

## Forms

Forms let you view your data in perhaps a more convenient fashion than the regular table view. Forms also let you integrate data from more than one table and display it on a single form. You can also arrange the fields in a convenient manner, and display just the fields needed at the moment. A simple form displays just a single record's data, but a multi-record form can display more than a single record.

After defining a form, you can use it at any time. You can use the form for entering data, editing data, or just for viewing data.

Each of your Paradox tables can have up to 15 different forms. You might have one form that presents the fields in an order that's convenient for entering data. You might have a second form that presents the fields in a convenient form for viewing and editing. A third form might present only the fields that are of interest for a particular job.

### The Paradox Standard Form

Each table has a *standard form* that you can use. The standard form is what you get when you press *F7 Form Toggle*. The form is very plain, but does illustrate the important parts of a form.

Viewing Students table with form F: Record 1 of 10

Main ==▼

	Students	#	1
Sn:	1000		
Last Name:	Smith		
First Name:	William		
Address:	78 Rye Street		
City:	Topeka		
State:	KS		
Zip:	66699		
Areacode:	316		
Phone:	285-5944		
Date Enrolled:	9/10/86		

At the top of the screen, Paradox tells us the name of the table we're viewing, the number of the form we're viewing (the standard form is form F, the rest are numbered 1 through 14), the current

record number, and the total number of records in the table. Then, Paradox uses the field names as labels in a column at the left side of the form, and displays the contents of the fields in a second column to the right of the labels.

## Moving Through Forms

---

When you're using the form view, some of the movement keys work differently than how they do in table view. Most notably, the Up arrow and Down arrow keys move from field to field, rather than from record to record as they do in table view. Use Page Up and Page Down to move from record to record. (If the form has more than one page, Page Up and Page Down will move to the other pages of the form. Then, they move to the next record.) Home and End move to the first and last records of the table. Control-Home and Control-End move to the first and last fields of the form.

## Designing a Form

---

To design a form, issue the *Forms Design* command. Then, select the table that the form is for. You can type the table name, or press the Enter key to get a list of tables to choose from. Then, select the form number that you want to design. As with reports, each table has a standard form (called form F), and up to 14 other forms, known as forms 1 through forms 14. Finally, fill in a description for the form. This description can be up to 40 characters, and serves as a better description of the form than just a number 1 through 14.

After completing the form description, you'll see the form design screen. On the form design screen, you can use the arrow keys to move about, type text, and use the *F10 Menu* key to issue commands. Use commands to place fields on the form and to do other things.

Near the top left of the screen, you'll see the position indicator, which tells the position of the cursor. A display like <10,44> means row 10, column 44. You can use this information to help align items on the form. Near the top left of the screen, you may see *Ins*, which means we're in insert mode. While in insert mode, your typing is inserted at the cursor position, and material to the right of the cursor moves further right to make room for your typing. When not in insert mode, your typing replaces existing text. Press the *Insert* key to switch to and from insert mode. Also, at the top right, you'll see an indication of the page of the form that you're viewing. *1/1* means we're viewing page one of a one-page form. *2/3* means you're viewing page two of a three-page form.

## Entering Text

---

You can enter text on a form by moving the cursor to a location and typing the text. There is no need to press the Enter key after typing the text.

Often, it's important to place text in a precise location. For example, the heading *Computer University* on this form is centered. Since Paradox has no centering command for this

---

application, just move the cursor to what looks like a good starting place and type the words. After typing, you can refine the placement of the text. To move text to the left, place the cursor somewhere to the left of the text and press the *Delete* key. To move text to the right, again place the cursor to the left of the text. Then, if necessary, press the *Insert* key so that you're in insert mode (you'll see *Ins* near the top right of the screen). Then, press the space bar as needed.

After typing the heading for the form, the next step is to start placing the field labels for the form. Here's the structure of the *Students* table:

STRUCT	Field Name	Field Type
1	Sn	A4
2	Last Name	A15
3	First Name	A15
4	Address	A20
5	City	A14
6	State	A2
7	Zip	A5
8	Areacode	A3
9	Phone	A8
10	Date Enrolled	D

Notice, for example, that the field name for the student number is *Sn*. But there's no reason why you can't use *Student Number* on the form. Also, notice that the three fields *City*, *State*, and *Zip* are identified as a group, rather than individually. This is fine as long as the identification of the fields is clear to the users of the form.

## Placing Fields

After entering some of the field labels, a good step is to place fields from the table on the form. The main command is *Field Place* from the menu. The types of fields that you can place are:

Regular	A regular field from the table. You will be able to view and edit the information in these fields.
DisplayOnly	A regular field from the table that you can view, but not change. The cursor won't move to a DisplayOnly field. You might want to use these fields when you want to let people view a field, but not change it.
Calculated	A field calculated from other fields in the current record of the table. Since a calculated field doesn't exist in the table, but is instead calculated as needed, you can't change a calculated field.
#Record	Displays the record number as a field. You can't change the contents of this field.

Generally, most of the fields that you'll place will be regular fields.

Next, identify the field to place. From the list of fields, you can type the first letter of the field's name. If there's only one field starting with that letter, you're ready for the next step. If there's

## Paradox Level 2

---

more than a single field starting with that letter, Paradox narrows the list to just those fields beginning with that letter. Then, use the arrow keys to highlight the field name, and press the Enter key.

Now, use the arrow keys to place the beginning of the field and press the Enter key. Then, use the arrow keys to adjust the width of the field, and press the Enter key again. For alphanumeric fields, Paradox initially shows the width of the field as it is defined in the table structure. You can make it narrower if you like, but you might not be able to see the entire contents of the field. For numeric or data fields, you can make the field smaller, but if the field value is too big to show in the width, Paradox will display asterisks (\*\*\*) instead.

Paradox shows the space reserved for displaying a field with underscores. As you move about the form and move to these underscores, Paradox shows the field name near the top right corner of the screen. After placing a field, you can move it to the left or right just as you'd move text to the left or right.

When Paradox shows the list of fields to place while you're placing regular fields, you'll see only those fields that haven't yet been placed. In other words, you can't place a regular field twice on a form. You can place a field as a DisplayOnly field as many times as you like.

To erase a field from the form, issue the *Field Erase* command from the menu. Move to the field to erase and press the Enter key. To adjust the width of a field, use the *Field Reformat* command. Move to the field to adjust and press the Enter key. Then, adjust the width of the field and press the Enter key again.

