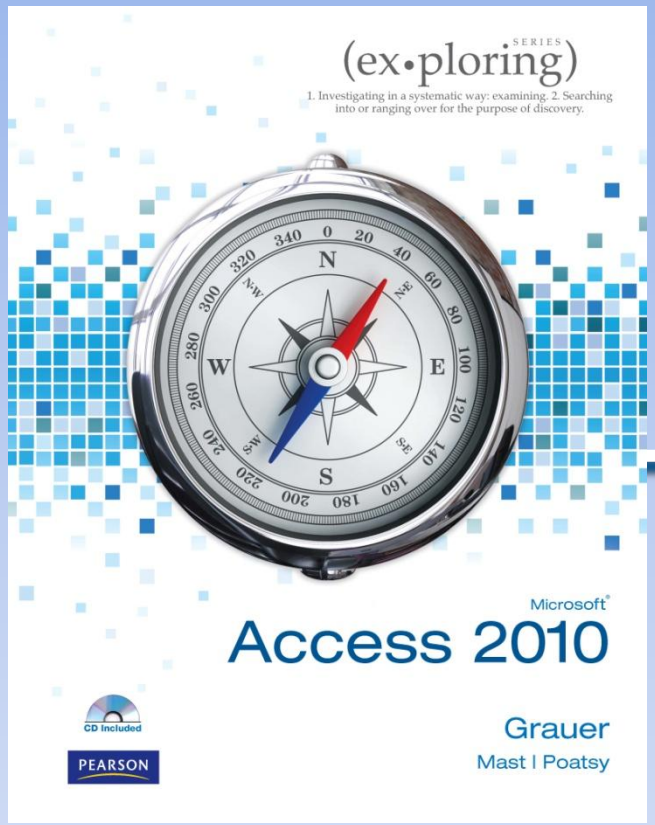


Exploring Microsoft Office Access 2010

by Robert Grauer, Keith Mast, and Mary Anne Poatsy

Chapter 6
Data Validation Techniques

Objectives

- Establish data validation
- Create an input mask
- Create and modify a lookup field
- Set the tab order
- Create a combo box control
- Restrict edits in a form
- Compare fields using a data macro
- Create a query to display missing data
- Create a missing data report
- Create an aggregate report

Establish Data Validation

- **Data validation** — a set of constraints or rules that help control how data is entered into a field
- Various data validation methods include:
  - Required
  - Default value
    - Can set Date() as default value
  - Validation rule
    - Can use wildcards, AND, OR etc like when creating queries
  - Validation text
  - Input Masks
  - Lookup lists

Demo 1: Required Fields and Default Values

- Open a06h1bank
- Modify Customers table with your own name for record C008
- In design view select the phone number field and in the Field Pane set the required property to yes.
- Set the accountType default Value property to Gold
- Click save – click yes for any warnings you get
- Look at new Record in data view and notice the value "Gold"
- Try to enter a new record with no phoneNumber
- After the error message type in a phoneNumber

Demo 2: Validation Rules and Text

- Open loans table in Design View
- Set the validation rule property for the Interest Rate field to **<= 0.15**
- Set the validation text property to **The maximum interest rate is 15%. Please adjust your entry.**
- Save and click yes in response to the warning
- In the datasheet view try to type in 22 as an interest rate
- After the error message set the interest rate to 6.9%
- Add a rule to make sure Term can't be negative. Test it out. Make a rule so Dates can't be added that are in the future.

Create an Input Mask

- Database designers can control **how** users enter data into tables by:
  - Selecting the appropriate data type when adding fields to a table, e.g., Text, Number, Currency, and Date/Time
  - Creating an ***input mask*** to further restrict the data being input into a field by specifying the exact format of the data entry.

Create an Input Mask (continued)

- Two ways to create input masks:
  - Typing the appropriate characters into the input mask field
  - Using the ***Input Mask Wizard*** to generate an input mask for a field based on responses to a few questions

Create an Input Mask (continued)

TABLE 6.1 Some Common Input Mask Characters and Uses		
Character	**Description**	**Requires Entry**
0	Digit (0 to 9) Plus + and Minus – not allowed	Yes
9	Digit or space Plus + and Minus – not allowed	No
#	Digit or Space Plus + and Minus – allowed Spaces display as blanks but are not stored	No
L	Letter (A to Z)	Yes
?	Letter (A to Z)	No
A	Letter (A to Z) or digit	Yes
a	Letter (A to Z) or digit	No
<	Converts all characters entered to lowercase letters	No
>	Converts all characters entered to uppercase letters	No
****	Displays the next character as literal: \# displays #	No
. (period)	Decimal placeholder	No
- or /	Date separators (6-5-2010 or 6/5/2010)	No
!	Forces the value to be input from left to right	Not Applicable

