scores on this problem fell in a rather narrow range between 22 and 29 points out of 30. The class has certainly mastered the basics, but some of the subtler points of queries remain difficult.

My grading scheme here was very simple: I deducted one point per error, up to a maximum deduction of 6 points per query. There were three classes of errors for which I only deducted one point for the first error, and no further points for subsequent similar errors:

- Incorrect ordering of columns with “show” checked (note: ordering of non-shown columns doesn’t matter)
- Confusion between “>” and “>=”, or between “<” and “<=”
- Confusion between “ascending” and “descending” sorting.

(a) For each rental of a minivan starting on or after May 15, 2008 by a customer from New Jersey, show the customer first last name, customer last name, rental start time, and return time. Sort the output alphabetically by customer last name.

Field:	FirstName	LastName	StartTime	ReturnTime	VehicleType	State
Table:	CUSTOMER	CUSTOMER	RENTAL	RENTAL	VEHICLE	ZIPCODE
Total:						
Sort:		Ascending				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteria:			>= #5/15/2008#		Minivan	“NJ”

The date 5/15/2008 needs to be enclosed in “#” symbols, since that is the Access format for a date. Note that #5/15/2008# denotes the instant in time at exactly 12:00 AM midnight at the start of May 15, so >= #5/15/2006# means any date/time during or after May 15, 2008. Strangely, many people seemed to forget that May is the 5th month of the year – I saw many other month numbers besides “5” – but I decided not to deduct for such errors. Since you are not using the “Total:” row, Access will not attempt to group and aggregate records. Therefore, there is no need to add a hidden CustomerID column

(b) For all rentals of a 2008 or newer vehicle to customers from the city of New Brunswick, NJ, show the customer last name, vehicle license plate, manufacturer, and model. Sort the output alphabetically by customer last name. (Note: New Brunswick has more than one zip code.)

Field:	LastName	LicensePlate	Manufacturer	Model	Year	City	State
Table:	CUSTOMER	VEHICLE	VEHICLE	VEHICLE	VEHICLE	CUSTOMER	CUSTOMER
Total:							
Sort:	Ascending						
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteria:					>= 2008	“New Brunswick”	“NJ”

Note that the *Year* field in VEHICLE is a short integer, not a date/time, so the year 2008 should *not* be formatted as a date and enclosed in “#” characters. Again, since you are not using the totals row, there is no need to worry about grouping.

- (c) For each customer from New York (“NY”) or Pennsylvania (“PA”), show the last name, driver’s license number, state, total number of rentals, and total rental revenue..

Field:	LastName	LicenseNumber	State	RentalID	RentalRevenue
Table:	CUSTOMER	CUSTOMER	CUSTOMER	RENTAL	RENTAL
Total:	Group By	Group By	Group By	Count	Sum
Sort:					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:			“NY”		
Or:			“PA”		

The criterion for *State* may also be implemented by a single cell containing “NY” or “PA”.

Note that since no two drivers should have the same combination of *LicenseNumber* and *State*, there should be no danger that two different drivers will be aggregated into one row; therefore an extra hidden grouping by *CustomerID* is not needed.

The *Count* column could use any other field in place of *RentalID*.

Using *Where* instead of *Group by* in the *State* column will not work, because *Where* is incompatible with checking the “show” box. If you use *Where* for the state criterion, you need to repeat the *State* column with *Group by* selected and “show” checked.

- (d) For each zip code outside New Jersey, show the zip code, city, state, total number of rentals ending in 2007 by customers from that zip code, and the total revenue from those rentals. Sort the results by total revenue, highest first. Show only zip codes with at least 5 rentals ending in 2007.

Field:	Zip	City	State	RentalID	RentalRevenue	ReturnTime	ReturnTime
Table:	ZIPCODE	ZIPCODE	ZIPCODE	RENTAL	RENTAL	RENTAL	RENTAL
Total:	Group By	Group By	Group By	Count	Sum	Where	Where
Sort:					Descending		
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteria:			<> “NJ”	>= 5		>= #1/1/2007#	< #1/1/2008#

This query was harder than I intended, because of the nature of the return-time criteria. The constraints on *ReturnTime* above make sure that the return time is any time including or after 12AM at the beginning of new year’s day 2007, and before 12AM on new year’s day 2008. Many people’s criteria were off by one day, and many students only put a one-sided constraint on the return time. You cannot put the two time constraints in the same column, as that would combine them by “or”, whereas you need “and”. A possible syntax using only one column would be *Between #1/1/2008# and #1/1/2008#*, although the result would incorrectly extend one second into 2008 (nobody thought of that answer, though).