## Drawing Borders and Lines

---

You can draw borders and lines on the form for decorative and dividing purposes. The command to use is *Border Place*. Next, you can select a single line border, a double line border, or a border made of any other character. Then, you'll need to place the cursor at one corner of the border that you'd like to draw and press the Enter key. Then, move the cursor to an opposite corner of the box or line and press the Enter key again.

In this example form, we made a double line border around the entire form. To make a line, such as the line beneath the *Computer University* title, issue the *Border Place* command and select either *Single* or *Double* as you need. Then, move to one end of the line and press the Enter key. Move to the other end of the line and press the Enter key again.

To erase a border, issue the *Border Erase* command. Move to the start of the border that you want to erase and press the Enter key. Move so that the rest of the border is highlighted and press the Enter key again.

## Completing the Form Design

To complete the designing of the form, press the *F2 Do\_It!* key or issue the *Do\_It!* command from the menu. At this time, Paradox saves the form. You can cancel out of designing a form by issuing the *Cancel* command from the menu.

Designing new F1 form for Students  
<10,44>

Form Ins 1/1

Computer University

---

Student Number: \_\_\_\_\_ Date Enrolled: \_\_\_\_\_

Last Name, First Name: \_\_\_\_\_

Address: \_\_\_\_\_

City, State, Zip: \_\_\_\_\_

Area Code and Telephone: \_\_\_\_\_

## Using a Form to View Data

After creating a form, you'll want to use it for viewing or editing data. The *F7 Form Toggle* key is the key that switches from table view to form view. Normally, this key uses form F, the standard form. To use a different form, issue the *Image Pickform* command. From the menu of forms, pick the form that you want to use.

Using *Image Pickform* tells Paradox to use this form for the current moment only. If you clear the table with *F8 Clear* or *Alternate-F8 Clear All* and then view the table again and press *F7 Form Toggle*, you'll see form F, the standard form again. To make a different form than F be the default form for the table, follow these steps: First, issue the *Image Pickform* command and select the desired form. Then, issue the *Image KeepSet* command. Paradox updates the *.set* file for this table to remember the default form. Now, when you view the table and press *F7 Form Toggle*, Paradox will use this form.

You can also use the form for data entry when you use the *Modify DataEntry* command. After

issuing this command, press *F7 Form Toggle* or issue the *Image Pickform* command as needed.

Viewing *Students* table with form *F1: Record 1 of 10*

Main ==▼

Computer University	
Student Number: 1000	Date Enrolled: 9/10/86
Last Name, First Name: Smith	William
Address: 78 Rye Street	
City, State, Zip: Topeka	KS 66699
Area Code and Telephone: 316 285-5944	

### A Form for Two Tables

A very valuable feature of Paradox forms is their ability to show data from two different tables on the same form. This is most useful when your data is in a *many-to-one* relationship, where many records from one table are related to a single record in a different table. Our example consists of students and the classes they've taken at a college or university. In this case, each student takes many classes. Here's the structures of the two tables in question:

Structure of <i>Students</i> Table	Structure of <i>Classes</i> Table																																				
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%;">Field Name</th> <th style="width: 50%;">Field Type</th> </tr> </thead> <tbody> <tr><td>Sn</td><td>A4</td></tr> <tr><td>Last Name</td><td>A15</td></tr> <tr><td>First Name</td><td>A15</td></tr> <tr><td>Address</td><td>A20</td></tr> <tr><td>City</td><td>A14</td></tr> <tr><td>State</td><td>A2</td></tr> <tr><td>Zip</td><td>A5</td></tr> <tr><td>Areacode</td><td>A3</td></tr> <tr><td>Phone</td><td>A8</td></tr> <tr><td>Date Enrolled</td><td>D</td></tr> </tbody> </table>	Field Name	Field Type	Sn	A4	Last Name	A15	First Name	A15	Address	A20	City	A14	State	A2	Zip	A5	Areacode	A3	Phone	A8	Date Enrolled	D	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%;">Field Name</th> <th style="width: 50%;">Field Type</th> </tr> </thead> <tbody> <tr><td>Sn</td><td>A4*</td></tr> <tr><td>Department</td><td>A12*</td></tr> <tr><td>Level</td><td>A3*</td></tr> <tr><td>Term</td><td>A1*</td></tr> <tr><td>Year</td><td>A2*</td></tr> <tr><td>Grade</td><td>A1</td></tr> </tbody> </table>	Field Name	Field Type	Sn	A4*	Department	A12*	Level	A3*	Term	A1*	Year	A2*	Grade	A1
Field Name	Field Type																																				
Sn	A4																																				
Last Name	A15																																				
First Name	A15																																				
Address	A20																																				
City	A14																																				
State	A2																																				
Zip	A5																																				
Areacode	A3																																				
Phone	A8																																				
Date Enrolled	D																																				
Field Name	Field Type																																				
Sn	A4*																																				
Department	A12*																																				
Level	A3*																																				
Term	A1*																																				
Year	A2*																																				
Grade	A1																																				

The *Students* table contains the student number (field *Sn*) and other biographical information about each student. The *Classes* table contains the student number again, along with information about each class the students have taken. Over time, the *Classes* table will grow to many times the size of the *Students* table.

We'd like to create a form that will show a student's biographical information, along with the classes they've taken. The end result should look something like this:

Viewing *Students* table with form F2: Record 2 of 10

Main ▲=▼

Computer University				
Student Number:	1001	Date Enrolled:	1/05/83	
Last Name, First Name:	Wilson	Steve		
Address:	8653 Rita Drive			
City, State, Zip:	Bloomington	KS	47401	
Area Code and Telephone:	316 343-2911			
Department	Level	Term	Year	Grade
Biology	101	F	85	A
Chemistry	101	S	85	A
Chemistry	110	U	86	B
English	101	F	83	C
English	110	S	84	A
History	110	F	84	A
History	200	S	85	B
History	220	F	86	D
Math	110	S	85	B

At the top of the form, you see the information from the *Students* table. The *Classes* table appears at the bottom of the table. There are several steps we need to take to create this form. The first step is to create a multi-record form for the *Classes* table. Then, we need to embed this form in a form for the *Students* table.

## Creating a Multi-Record Form

A multi-record form shows more than a single record at a time. The steps to follow when creating this type of form is to create the fields for the first or original record, and then to duplicate these fields as many times as you need.

Start by creating form 1 for the *Classes* table. The commands are *Form Design Classes 1*. Use *For inserting in Students table* as the description.