Correction: ! Forces the value to be stored from right to left

Demo 3: Input Mask

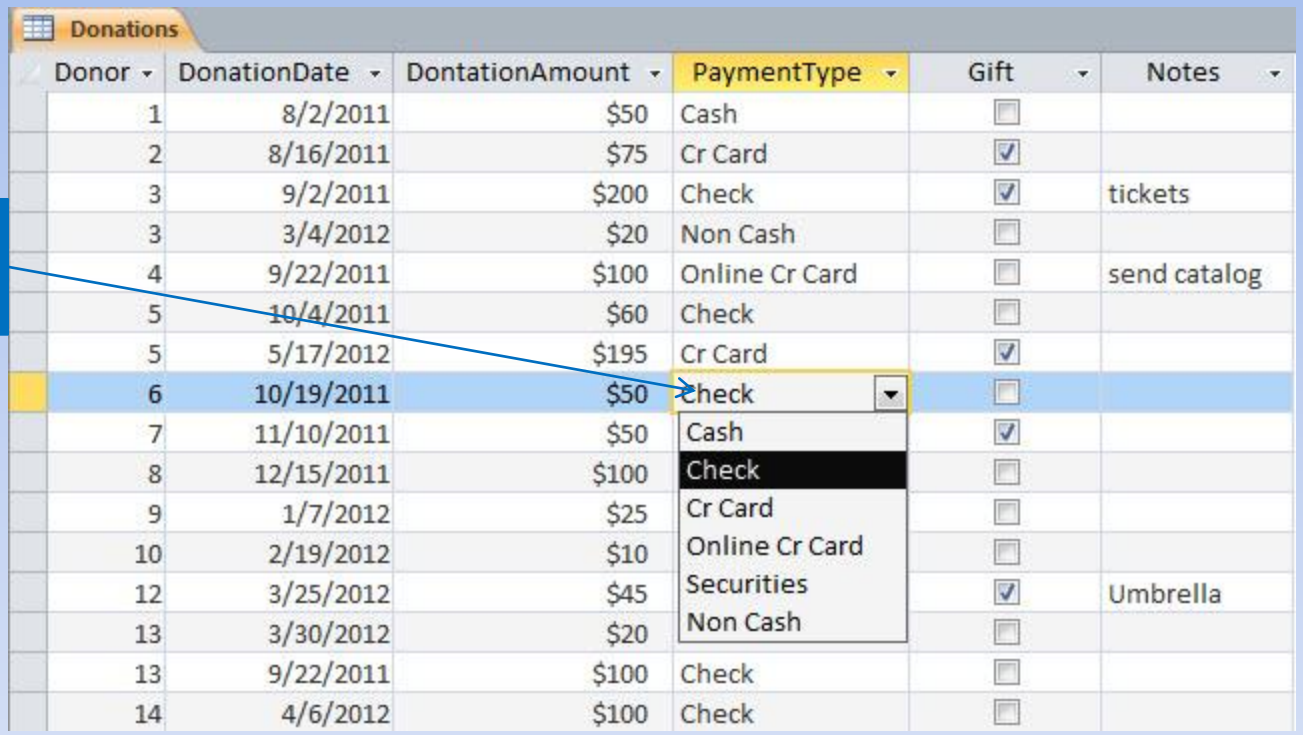
- Open customers table in design view
- Click on the PhoneNumber Field then the Input Mask Property and click Build on the right had side to activate the wizard.
- Try typing input in the try it box.
- Click next twice then click the With the symbols in the mask option. Click next then Finish.
- Save table and test typing phoneNumbers in the Datasheet view. (Press ESC to return record to original state)

Create and Modify a Lookup Field

- **Lookup field** — provides the user with a finite list of values to choose from
- **Lookup Wizard** — helps to create the lookup field
  - Asks you six questions
  - Uses your answers to create the options list

Create and Modify a Lookup Field (continued)