The *ReturnTime* criteria must use *Where*, or the results will be grouped by return time as well as by zip code, probably resulting in virtually every rental getting its own output line.

On the other hand, you should use *Group by*, and not *Where*, for the “not New Jersey” criterion, since we want to show the *State* field and we are already grouping by zip code, which is a finer division than state. Remember, using *Where* is incompatible with checking “show”. *Not “NJ”* will work just as well as *< > “NJ”*.

You should not group by additional hidden *CustomerID* or *RentalID* fields in this query, as they would interfere with the desired aggregation by customer zip code.

Finally, note that the *>= 5* criterion on the *Count* column is applied *after* aggregation, as required by the question.

- (e) For each 2008 car in your fleet made by Toyota or Mazda, show the make, model, the total number of times it has been rented, and the average miles driven per rental. The output should have one line for each individual car meeting the criteria. Sort alphabetically by make; within makes, sort alphabetically by model; for cars of the same model, sort by the number of rentals, largest number first.**

Field:	Manufacturer	Model	RentalID	MilesDriven	Year	LicensePlate
Table:	VEHICLE	VEHICLE	RENTAL	RENTAL	VEHICLE	VEHICLE
Total:	Group by	Group by	Count	Avg	Group By*	Group By
Sort:	Ascending	Ascending	Descending			
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteria:	Toyota				2008	
Or:	Mazda				2008	

You need to add the hidden *LicensePlate* “Group By” field in order to keep all 2008 cars with the same make and model (for example, all 2008 Honda Accords) from being aggregated into a single output line. Just the license plate is sufficient because all your cars are registered in the same state, so no two of them can have the same license plate.

If you have “Toyota” and “Mazda” on two separate lines, then you need to repeat the “2008” criterion on both of those lines. If you only have the “2008” criterion in the first of the two rows above, for example, it would apply only to Toyotas, and all Mazdas would be included in the query, regardless of year. Remember that each row is first combined with “and”, and then multiple rows are combined with “or”. Alternately, you can have a single criterion line like

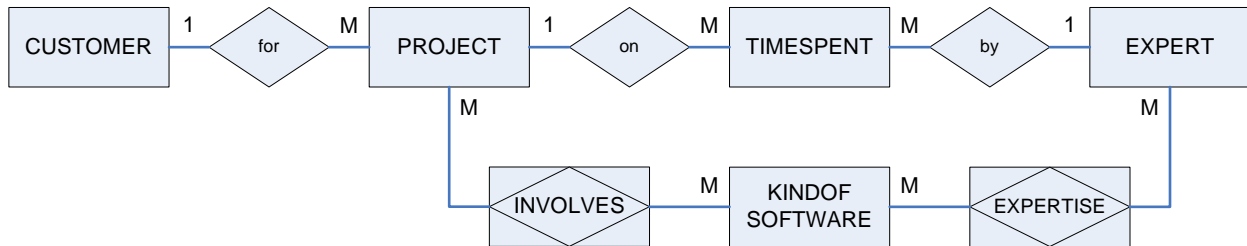
Criteria:	“Toyota” or “Mazda”				2008	
-----------	---------------------	--	--	--	------	--

In this case, the “or” is applied first, because it’s within a single cell.

You could also use *Where* instead of *Group by* in the *Year* column; because you are constraining this column to have only one value and you are not showing that value, either method will give the same result.

Instead of *RentalID*, you could use just about any field in the COUNT column.

Q2. Database Design: Software Experts, Inc.



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

PROJECT(ProjectID, Name, StartDate, EndDate, HourlyRate, CustomerID)
CustomerID foreign key to CUSTOMER

EXPERT(EmployeeID, FirstName, LastName, OfficeNumber,
PhoneExtension, CellPhone, EMail)

TIMESPENT(EmployeeID, ProjectID, DateTimeStarted, DateTimeEnded)
EmployeeID foreign key to EXPERT
ProjectID foreign key to PROJECT

KINDOFSOFTWARE(SoftwareID, Description)

INVOLVES(ProjectID, SoftwareID)
ProjectID foreign key to PROJECT
SoftwareID foreign key to KINDOFSOFTWARE

EXPERTISE(EmployeeID, SoftwareID)
EmployeeID foreign key to EXPERT
SoftwareID foreign key to KINDOFSOFTWARE

You could also view the two one-to-many relationships PROJECT-TIMESPENT-EXPERT as a single many-to-many relationship between PROJECT and EXPERT, so long as the intermediate table between PROJECT and EXPERT stores the right fields and has the right primary key.