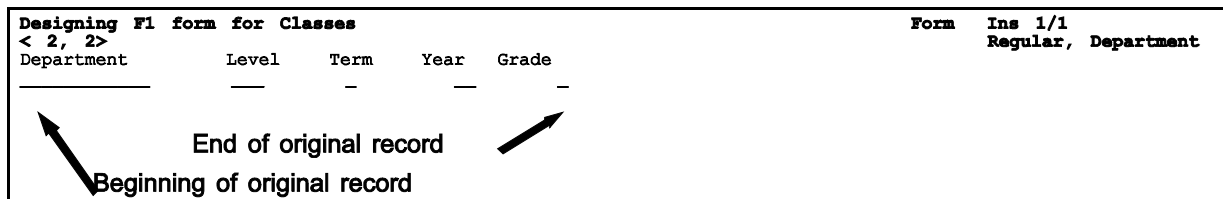
Now, create a table that looks like this:

&lt; 2, 2&gt;

Regular, Department

Department	Level	Term	Year	Grade
_____	___	-	___	-

Start designing the form at the very top of the screen. A good first step is to move to the second line of the form and use the *Field Place* command to start placing the fields. You can leave plenty of space between the fields. Then, move to the first line of the form and type the field titles.



The beginning and ending parts of the original record for a multi-record form.

After designing the original record, we're ready to define the multi-record region. The steps are as follows: First, issue the *Multi Records Define* command. Move to one corner of the original record and press the Enter key (the correct location is at the very beginning of the *Department* field). Then, move to the other corner of the original record and press the Enter key again (the correct location is the very end of the *Grade* field). Now, use the Down arrow key to duplicate the original record as many times as you need. We'll need about eight copies, for a total of nine records, counting the original record. When finished, your form should look something like this:

&lt;10,41&gt;

Department	Level	Term	Year	Grade
_____	___	-	___	-
_____	___	-	___	-
_____	___	-	___	-
_____	___	-	___	-
_____	___	-	___	-
_____	___	-	___	-
_____	___	-	___	-
_____	___	-	___	-
_____	___	-	___	-

Complete the form design by pressing *F2 Do\_It!*. To see how a multi-record form works, view the *Classes* table and use the *Image Pickform* command to select form 1. You'll see a number of records. If you use the Down arrow key to move to the last displayed record and then move down even further, Paradox will bring additional records on the form for viewing.

Using a multi-record form is in many ways similar to viewing records in table view. Some of the differences include:

- The original record, and therefore the copies of the original record, can occupy more than a single line on the form.
- You can't scroll horizontally off the screen in the form as you can in a table.

- You can place calculated fields on a multi-record form. A table can't have or display calculated fields.

---

## Copying a Form

---

The next step in creating a multi-table form is to create the form for the *master* table, which is the *Students* table in this example. The form we need is nearly identical to form number 1 that we've already created. By copying it, we can create a new form number 2 that is just like the existing form number 1. Follow these steps:

- 1) Issue the *Tools Copy Form* command.
- 2) We're copying within the same table, so select the *SameTable* command. You can, as you see, copy a form to a different table if the two tables have a similar structure.
- 3) Select the table that you want to work with, which is *Students*. You can type the table name or press the Enter key to get a list of tables.
- 4) Choose the form that you want to make a copy of, which is form 1.
- 5) Choose the form number that you'd like to copy to. We'll use form number 2 for this example.

After copying, form 2 is identical to form 1. We can now change form 2 to meet our needs. This copying is a good tool to remember, as many times when you need to create a new form, the new form is similar to an existing form. You can copy reports in the same way by using the *Tools Copy Report* command.

---

## Placing an Embedded Detail Table

---

For the tables that we're working with, the *Students* table is known as the *master* table, while the *Classes* table is known as the *detail* table. In other words, *Classes* contains the detail about the enrollments for records in the *Students* table. We want to set up a *linked* set of tables, so that as we move through the records in the *Students* table, Paradox displays the matching records from the *Classes* table. You can have two tables on the same form and have them not be linked, but linking the two tables is by far the most common and useful thing to do.

When setting up a form with two linked tables, there are some restrictions that you must follow:

- The detail table (*Classes* in this example) must have a key field, and the key field must be the field used to link the two tables together. In this case, the linking field is *Sn*, so *Classes* must be keyed on *Sn*. The key field or fields used to establish the linkage to the master table can not be placed on the embedded form. This is why we didn't place the *Sn* field on the form.

## Paradox Level 2

---

- The detail table can have other key fields besides the ones used to establish the linkage to the master table, as our *Classes* table does. If there are these not-linking key fields, you must place them on the detail form that will be embedded in the master table form.
- The key field or fields used to establish the linkage must be the first field or fields in the detail table. In the *Classes* table, then, this means that the *Sn* field must be the first field in the table.

After making sure our example meets these restrictions, we're ready to place the detail table from the *Classes* table into form2 for the *Students* table. Follow these steps:

- 1) Issue the *Form Change* command and identify *Students* as the table. Select form 2 as the form to change.
- 2) Change the form's description to something more meaningful, such as *Students and their classes*. Remember that *Control-Backspace* deletes the entire contents of a field with a single keystroke.
- 3) Issue the *Multi Tables* command to start placing an embedded detail table in this form.
- 4) Issue the *Place* command. You can see that you can also remove or move an embedded table from this menu.
- 5) Because the detail table *Classes* is linked to the master table, issue the *Linked* command.
- 6) Select the name of the detail table, which is *Classes*.
- 7) Pick the form from the *Classes* table to embed, which is form number 1.
- 8) Now, we need to identify the field from the master table (*Students*) to match the key field from the *Classes* detail table that isn't placed on the form. The menu Paradox presents looks like this:

```
Select STUDENTS field to match Sn in CLASSES.                Form      1/1
Sn Last Name First Name Address City State Zip Areacode Phone >
```

The field we need is *Sn*, so choose it from the menu.

- 9) Using the arrow keys, place the image of the embedded form in the proper place. Your form design should look like this:



## Validity Checks

---

Validity checks are an important feature of Paradox that let you exert control over the data being entered or edited in a table. Some of the things that validity checks can do include establishing minimum and maximum values for a field, setting a default value for a field, performing a lookup into a different table, setting a template to automatically format data, and requiring that a field not be left blank.