Payment type with lookup options



Donations

Donor	DonationDate	DontationAmount	PaymentType	Gift	Notes
1	8/2/2011	$50	Cash	☐	
2	8/16/2011	$75	Cr Card	☑	
3	9/2/2011	$200	Check	☑	tickets
3	3/4/2012	$20	Non Cash	☐	
4	9/22/2011	$100	Online Cr Card	☐	send catalog
5	10/4/2011	$60	Check	☐	
5	5/17/2012	$195	Cr Card	☑	
6	10/19/2011	$50	Check	☐	
7	11/10/2011	$50	Cash	☑	
8	12/15/2011	$100	Check	☐	
9	1/7/2012	$25	Cr Card	☐	
10	2/19/2012	$10	Online Cr Card	☐	
12	3/25/2012	$45	Securities	☑	Umbrella
13	3/30/2012	$20	Non Cash	☐	
13	9/22/2011	$100	Check	☐	
14	4/6/2012	$100	Check	☐	

Demo 4: Lookup Fields

- Note: We have an existing table called LoanTypes with the values for the lookup fields already.
- Open Loans table in design view.
- Click the Data Type field for LoanType. Click the arrow and choose Lookup Wizard.
- Click the I want the lookup field to get the values from another table or query option Click Next and Choose the Table:LoanTypes Click Next
- Click All Fields (>>) Click Next
- Sort on Type. Click Next. Click Next, Click Finish
- Go to the DataSheet view and add a record.

Data Validation in Forms and Data Macros

- Data validation rules created for a table will also apply to any forms based on the table.
- However validation rules created for a form do not apply to the underlying table.
- Thus most database designers apply data validation rules to tables first before creating forms.
- There are other techniques to ensure valid data that may also be used in forms.

Setting the Tab Order

- ***Tab order*** — the sequential advancing in a form from one field or control to the next when you press Tab
- If tab order is not logical users might accidentally enter data into the wrong field.
- To set the tab order in a form:

1. Switch to Design view
2. Click Tab Order in the Tools group on the Design tab
3. Click Auto Order if the form has a Stacked layout and you want to enter data from top to bottom
4. Click OK to accept the changes
5. Close the Tab

Set the Tab Order (continued)

- Remove a Tab Stop
  - When you want the tab order to skip a field completely, e.g., a calculated field that does not require data entry
  - Set the Tab Stop property to No for that field
- Remove Tab Stop from Unbound Controls
  - Unbound controls do not require data entry, e.g., a Close Form command button, a calculated field.

Create a Combo Box Control

- **Combo box control** — provides a list of options from which the user can choose a single value
- If you create a form based on a table with a lookup field, the lookup field will become a combo box control on the new form. A ***combo box control*** provides a list of options from which the user can choose a single value.

Restrict Edits in a Form

- When too many users make changes to the data, the data can become unreliable and difficult to maintain
  - Protect data in a database by restricting casual users from editing the data
  - Casual users may need access to customer information or order details, but they may not need permission to add, edit, and delete
- Easier to maintain data integrity if only a select group of users is allowed to enter and edit data

Restrict Edits in a Form (continued)

- Create a Lookup Copy of a Form
  - When casual users need to look up information without making changes
  - After you make a copy of the original form, name the copy as Lookup Form
  - Limit the Lookup Form to only allow viewing data by setting the Allow Edits Property to No

Demo 5: Create a lookup copy of a form

- Make a copy of the customers form called Lookup Customers and open in Design View.
- In the Property Sheet select form, select the data tab and change the allow edits property to no.
- Change the allow additions property to no and the allow deletions property to no.
- Switch to form view and test your changes.