The composite attribute (*EmployeeID, DateTimeStarted*) is a valid primary key for TIMESPENT because no employee can be working on more than one project at a time; you could also use *DateTimeEnded* instead of *DateTimeStarted*. For simplicity, it would also be acceptable to choose a synthetic key instead, but then you would lose some built-in defense against overlapping blocks of time for the same employee. You *cannot* use (*EmployeeID, ProjectID*) as a primary key for TIMESPENT, because that would limit each employee to at most one work session on any given project: for example, if “Joe” spent 3 hours working on the “Rutgers website” project last week, then he can never work on the Rutgers website again! One drawback of thinking of PROJECT-TIMESPENT-EXPERT as a single many-to-many relationship is that it tempts you to use this incorrect primary key for TIMESPENT.

My grading scheme (out of 40 points) was as follows:

- 4 points for each table, broken down as follows:
 - 1 point for having the table in your diagram
 - 1 point for having the table in your outline
 - 1 point for having the correct primary key
 - 1 point for having all other necessary fields
- 4 points for each relationship: if an entire relationship missing, misplaced, reversed, or of the wrong type, both in the diagram and outline, you lost the full 4 points. Otherwise, the 4 points broke down as follows:
 - 1 point for having the correct relationship in your diagram
 - 3 points for having a correct foreign key placement and annotations “X foreign key to Y”. For many-to-many relationships, these three points also include having the correct mediating auxiliary table.

A few people tried to turn “past and present” into a double relationship between PROJECT and CUSTOMER, for which there is no real need. I deducted one point for doubling the PROJECT-CUSTOMER relationship on your diagram, and one point for doubling it in your design outline.

Q3. Normalization: Election Results

This problem proved to be much more difficult than I thought it would be. Many people did reasonably well, but a significant number of students seemed flummoxed and scored below 20 points out of 30; scores on this problem were much more variable than on other parts of the exam. I thought there might be some difficulties regarding the field *Ward* (see below), but many students seemed to have much deeper problems. I will try to work a review problem similar to this on the last day of class, so we are better prepared for the final exam.

(a) [1 point] What is a possible primary key for the VOTECOUNT table on the last page? (Note: it may have to be a composite key.)

(Mun#, Ward, Can#) is the most sensible choice. Just *(Mun#, Ward)* does not work – there is more than one row with the same combined value of these two fields.

(b) [2 points] Identify all the partial dependencies in the VOTECOUNT table on the last page. Also, identify all the transitive dependencies in the table.

A partial dependency occurs when a field depends on part of the primary key but not all of it. A transitive dependency occurs when a field depends on another field that is not part of the primary key, and is not a possible alternative primary key. Here,

<i>Mun#</i> → <i>Municipality</i>	is a partial dependency
<i>Can#</i> → <i>Candidate</i>	is a partial dependency
<i>Can#</i> → <i>Off#</i>	is a partial dependency
<i>Can#</i> → <i>Par#</i>	is a partial dependency
<i>Off#</i> → <i>Office</i>	is a transitive dependency
<i>Par#</i> → <i>Party</i>	is a transitive dependency

In addition, we have “chained” combinations of the above:

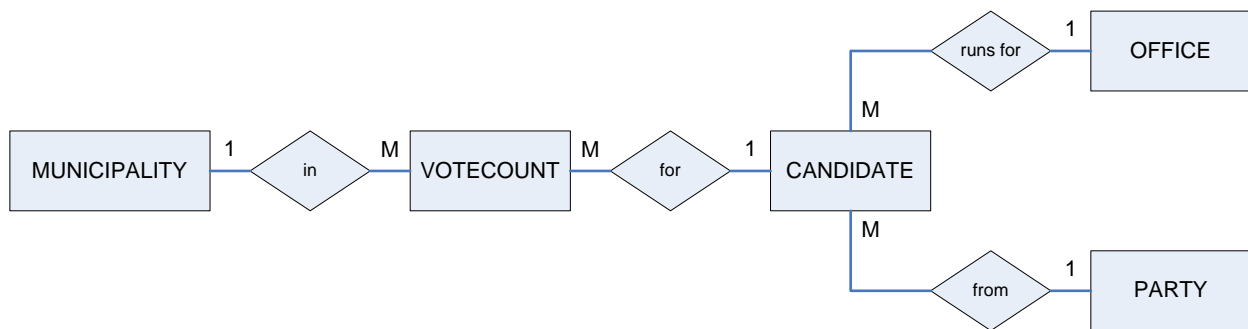
Since $Can\# \rightarrow Off\#$ and $Off\# \rightarrow Office$, we also have $Can\# \rightarrow Office$, which is a partial dependency.

Since $Can\# \rightarrow Par\#$ and $Par\# \rightarrow Party$, we also have $Can\# \rightarrow Party$, which is a partial dependency.