To define validity checks, you must be either editing, coediting, or entering data to the table. Then, you can press the *F10 Menu* key and issue the *ValCheck* command. After this command, you can choose to define validity checks, or clear ones that are already set. When you choose the *Define* command, you'll be prompted to move to the field that you want to define a check for, and then to press the Enter key. You can then choose the type of validity check to define.

## Minimum and Maximum Values

---

The *LowValue* and *HighValue* commands let you determine the lowest and highest values you can enter to the field. This is most commonly used for a numeric field. For example, you might specify that the lowest value to be entered for the quantity ordered as one. This will prevent the accidental entry of zero or a negative number.

For date fields, you can enter the word *today*. If entered for the *LowValue* command, you won't be able to enter dates in the past. If entered for the *HighValue* command, you won't be able to enter dates in the future.

## Default Values

---

If you enter a default value for a field, Paradox will automatically enter it when you move the cursor out of the field. Use a default value when the field contains a high proportion of a certain value. For example, if most of your customers are in Kansas, you might enter *KS* as the default value for the *State* field. For date files, the word *today* automatically enters the current date.

In all cases, if the default value isn't the right value for a particular record, you can enter a different value, or delete the default value that Paradox entered and enter some other value.

## Required Values

---

The *Required* command, when used on a field, means that the field can't be left blank. If you try to move the cursor out of this field, Paradox will remind you that the field can't be left blank and will not move out of the field. Use this sparingly, as there are few fields that you'll always know the value of as you perform data entry.

## Table Lookups

A table lookup is a powerful feature. One of the things it can do is require that values entered into a field match values entered in a different table. For example, suppose you'd like to require that the two-letter state abbreviations being entered into a table are correct, so that someone doesn't enter *KA* as the abbreviation for Kansas.

The first step in accomplishing this goal is to create a table of the state name abbreviations. Two important restrictions exist:

- The table of state names can have many fields, but the state names (the field you want to look up in) must be the first field of the table.
- The lookup table should be keyed on the first field. This is not an absolute requirement, but for good performance, the table should be keyed. If it is keyed, only the first field can be the key. You couldn't use a key composed of the first two fields. An example of this type of table looks like this:

STABBREV	Abbr	State
1	AK	Alaska
2	AL	Alabama
3	AR	Arkansas
4	AZ	Arizona
5	CA	California
6	CO	Colorado
7	CT	Connecticut
8	DE	Delaware
9	FL	Florida
10	GA	Georgia

After creating the lookup table, follow these steps:

- 1) Edit the table that you want to implement the lookup from.
- 2) Issue the *ValCheck Define* command. Move to the *State* field and press the Enter key.
- 3) Issue the *TableLookup* command.
- 4) Enter the name of the lookup table (the table of state names). You can press the Enter key to get a list of tables.
- 5) Choose *JustCurrentField*.
- 6) Choose *HelpandFill*.

The sixth step deserves explanation. Selecting *HelpandFill* means that during data entry or editing, you can look at the table of state names to help choose the right name. As you move to this field, Paradox displays a message at the top of the screen indicating that pressing *F1 Help* provides help with fill-in. After pressing this key, you can see the lookup table. After moving the

cursor to the record of interest, press *F2* to enter this value in the data entry table.

Setting a lookup using the *HelpandFill* option does not mean that you must press *F1 Help* to select values from the lookup table when you enter data. You can type values directly into the field. But if the value you enter isn't in the lookup table, you'll see a message indicating this fact. You can then press *F1 Help* if you want and select a value from the lookup table.

The other choice besides *HelpandFill* is *PrivateLookup*. In this case, if you enter a value that's not in the lookup table, Paradox displays a message, and you'll have to try again. There is no way to view the lookup table for help in selecting a correct value.

When using table lookup, make sure that the lookup table contains all values that you'll need to enter to the entry table. If during data entry you find that you legitimately need to enter a value that's not in the lookup table, you'll have to exit the data entry, add the value to the lookup table, and then resume data entry again.

## Pictures

Pictures can restrict the type of data entered into a field, and can help make sure the data fits a defining template.

The key to creating pictures is to enter *pattern elements* and *constants* that control the data being entered. Here's a table of pattern elements and their meanings:

Pattern Element	Meaning
#	Accept only a numeric digit. You won't be able to enter an alphabet letter.
?	Accept only an alphabet letter, either upper case or lower case.
&	Accept only alphabet letters, and convert them to upper case if entered in lower case.
@	Accept any single character.
!	Accept any character, and convert lower case alphabet letters to uppercase.
;	Take the next character literally. Use this when you want to use one of the special pattern elements as a constant.
*	Defines a repetition count for repeating a pattern element a certain number of times. For example, *10# means the same thing as #####. Not including a number for the repetition count means to accept zero or more repetitions of the following pattern element.
[]	Defines part of the picture as an optional element.
{ }	Defines a group of operators.
,	Defines a set or list of alternatives.

Pattern Element	Meaning
Any other character	Establishes a constant character that will be entered into the field.

Here's some examples of the pictures in use:

Picture	Meaning
###-####	Accept three numeric digits, then enter a hyphen, and accept four more numeric digits. This defines a telephone number, not taking into account the area code.
(###) ###-####	Enter a parenthesis, accept three numeric characters, then enter a parenthesis, then accept a telephone number as above. This defines a telephone number with the area code.
[(###) ]###-####	Defines a telephone number with an optional area code in parenthesis. If the first character entered is (, the Paradox assumes it's part of the area code and Paradox will require entry of the rest of the area code. If the first character entered is a numeric digit, Paradox will assume it's the first part of the local part of phone number, since it doesn't match the optional part of the picture. Note that entering a space character matches the first constant pattern value in a picture.
True,False	Establishes either <i>True</i> or <i>False</i> as the only two allowable values for this field.
Red,Green,Blue	Establishes these three colors as the only allowable values for this field.
&&	Accept two alphabet letters and convert them to uppercase. This is useful for the two-letter state abbreviations, which should be upper case. With this picture, you must enter two letters. You won't be able to enter just one letter.
&*?	Enter a word of any length, but capitalize the initial letter. The & means to capitalize an entered letter. The ? means to accept an letter in upper or lower case. The * before the ? means to repeat the ? pattern element any number of times.
&. &. &*?	Two capitalized initials followed by space characters, followed by a capitalized word of any length.
##[#]	A number with two or three digits. The third digit is optional because of the [] characters.
#####[-#####]	A five- or nine-digit zip code. The - and the four # characters are optional because of the [] characters.
{40,60,77,100}W	Forces entry of either 40, 60, 75, or 100, and follows it with W, such

Paradox Level 2

Picture	Meaning
	as the common light bulb wattages. The { } characters make a group out of the <i>40,60,75,100</i> list.