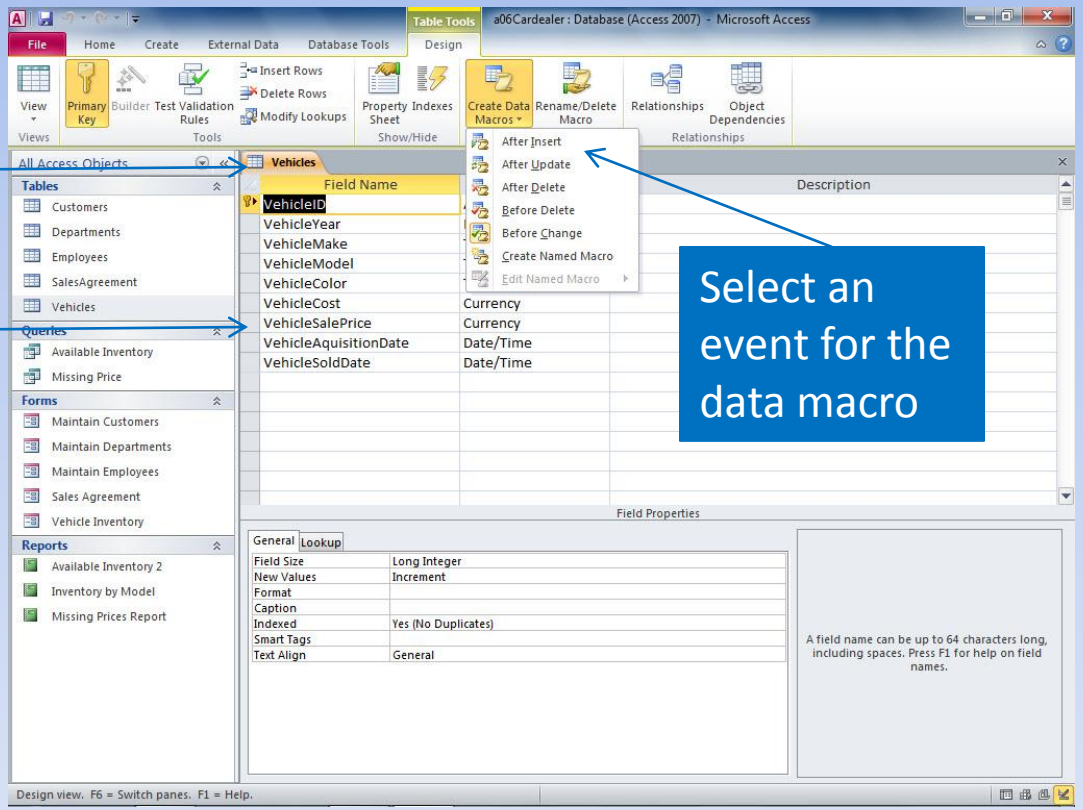
Compare Fields using a Data Macro

- You can't do things like compare two fields using data validation rules.
- **Data macro —** executes a series of actions when a table event occurs or whenever a named macro is executed
  - Two types of data macros:
    - Event-driven
    - Named (we will not look at these ones in this chapter)
- **Table event —** occurs naturally as users enter, edit, and delete table data

Compare Fields using a Data Macro (continued)

Design view of Vehicles table

Selecting a field is not required



Select an event for the data macro

Demo 6: Using Macros to Compare Fields

- We want to make sure personal loans can't be for a period of greater than 5 years.
- Open Loans table in Design View
- Click Create Data Macros and click Before Change.
- In the macro builder
  - Click the Add the New Action Arror and choose If
  - Type [LoanType]="p" and [Term] > 5
  - Select RaiseError from the Add New Action Error and type 10 for the Error Number

Error Detection Techniques

- Validating data in tables and forms is the best way to keep invalid data out of your database.
- Also good to check for errors after data has been entered.
- Queries and Reports can help find invalid data.

Create a Query to Display Missing Data

1. Create a query based on the table that you want to check to display records with missing information
2. Add criteria to check for missing data in key fields, e.g., Last Name or Order Date.
3. Checking for missing data in all the fields may become overwhelming because some are blank half the time

Create a Query to Display Missing Data (continued)

- Create a Query to Check for Null Values
  - Choose the table to use
  - Add the required fields in Query Design
  - Add the Is Null criteria to the fields that should not be blank
  - Make sure that the Is Null criteria for the fields are on separate rows if you want to create the OR condition

Creating a Missing Data Report

- Use the Report Wizard to Create a Report:
  - Click Report Wizard in the Reports group on the Create tab
  - Choose the correct table or query to base the report on (e.g., use the Missing Price Query)
  - Add the fields you want using the Add One Field (>) button
  - Click at the remaining prompts until the report is displayed

Exercise: Finding Invalid Data

- Create a query on the customer data to find records where the Address, City, State,ZipCode or PhoneNumber fields are null.
- Modify data and run it to test it works.
- Use your query to create a missing data report.
- Add any missing data and open the report again, no records should appear.

Summary

In this chapter, you learned how to apply the various data validation techniques using tables, forms, and reports to reduce errors while managing the database.