I graded this part of the question very liberally, giving one point for each correct dependency; thus, if you got even two of the eight parts of the correct answer, you got full credit.

Note that a dependency is a relationship between two fields. A field, by itself, cannot be a dependency. Thus, answers like “*Candidate, Office, Ward*” make no sense for this question.

(c) [27 points] Normalize the structure of VOTECOUNT, breaking it up into multiple tables, so the resulting database is in third normal form. Draw an entity-relationship diagram and write a design outline for the resulting database.



MUNICIPALITY(Mun#, Municipality)

OFFICE(Off#, Office)

PARTY(Par#, Party)

CANDIDATE(Can#, Candidate, Off#, Par#)

Off# foreign key to OFFICE

Par# foreign key to PARTY

VOTECOUNT(Mun#, Ward, Can#, Votes)

Mun# foreign key to MUNICIPALITY

Can# foreign key to CANDIDATE

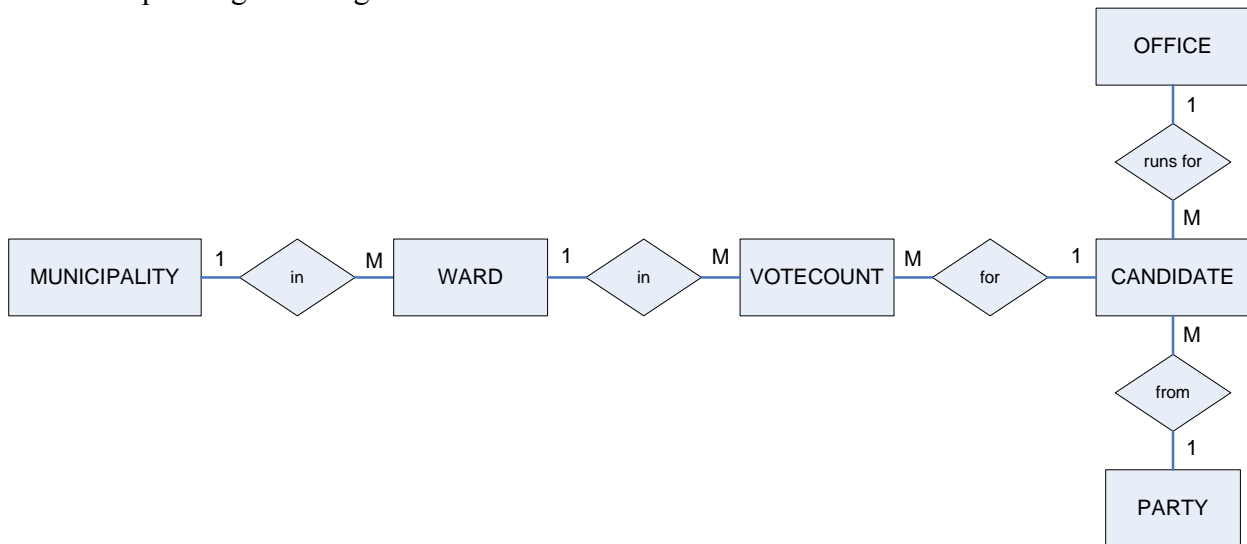
You can also view MUNICIPALITY-VOTECOUNT-CANDIDATE as a single many-to-many relationship between MUNICIPALITY and CANDIDATE.

Note that you do not need a separate table needs for wards (or equivalently ballots), since there is no data to be stored for them except *Votes*, which depends not only on the municipality/ward combination, but also on the candidate. However, it is acceptable to include an extra table for wards if you wish. *If* we have a table for wards, it should have either a synthetic key or the primary key (*Mun#*, *Ward*), since, for example, Huxville ward 1 is not the same place as Brackettown ward 1. With an extra table for wards, we would have the same tables for MUNICIPALITY, OFFICE, CANDIDATE, and PARTY as before, but in place of the VOTECOUNT table above, we instead have (with a synthetic key for WARD):

WARD(WardID, Mun#, Ward)
 Mun# foreign key to MUNICIPALITY

VOTECOUNT(WardID, Can#, Votes)
 WardID foreign key to WARD
 Can# foreign key to CANDIDATE

The corresponding E-R diagram is:



In this case, you can depict WARD-VOTECOUNT-CANDIDATE as a many-to-many relationship between WARD and CANDIDATE.

My grading scheme (out of 27 points) for part (c) was as follows:

- 3 points for each table (in the first solution above), broken up as follows:
 - 1 point for having the table in your diagram and outline
 - 1 point for having the correct primary key
 - 1 point for each missing or misplaced field – for example, one point for putting *Votes* in the wrong table.
- 3 points for each relationship (in the first solution above).