## More About Queries

Queries, of course, are one of the most important skills to develop when using Paradox. This section describes how to perform some useful actions with Paradox queries.

### Query Words and Operators

Operator	Example	Explanation
√	√	The check, produced (and removed) by pressing <i>F6 Check</i> . Includes the checked field in the <i>Answer</i> table, eliminates duplicate values from the checked fields, and produces the records sorted in ascending order. If the query contains summarization operators such as <i>calc sum all</i> , then the checks produce groupings of the summary by the checked field or fields.
√+	√+	The Check Plus, produced by <i>Alt-F6 Check Plus</i> . Includes the checked field in the <i>Answer</i> table, but doesn't eliminate duplicate values from the checked fields, and doesn't produce the records in sorted order.
√∇	√∇	The Check Descending, produced by <i>Control-F6 Check Descending</i> . Includes the checked field in the <i>Answer</i> table, eliminates duplicate values from the checked fields, and produces the records sorted in descending order.
A word, date, or number	Wichita 100000 3/25/91	Tells Paradox to select records where the field contains the value specified. With alphanumeric fields, Paradox does distinguish between uppercase and lowercase letters. Thus, <i>Bob</i> does not match <i>BOB</i> . Don't type commas in numeric conditions, as the comma has a special meaning when used in a query form. Also, for date conditions, it's not necessary to pad the date with leading zeros.
like	like Wichutaw like ks	Finds data in a table by a close spelling or phonetically. Also disregards punctuation differences.
blank	blank	Finds fields that are totally empty.
not	not Wichita not blank	Reverses the meaning of a condition. <i>Not blank</i> finds records where the field isn't blank, while <i>not Wichita</i> finds everyone who doesn't live in Wichita.
..	bob.. ..bob ..bob..	The pattern operator, which matches any number of any characters. <i>bob..</i> matches anything beginning with <i>Bob</i> , such as <i>Bob</i> , <i>Bobby</i> , and <i>Bob's</i> . <i>..bob</i> matches anything ending with <i>bob</i> , such as <i>Bob</i> and <i>Jim Bob</i> . <i>..bob..</i>

Operator	Example	Explanation
		matches anything containing <i>bob</i> , including all the above examples and words like <i>carbобензоxy</i> . When using <i>..</i> , Paradox disregards differences between uppercase and lowercase letters.
@	b@b	The single character pattern matcher, which matches any single character. This example matches <i>bab</i> , <i>bbb</i> , <i>bcb</i> , and so forth. As with the <i>..</i> pattern, capitalization doesn't matter.
"	"Doe, John"	Since the comma has a special meaning in a query form, if we need to search for a value containing a comma, we must enclose the search condition in quotation marks. This example searches for a field with the value <i>Doe, John</i> .
= > >= < <=	=1000 >1000 >=1000 <1000 <=1000	The range operators. The equal sign is optional.
,	>=1000, <2000	Links two ranges operators together so that records must match both conditions. This example means the records that are greater than or equal to 1000 and less than 2000, which are the values from 1000 up to 2000.
,	<u>a</u> , >1000	In this example, the comma is used to separate the example element <u>a</u> from the condition <i>&gt;1000</i> .
or	KS or OK	Links to conditions with the logical operator <i>or</i> . As long as the value matches either condition, Paradox will find it. This query asks to include those records where the state is Kansas or Oklahoma, so both states' residents will be included.
today	today <today	In date fields, serves as a shorthand notation for today's date.
Example element	<u>bob</u>	On the screen, example elements show in reverse, while in this booklet, they show in double underline. Example elements generally serve to link two fields or table together, or serve to supply a name for a field for further use.
calc	calc sum calc average calc count calc max calc min	The summary operators in Paradox. When typed in a field, calculates the summary in the <i>Answer</i> table. When using <i>calc sum</i> , in a field called <i>Sales</i> , for example, the <i>Answer</i> table will contain a field called <i>Sum of sales</i> . If the query form contains checked fields, these summaries will be formed for each row of the table.

Operator	Example	Explanation
all	calc count all	Tells Paradox to summarize all records, even duplicates.
unique	calc count unique	Tells Paradox to not include duplicate records in the summary calculation.
as	calc sum as total	Lets you rename the summary field in the <i>Answer</i> table. Instead of a field called something like <i>Sum of sales</i> , the field will be called <i>Total</i> .
A formula	calc <u>pop</u> / <u>area</u>	Tells Paradox to perform the indicated calculation. The words <u>pop</u> and <u>area</u> are example elements that have been defined somewhere else in the query. Use + for addition, - for subtraction, * for multiplication, / for division, and () to group expressions for a special calculation order.
delete	delete	Use to perform a query to delete records.
find	find	Used to perform a query that finds records.
insert	insert	Used to perform a query that insert records into a table.
changeto	changeto	Used to change values in a table.

## Finding Records

One thing a query can do is to find records that match a condition. By finding the record, we mean that Paradox moves the cursor to the first record that matches a condition. The way to use this query is to type the word *find* in the leftmost field of the query form, under the table name. Then, enter the record match conditions and press *F2 Do\_It!*. Here's an example:

```

PEOPLE-----First Name-----Last Name-----City-----State-----
find          |                   |                   | New York | NY
  
```

This will move the cursor through the *People* table to the first record where the *City* field is *New York* and the *State* field is *NY*.

Some things to remember about find queries:

- The cursor moves to the first record that matches the condition. There may be other records that match, but the cursor moves only to the first matching record.
- Find queries can't have checked fields.
- A find query does create an *Answer* table. Paradox doesn't automatically display it, but you can use the *View* command to view it. This table will contain all matching records. The

records in this *Answer* will be in table order, rather than in a sorted order like records in an *Answer* table usually are.

- What is the difference between using a find query and the *Zoom* (Control-z) and *ZoomNext* (Alternate-z) keys? *Zooming* can look for a record based on the contents of just a single field. You wouldn't be able to enter conditions in two fields, as in the above example. On the other hand, the *ZoomNext* key can look for additional matches besides the first match. A find query can't do that.