Normal Forms, Fine-Tuning the Database

Based on
notes on normal forms by Fred Coulson
and slides by
Robert Grauer, Keith Mast, Mary Anne Poatsy

Objectives

- Transform tables to First Normal Form
- Transform tables to Second Normal Form
- Transform tables to Third Normal Form
- Use the Database Documenter Tool
- Use the Performance Analyzer Tool
- Use the Table Analyzer Tool
- Use the Database Splitter Tool

Database Normalisation

- **Normalisation** – the formal process of deciding which fields should be grouped together into which tables
- Rules of normalization help you correctly design or fix a database.
- Benefits of normalization are
  - Minimise data redundancy
  - Improvement of referential integrity enforcement
  - Ease of maintaining data (add,update, delete)
  - Accommodation of future growth of database
  - Helps keep tables free of anomalies

Data Normalization cont…

- **Anomaly** – an error or inconsistency that occurs when you add, edit, and delete data
  - Update anomaly
    - To update a value, we would have to update multiple rows to avoid inconsistent data
  - Insert anomaly:
    - Unable to insert valid data
  - Delete anomaly
    - Unable to delete without losing valid data

Anomaly Example

StudentID	StudentName	StaffID	StaffName
z1	Dave	S2	MA
z2	Greg	S1	WE
z3	Tina	S2	MA

- **Insert Anomaly:** We can't store information about a staff member who has no students
- **Delete Anomaly:** Deleting student z2 would result in losing information about staff member S1
- **Update Anomaly:** If S1 changes her name, we would have to update it in multiple rows. If we didn't we would have inconsistent data

Normal Forms Summary

- First Normal Form (1NF):
  1. A tuple *cannot contain repeating groups* of similar data (atomicity),
  2. Each tuple must have a unique *primary key* value.
- Second Normal Form (2NF):
  - 1NF **and** no non-key field may be dependent on a subset of a concatenated key
- Third Normal Form (3NF):
  - 2NF **and** no field may be dependent on a non-key field
- The normal form rules are sometimes deliberately violated in a DB design in order to achieve faster query results (e.g. by avoiding a join operation). If this is done, then steps must be taken to avoid anomalies.

The Problem

Pressed Rat & Warthog
23, The Oaks,
Little Whinging, Surrey

Invoice No.: 1729
Issue Date: 1 August, 1968

To: Captain B. Madman
Badde Manor
The Glebe, Herts.

Customer ID: 666

INVOICE

Quantity	Item Code	Item Description	Unit Price	Amount
77	123	Atonal Apples	£9	£693
1	3	Amplified Heat	£99	£99
42	965	Dog legs	£2	£84
			Total, please pay	£876

- How do we represent this in a database?
- Notice that it has a group of fields (Quantity, Item Code, Item Description, Unit Price, Amount) that is repeated 3 times, and *could* be repeated indefinitely

One Possibility

Invoice No	Cust ID	Cust Name	Cust Address	Invoice Date	Item IDs	Item Descs	Item Qtys	Item Prices	etc
1729	666	Madman, B	Badde Manor	1972-04-03	123,3, 965	Atonal Apples, Amplified Heat, Dog legs	77,1, 42	9,99,2	…

- This (terrible) approach has a primary key – the invoice number.
- However there are multiple values in Item IDs, Item Descs, Item Qtys and Item Prices
  - These fields are not atomic and contain repeated groups of data within the fields so the table violated First Normal Form
- We could create fields like Item ID1 Item ID2 and Item ID3 etc but not all orders are going to have exactly 3 items in them
  - This kind of table would have repeating groups of fields and would violate First Normal Form.

More on the Problem

- The user may want to ask queries like "How much Amplified Heat was sold in April, 1973?"
- This would become difficult if a table with repeating fields was used, as each of several Product description fields would have to be checked.
- So maybe we do something like this – for each item in the invoice, we have a tuple:

Invoice No.	Cust. ID	Cust. Name	Cust. Address	Invoice Date	Item ID	Item Desc.	Item Qty	Item Price	Item Total	Invoice Total
1729	666	Madman,B.	Badde Manor.	1972-04-23	123	Atonal …	77	9.00	693.00	876.00
1729	666	Madman,B.	Badde Manor.	1972-04-23	3	Amplified ..	1	99.00	99.00	876.00
1729	666	Madman,B.	Badde Manor.	1972-04-23	965	Dog legs	42	2.00	2.00	876.00

First Normal Form

- The table no longer has records with repeating groups
- The primary key in this case is a *concatenated* or *composite* key, consisting of Invoice No. & Item ID.

Invoice No.	Cust. ID	Cust. Name	Cust. Address	Invoice Date	Item ID	Item Desc.	Item Qty	Item Price	Item Total	Invoice Total
1729	666	Madman,B.	Badde Manor.	1972-04-23	123	Atonal …	77	9.00	693.00	876.00
1729	666	Madman,B.	Badde Manor.	1972-04-23	3	Amplified ..	1	99.00	99.00	876.00
1729	666	Madman,B.	Badde Manor.	1972-04-23	965	Dog legs	42	2.00	2.00	876.00

Composite primary key

- So our new representation *does not* have repeated groups; & *does* have a unique primary key in each tuple → 1NF

Second Normal Form

- Notice that there is still a problem:
  - If the price of an item is updated in one record it will not be updated everywhere as it may exist more than once… such an inconsistency is termed an *anomaly* in the DB.
- 2NF: *no partial dependencies on a concatenated key*
- In more detail, for each (non-primary-key) field:
  - can this field exist *without* one or the other part of the concatenated primary key?
  - If the answer is "yes", then the table is not in 2NF
- In our case, InvoiceDate depends on InvoiceNo, but not on ItemID, but ItemID is part of the primary key. ☹

Invoice No.	Cust. ID	Cust. Name	Cust. Address	Invoice Date	Item ID	Item Desc.	Item Qty	Item Price	Item Total	Invoice Total
1729	666	Madman,B.	Badde Manor.	1972-04-23	123	Atonal …	77	9.00	693.00	876.00
1729	666	Madman,B.	Badde Manor.	1972-04-23	3	Amplified ..	1	99.00	99.00	876.00
1729	666	Madman,B.	Badde Manor.	1972-04-23	965	Dog legs	42	2.00	2.00	876.00

Second Normal Form 2

- What about the other fields?

Field	Meets 2NF criterion?	
Invoice No.		(part of primary key)
Cust. ID	?	Depends on *neither* Invoice No. *or* Item ID
Cust. Name	?	As above
Cust. Address	?	As above
Invoice Date	X	Independent of Item ID: fails 2NF
Item ID		(part of primary key)
Item Desc.	X	Depends on Item ID but not on Invoice No.
Item Qty	√	Depends on both Item ID and on Invoice No.
Item Price	X	Independent of Invoice No.
~~Item Total~~	–	Calculated value – doesn't really belong in table at all - ignore
~~Invoice Total~~	–	Calculated value – doesn't really belong in table at all - ignore

Second Normal Form 3

- What do we do now?
- Take out the second part of the concatenated primary key and put it in its own table, along with all the fields that depend (only) on it.
- The other fields stay where they were.
- This gives us tables called
- *Invoices* and *InvoiceItems*

Invoice No.	Invoice Date	Cust. ID	Cust. Name	Cust. Address
1729	1972-04-23	666	Madman,B.	Badde Manc

Invoice No.	Item ID	Item Desc.	Item Price	Item Qty
1729	123	Atonal …	9.00	77
1729	3	Amplified …	99.00	1
1729	965	Dog legs	2.00	42

- InvoiceNo comes over to *InvoiceItems* so that each InvoiceItem can "remember" which Invoice it is part of.
- There is a one-to-many relationship between Invoices and InvoiceItems.

Second Normal Form, *Again*

- Invoices table now passes 2NF, since it doesn't have a concatenated primary key. We now perform the same sort of 2NF analysis on our new table, InvoiceItems, which still has InvoiceNo & ItemID as primary keys.

Field	Meets 2NF criterion?	
Invoice No.		
Item ID		
Item Desc.	X	Depends on Item ID, but not on Invoice No.
Item Price	X	Depends on Item ID, but not on Invoice No.
Item Qty	√	Depends on both

- The process is similar (not identical) to before – create a new table, *Items*…

Second Normal Form, Again, ctd

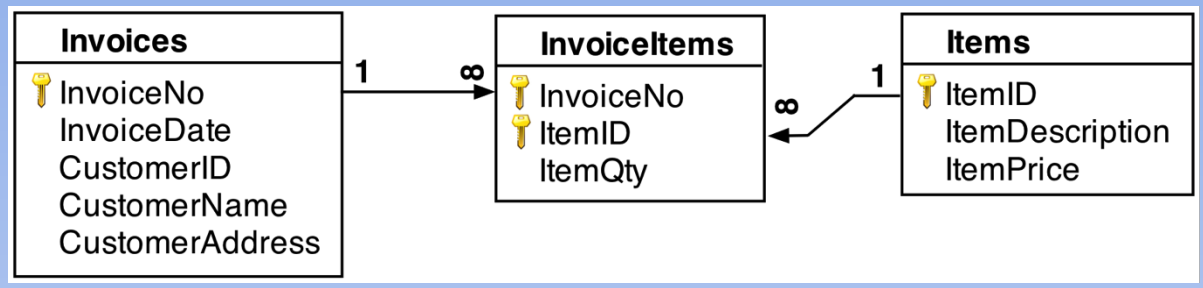
- We get the *Invoices* table as before, plus