## Deleting Records

---

A query can also delete records that meet a certain criteria. For example, suppose your company decides to cease doing business in a state. You might want to delete the customer names from that state as follows:

PEOPLE	First Name	Last Name	State	Zip
delete			NY	

To delete records with a query, type the word *delete* in the leftmost column of the query form, under the table name. Then, type the condition that matches the records that you want to delete. In this example, we're deleting records where the *State* field is *NY*. Don't check any fields in a delete query.

When deleting records, these things happen:

- Paradox removes the matching records from the table being queried, which is *People* in this example.
- Paradox creates a new table called *Deleted* that contains the deleted records. This table serves as a backup—if you delete too many or the wrong records, you can add them from the *Deleted* table to the original table (*People* in this example) with the *Tools More Add* command. *Deleted* is a Paradox temporary table. This means that if you perform another delete query, exit Paradox, or change the default directory, the *Deleted* table disappears.

## Inserting Records

---

Inserting records with a query is much like using the *Tools More Add* command. When using *Tools More Add*, however, the source table and the target table must have a nearly identical structure. If the two tables don't have a similar enough structure, you can't use this command.

Inserting records uses example elements to help Paradox insert data from one table to another, along with the word *insert* in the table that is to accept the records (the target table). Here's an example:

FRIENDS	First Name	Last Name	City	State
	<u>fn</u>	<u>ln</u>	<u>city</u>	<u>state</u>

PEOPLE	First Name	Last Name	City	Stat
<b>insert</b>	<u>fn</u>	<u>ln</u>	<u>city</u>	<u>state</u>

What these query forms don't show is that the *Friends* table contains only the four fields shown in the query form, while the *People* table contains many more fields. Therefore, the *Tools More Add* command won't work in this example.

In this example, the *People* table is the target table, due to the word *insert* entered under the table name in the query form. The example elements fn, ln, city, and state (double underlines in this booklet mean the word is an example element) serve as the linkage, telling Paradox how to match up the fields. Remember, you'll need to use the *Ask* command twice, once for each of the two table participating in this query. Press *F5 Example* before typing an example element. It doesn't matter what word you use for the example elements, as long as you use them consistently, and in this case, use a different example element word for each of the field pairs to match.

When you perform an insert query, keep these things in mind:

- Don't check any fields in an insert query.
- Check your example elements carefully to make sure they match the correct fields. You'll notice that in this example, the field names of the matching fields in the two tables is the same. This is not sufficient for Paradox to match the fields; you must still use the example elements.
- After pressing *F2 Do-It!* to perform the query, Paradox adds the records to the target table. Paradox also creates a temporary table called *Inserted* that contains the inserted records. You can review this table to see how the insert process went. Since this is a Paradox temporary table, it will disappear when you leave Paradox, change the default directory, or perform another insert query.
- The records you insert will appear at the end of the target table, unless the target table has a key field or fields. In this case, the inserted records will appear in the target table in key field order.
- If the target table has a key field or fields, there is the possibility of a key conflict. Records incoming from the source table will be placed in the *Keyviol* table if they conflict with existing records in the target table.

## Changing Data in Tables

A query can also change data in a table by using a *changeto* query. For each field you'd like to change, use the word *changeto* followed by an expression or value. Paradox makes the change to the original table, and also creates a temporary table called *Changed*. This table contains an original copy of all the data that was changed, so that if you're not satisfied with the results, you have a backup.

Here's a simple example:

```
PEOPLE-----Credit Limit-----
      |      |
      |      | changeto 8000 |
      |      |      |
```

This query changes the *Credit Limit* field for all records in the table to \$8,000. Another example:

```
PEOPLE-----State-----Credit Limit-----
      |      |      |      |
      |      |      | changeto 8000 |
      |      |      |      |
```

Because of the condition *CA* in the *State* field, only those records that match the condition will change.

```
PEOPLE-----Credit Limit-----
      |      |
      |      | >=10000, changeto 8000 |
      |      |      |
```

In this example, the *Credit Limit* field first contains a condition,  $\geq 10000$ . This serves to limit the changes to just those records that meet the condition. To separate the condition from the *changeto* word, use a comma. This query, then, finds those records with a *Credit Limit* of \$10,000 or greater, and changes the *Credit Limit* to \$8,000.

This query will increase all *Credit Limit* fields by ten percent:

```
PEOPLE-----Credit Limit-----
      |      |
      |      | √ a, changeto a * 1.10 |
      |      |      |
```

The example element a serves as a label for the *Credit Limit* values so that we can refer to them in the formula after the *changeto* operator.

This query changes the *Credit Limit* values for all the records. You could, of course, narrow the action of the query to just people in California by writing *CA* under the *State* field, for example. Or to increase the *Credit Limit* just for those who already have a *Credit Limit* value of \$5,000 or greater, use this query:



## Joining Tables With Example Elements

---

Paradox queries contain a powerful tool called *examples* that let you link together several table in a single query, or even link different rows of the same table. To start the study of examples, let's look at the *Students* and *Classes* tables.

We're already familiar with the *Students* and *Classes* tables from creating forms with them. You can see that the *Students* table contains the biographical or personal information about each student—name, address, and so forth. The *Classes* table contains information about classes taken by all students. Each student is identified by a student number (the field *Sn*), and so too each class in the *Classes* table is associated with a student number.

Our question is to develop a list of all classes that William Smith has taken. No single table can provide the answer, as no table contains information about classes and also student names. But combining the *Students* and *Classes* tables together can provide the answer, as in this query:

STUDENTS	Sn	Last Name	First Name	Address
	<u><u>a</u></u>	Smith	William	

CLASSES	Sn	Department	Level	Term
	<u><u>a</u></u>	√	√	

There are two query forms in this query, one for the *Students* table and one for *Classes*, requiring two separate *Ask* commands. We can already understand two of the condition in the *Students* query form that select the record for William Smith. The third part of the query, the a beneath the *Sn* (student number) field, is an example. Example elements serve to link one table with another. Examples can be anything you want—the use of the word a is an arbitrary choice. To enter an example element, you first press *F5 Example* and then type the example. On the computer screen, example elements appear in reverse or highlighted video or in a different color, while in printed or stored queries, they're preceded by an underscore character. In the examples in this booklet, example elements appear in double underline.

The query form for the *Classes* table contains the same example element a beneath its *Sn* field, and checks for the *Department* and *Level* fields.

The example element a serves to link the two tables together. The example stands for whatever value is selected from the table. In this case, the example a stands for William Smith's student number. In effect, this query says *Find the student number for William Smith and store it as a. Then, move to the Classes table and find all classes where the associated student number is the value stored in a.*

The *Answer* table for this query will contain just two fields, the *Department* and *Level* fields from the *Classes* table, because those are the only fields checked. Fields used in an example do not necessarily need to be checked.

---

A second example query: Suppose you want to find the names of all students who have taken Chemistry 101. These query forms provide the answer:

STUDENTS	Sn	Last Name	First Name	Address
	<u>bob</u>	√	√	

CLASSES	Sn	Department	Level	Term
	<u>bob</u>	Chemistry	101	

In this query, the conditions in the *Classes* table find those records where the *Class* field contains Chemistry 101. The example bob links the student numbers together, so that each time Paradox selects a record from the *Classes* table, it finds all records from the *Students* table with matching student numbers.

### Including All Records in a Join

When you perform a query that joins records from two tables, a record appears in the *Answer* table only when a match is found between the two tables. Look at the following example:

STUDENTS	Sn	Last Name	First Name	Address
	√ <u>sn</u>	√	√	

CLASSES	Sn	Year	Grade	Department
	<u>sn</u>			

ANSWER	Sn	Last Name	First Name
1	1000	Smith	William
2	1001	Wilson	Steve
3	1002	Butler	Susan
4	1003	Jones	Sarah
5	1004	Smith	John
6	1005	Jackson	William
7	1006	Brown	Arthur
8	1007	Connors	Joseph
9	1008	Johnson	Chester
10	1009	Smith	Albert

Since each student in the *Students* table has taken at least one class, all students appear in the *Answer* table. In the next example, we place the condition 86 in the *Classes* table, in effect asking for those students who took a class in 1986:

STUDENTS	Sn	Last Name	First Name	Address
	√ <u>sn</u>	√	√	

CLASSES	Sn	Year	Grade	Department
	<u>sn</u>	86		

ANSWER	Sn	Last Name	First Name
1	1000	Smith	William
2	1001	Wilson	Steve
3	1003	Jones	Sarah
4	1004	Smith	John
5	1005	Jackson	William
6	1006	Brown	Arthur
7	1007	Connors	Joseph
8	1008	Johnson	Chester
9	1009	Smith	Albert

Student number 1002 is missing because that student didn't take any classes in 1986. But look at this example:

STUDENTS	Sn	Last Name	First Name	Address
	√ <u>sn</u> !	√	√	

CLASSES	Sn	Year	Grade	Department
	<u>sn</u>	86		

ANSWER	Sn	Last Name	First Name
1	1000	Smith	William
2	1001	Wilson	Steve
3	1002	Butler	Susan
4	1003	Jones	Sarah
5	1004	Smith	John
6	1005	Jackson	William
7	1006	Brown	Arthur
8	1007	Connors	Joseph
9	1008	Johnson	Chester
10	1009	Smith	Albert

Student number 1002 is back. The exclamation point after the example element in the *Students* table is responsible for the inclusion of 1002 in this *Answer* table. The exclamation point tells Paradox to perform an *outer join*, which means to include all records from the table where the !

---

appears, even if there are no matching records in the other table.

As another example, consider a *Customer* table that contains customer information, and an *Orders* table that contains details about the orders each customer has placed. The tables are linked by a *Customer number* field. You'd like to create a table of customer and the orders they've place. By linking the tables with example elements in the *Customer number* field, you're telling Paradox to include only those customers who have placed an order. But if you place an exclamation point after the example element in the *Customers* table, Paradox will include all customers in the *Answer* table—even those who haven't yet placed an order.

## More About Reports

---

Reporting is an important topic in Paradox, because the way to print data in Paradox is through a report.

## Lookups In Reports

---

An important feature of a report is the ability to lookup data in other tables. Our goal here is to produce student transcripts. An example of the report output looks like this:

2/06/91

Student Transcripts

Page 1

Arthur                    Brown                    Student Number: 1006  
8897 Plaines Lane  
Cheyenne, WY 82005

Department	Level	Term	Year	Grade
-----	-----	----	----	-----
English	101	F	83	C
History	100	S	83	D
Physical Ed.	101	U	83	A
Physical Ed.	110	F	83	B
Psychology	101	S	83	C
Sociology	101	S	83	F
History	110	S	84	C
Math	101	F	84	B
Music	101	S	84	D
Physics	101	S	84	D
Psychology	110	F	84	A
Sociology	110	F	84	D
Chemistry	101	S	85	C
History	200	F	85	B
Music	110	F	85	C
Biology	101	S	86	C
Chemistry	110	F	86	C
History	220	U	86	C
Math	200	S	86	A
Math	220	F	86	C
Music	200	S	86	I
Physics	110	F	86	D

2/06/91

Student Transcripts

Page 2

Susan                    Butler                    Student Number: 1002

---

4902 Bluffside Road  
Muskegon, MI 49450

Department	Level	Term	Year	Grade
English	101	U	83	B
History	100	S	83	D
Physical Ed.	101	S	83	A
Physical Ed.	110	F	83	I
Psychology	101	F	83	C
English	110	F	84	A
History	110	F	84	D
Psychology	110	S	84	D
Sociology	110	F	84	I
Chemistry	101	F	85	A
History	200	S	85	B
Math	110	S	85	B
Music	110	F	85	C

To start designing the report, issue the *Report Design* command. Select *Classes* as the table, and report number 1 as the report to design. Give a description such as *Student Transcripts*. Select *Tabular* as the type of report.

Next, remove the *Sn* field from the table command. The way to do this is to issue the *TableBand Erase* command, move to the table band in the column that you want to remove, and press the Enter key. Your report design should look something like this when finished:

```

Designing report R1 for Classes table                                Report 1/1
Table Band                                                         Department
....+...10....+...20....+...30....+...40....+...50....+...60....+...70....+...8*
  
```

—▼page—

mm/dd/yy Student Transcripts Page 999

┌▼table

Department	Level	Term	Year	Grade
AAAAAAAAAAAA	AAA	A	AA	A

└▲table

—▲page—



example) must be keyed on the lookup expression. This means that the *Students* table must be keyed on *Sn*.

- You can establish lookups into more than one other table.

To establish the link, issue the *Field Lookup Link* command. Select the name of the table that you want to look into. Next, identify the common field between the tables.

```
Select CLASSES field to match Sn in STUDENTS.
Sn Department Level Term Year Grade
```

Paradox wants to know which field from the *Classes* table (the table this report is for) matches the *Sn* field in the *Students* table. How did Paradox know that we wanted to match with the *Sn* field in the *Students* table? Because it's the only key field in that table. Select the *Sn* field.