Items		
Item ID	Item Desc.	Item Price
123	Atonal …	9.00
3	Amplified …	99.00
965	Dog legs	2.00

InvoiceItems		
Invoice No.	Item ID	Item Qty
1729	123	77
1729	3	1
1729	965	42

- The first time, we took out *all* the fields that relied on ItemID, now we only take the fields that fail the test – so ItemQty stays put.
- This is because, the first time, we removed ItemID from Invoices altogether, because of the one-to-many relationship between *Invoices* and *InvoiceItems*. So ItemQty *had* to go into the new table.
- This time, ItemID is <u>not</u> removed from *InvoiceItems*, so, since ItemQty does not violate 2NF, it stays put.

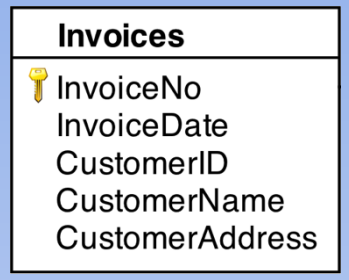
Second Normal Form, Again, ctd 2



- *Items* and *Invoices* "pass" 2NF, since they don't have a concatenated primary key. *InvoiceItems* passes 2NF, since ItemQty genuinely depends on both parts of the key.
- So we're done with 2NF.
- Remember the problem with update anomalies? It's still present, since a customer can have more than one invoice, and at present, the address is stored with each invoice so if the address is changed in one record the data may become inconsistent.

Third Normal Form

- 3NF requires *no dependencies on non-key attributes* - also referred to as *transitive dependencies*
- CustomerName and CustomerAddress in the new *Invoices* table depend on CustomerID, not on the key (InvoiceNo).
- This violates 3NF.
- Solution – make a new table, *Customers*, with customer info in it, and CustomerID as the primary key:



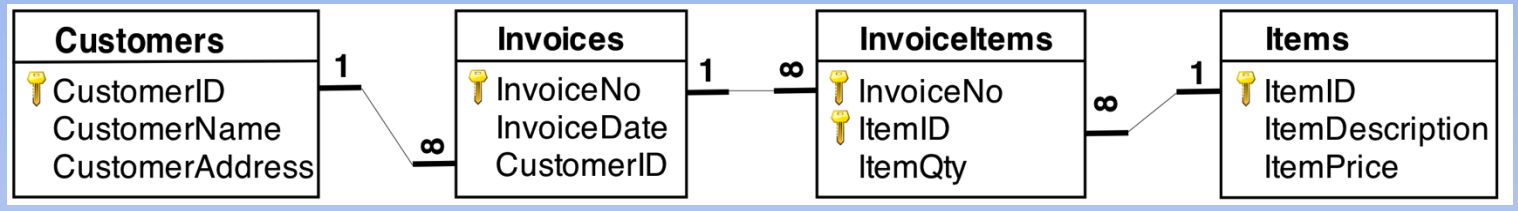
Cust. ID	Cust. Name	Cust. Address
666	Madman,B.	Badde Manor

Invoice No.	Invoice Date	Cust. ID
1729	1972-04-23	666

- The other info, including a copy of CustomerID, stays in *Invoices*

Third Normal Form

- The relationship diagram now looks like this:



- All four tables now meet the criteria for 1NF, 2NF, and 3NF.
- The potential update anomaly has been removed, since the CustomerName and CustomerAddress have each been stored exactly once, in the Customers table.
- Customer info can still be wrong ☹, but cannot be different in different places in the DB.

Normal Forms

- The material on normal forms follows Fred Coulson's http://www.phlonx.com/resources/nf3/. A copy of this material in printable form can be found at http://www.phlonx.com/resources/nf3/nf3_tutorial.pdf
- Chapter 9 of the Grauer Access Comprehensive reference book also covers normal forms. This version uses a different example. However, Grauer's version of 2NF is wrong, so ignore it.

Use the Table Analyzer Tool

- **Table Analyzer** – analyzes the tables in a database and then normalize the tables
- Use results to:
  - Rename tables
  - Split tables
  - Rearrange fields in tables
  - Create relationships between tables

Demo 1: Table Analyzer

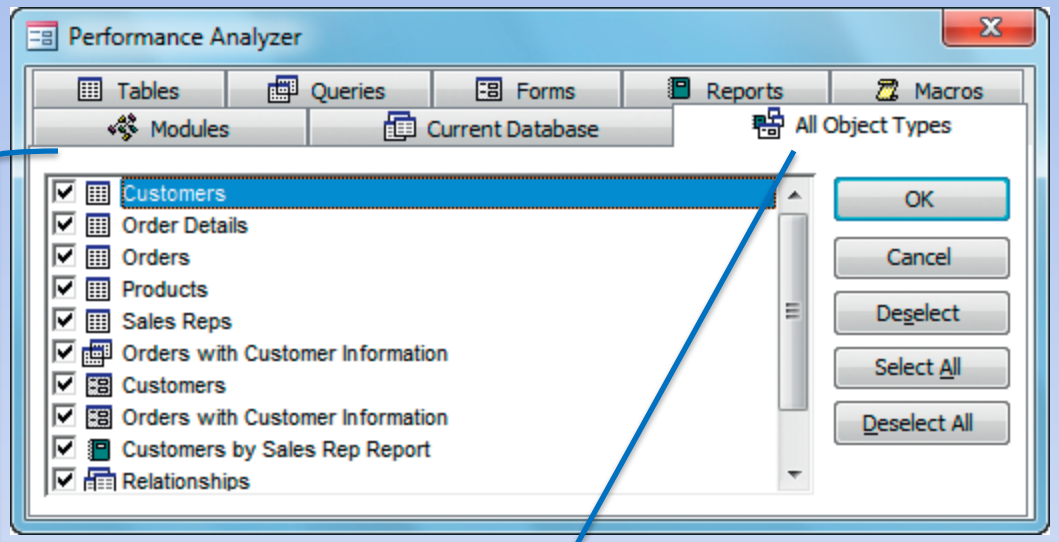
- Open trainers table and type your first name and last name into a new record. Close the table
- Click the database tools tab and click Analyze Table. In the wizard click next twice
- Verify the Animals table is selected. Click Next twice.
- Click rename Table, type Animals_new and click ok. Click Table 2, click Rename Table and type Trainers_new then click OK.
- Click next. Click the AnimalID field in the Animals_new table and then click Set Unique Identifier to set the primary key. Click next twice.
- Verify the No, don't create the query option and then click Finish. Click OK in the response to the warning.

Use the Performance Analyzer Tool

- **Performance Analyzer** – used to evaluate the design of the database and make recommendations for optimizing the database
- The Performance Analyzer lists three kinds of analysis results:
  - Recommendations
  - Suggestions
  - Ideas

Use the Performance Analyzer Tool (continued)

Figure 9.16 Performance Analyzer Dialog Box



All objects are selected for analysis

All Object Types tab selected

Demo 2: Performance Analyzer

- Open a09h1zoo.
- Click the Database Tools Tab and click Analyze Performance in the Analyze group.
- Click the All Object Types tab, click Select All and then click OK
- Establish relationships between
  - Diet and Animals table: AnimalID
  - Animals and Exhibits table: ExhibitID
  - Animals and Trainers table: TrainerID
- Run performance analyser again.

Use the Database Splitter Tool

- **Database Splitter** – enables you to split a database into two files:
  - Back-end database
    - Contains the data tables
    - Is typically placed on the server
  - Front-end database
    - Contains all the other database objects
    - Placed on each individual user's computer
  - In the Database Tools tab click Access Database in the Move Data group.

Create a Menu System

- **Navigation Form** – helps users open the forms and reports they need quickly

Demo 3: Navigation Form

- In the create Tab, click Navigation in the Forms group. Select the Horizontal tabs option
- Drag forms and reports one at a time from the navigation pane onto [Add New].
- Switch to form view and test the navigation panel.

Encrypt and Password-Protect a Database

- Access 2010 incorporates encryption methods to help keep your databases secure
- **Encryption** – the process of altering digital information using an algorithm to make it unreadable to anyone except those who possess the key (or secret code)
- **Open Exclusive** – guarantees that you are the only one currently using the database

Demo 4: Encrypt and Password Protect a Database

- Open database in exclusive mode
  - When you open the file, click the arrow next to the open button in the open dialogue and choose open exclusive
- Click the File tab and click Encrypt with Password.
- Create a password and click ok in response to the warning message.
- Close the database and open normally and test the password.

Summary

- Majority of database designs only require the first three normal forms
- Access provides the following useful tools to improve the performance of a database:
  - Performance Analyzer tool
  - Table Analyzer tool
  - Database Splitter tool
- Access provides password protection and creation of navigation panels.