After completing this command, you won't see anything different. You will, however, see something like this when you perform the *Field Place Regular* or *Group Insert Field* commands:

```
Field to place
Sn Department Level Term Year Grade [Students->]
```

The entry [**Students->**], when selected, displays a list of fields from the *Students* table. You can place these fields or create groups on them as you would any other field.

## Placing Fields

Now, place the student name and address fields in the group header for *Sn*. On the first row, place the *First Name* and *Last Name* fields. (You'll need to use the *Field Place Regular* command and select [**Students->**].) Type a label and place the *Sn* field. Then place the *Address* field on the next line.

For the city, state, and zip code, you could place them as you did the *First name* and *Last name* fields. Doing this, however, will lead to lines that look like this:

```
Wichita           KS 67207
Kansas City       MO 60610
```

A better approach is to use a calculated field. Normally, these fields are used for a mathematical calculation, but you can also concatenate alphanumeric fields and constants. Issue the *Field Place Calculated* command and enter this for the formula:

```
[Students->City]+", "+[Students->State]+" "+[Students->Zip]
```

Note that field names are enclosed in the [] characters. If the fields we wanted to use existed in the table the report is being designed for, we could simply type `[City]+", "+[State]+" "+[Zip]`. Since these fields aren't in the table the report is being designed for, but are in the *Students* table instead, we need to precede the field names with the name of the table they're found in, a minus sign, and a greater than sign. Notice that constants are surrounded by quotation marks, and the

character strings are joined by plus signs.

When finished, your report design should look like this:

Changing report R1 for Classes table Report Ins 1/1  
 Page Header  
 ....+...10....+...20....+...30....+...40....+...50....+...60....+...70....+...8\*  
 -▽page

mm/dd/yy Student Transcripts Page 999

—▽group Sn—  
 AAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAA Student Number: AAAA  
 AAAAAAAAAAAAAAAAAA  
 AAAAAAAAAAAAAAAAAA

┌▽table

Department	Level	Term	Year	Grade
-----	----	----	----	----
AAAAAAAAAAAA	AAA	A	AA	A

└▲table

—▲group Sn—

## Report Formatting

In order to send these transcripts, we'd like to print each student's transcript on a separate sheet of paper. The special word *PAGEBREAK* can do this. This word, for it to have its intended meaning, must be type in all capital letters, and must start at the very left margin of the report design. When Paradox comes to this word, it will start a new page.

The best place to put this word in our report is in the group footer for *Sn*.

Another type of formatting concerns when Paradox prints the heading rows from the table band. The command is *Settings Format*, and the default setting is *TableOfGroups*, which means that Paradox prints these lines at the top of each page. The other setting, which is *GroupsOfTables*, means that Paradox prints the heading lines after each group header.

This example was printed with the default setting of *TableOfGroups*.

3/06/91

Student Transcripts

Page 1

Department	Level	Term	Year	Grade
-----	-----	-----	-----	-----
William	Smith			Student Number: 1000
78 Rye Street				
Topeka, KS 66699				

Biology	101	S	86	C
Chemistry	101	U	85	C
English	101	F	83	B
English	110	F	84	B

This example was printed after changing to *GroupsOfTables*.

3/06/91

Student Transcripts

Page 1

William	Smith			Student Number: 1000
78 Rye Street				
Topeka, KS 66699				

Department	Level	Term	Year	Grade
-----	-----	-----	-----	-----
Biology	101	S	86	C
Chemistry	101	U	85	C
English	101	F	83	B
English	110	F	84	B

## Regrouping

---

Currently, these transcripts print in student number order because of the grouping on the *Sn* field. Suppose you decide you'd like to print the transcripts in alphabetical order by the students' last names. You can change the field that a group is based on by issuing the *Group Regroup* command. Select *Field* again, and select the *Last Name* field from the *Students* table. The report design should look like this when finished:

Paradox Level 2

---

Changing report R1 for Classes table Report 1/1  
Group Header for [Students->Last Name] [Students->First Name]  
.....10.....20.....30.....40.....50.....60.....70.....8\*  
-vpage

---

mm/dd/yy Student Transcripts Page 999

—vgroup [Students->Last Name]—  
AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAA Student Number: AAAA  
AAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAA

r^vtable

Department	Level	Term	Year	Grade
-----	-----	-----	-----	-----
AAAAAAAAAAAA	AAA	A	AA	A

L^table  
PAGEBREAK  
—^group [Students->Last Name]—

---

### Adding Additional Groups

---

We might want to add a second group to the report so that the classes print in year order, with a blank line between years. To do this, issue the *Group Insert Field* command and identify *Year* as the field to group on. When Paradox asks where to insert the group, make sure the cursor is below the *Last Name* group. Then, delete the black line from the group header for *Year*. (Use Control-Y at the beginning of a line to delete a line.) Your design should look like this when finished:

Changing report R1 for Classes table

Report Ins 1/1

Page Header

....+...10....+...20....+...30....+...40....+...50....+...60....+...70....+...8\*

-vpage

mm/dd/yy

Student Transcripts

Page 999

—vgroup [Students->Last Name]

AAAAAAAAAAAAAAAAAAAAAAAAAAAA Student Number: AAAA  
 AAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
 AAAAAAAAAAAAAAAAAAAAAAAAAAAAA

—vgroup Year

rtable

Department	Level	Term	Year	Grade
AAAAAAAAAAAA	AAA	A	AA	A

L^table

—^group Year

PAGEBREAK

It's important that the group for *Year* appear within the *Last Name* group so that the data is sorted first by *Last Name*, and then by *Year*.

The output from this design, unfortunately, looks like this:

History	110	S	84	C
Math	101	F	84	B
Music	101	S	84	D
Physics	101	S	84	D
Psychology	110	F	84	A
Sociology	110	F	84	D

Department	Level	Term	Year	Grade
Chemistry	101	S	85	C
History	200	F	85	B
Music	110	F	85	C

Department	Level	Term	Year	Grade
Biology	101	S	86	C
Chemistry	110	F	86	C
History	220	U	86	C
Math	200	S	86	A
Math	220	F	86	C
Music	200	S	86	I

You can see that Paradox repeats the heading rows from the table band before each year. That's

