Creating Tables

ACCESS

Normalisation Techniques

INTRODUCTION

A database is a collection of data or information. Access for Windows allow files to be created, each file consists of many records; the records contain various fields which hold data items.

Each record contains information about one ‘thing’; this could be a stock control item, a customer or an employee. The file contains all of the records in the same format; - that is each employee would have a surname, date of birth, salary and so on. You do not mix different sort of records in one data file; - do not put information about stock and information about people in the same file. The term database refers to a collection of files; the advanced skill in the use of database is the ability to make links between files. However, it is very useful just to be able to use one table, and produce outputs from it. The two main outputs are report and the query.

USE OF ACCESS

ACCESS for Windows can be used to produce reports with data manipulated any way you wish. The data is organised into tables e.g. could be the Customer table including Customer No., Name, Address, etc.

Another table could be the Orders table and these would link together using the Cust# as a key field. The Name and address etc. does not have to be stored more than once. Key fields are optional but are beneficial as they prevent duplicate values in a key field, where it is important e.g. for a Cust# and can speed up processing.

STARTING ACCESS for Windows

Find the ‘ACCESS’ icon on the Windows screen.
Move the mouse pointer over the ‘ACCESS’ icon.
When the mouse pointer is over the icon you can either press the left button twice or press once and then press 'Enter', this will take you into ‘ACCES’.

CREATING A DATABASE

1. Double click ‘File Menu’;
2. Click ‘New Database’ command;
3. The ‘New Database’ dialog box will appear;

4. In the ‘File Name’ box type the name of a file (a database file can contain up to 8 characters but can not contain a space);
5. To store the database in a different directory select the drive from the drives list;
6. Click OK.

CREATING A TABLE

When ‘ACCESS’ doe Windows has started, the desktop appears. The desktop is the central working area in ‘ACCESS’. All tasks are started from the desktop. In ‘ACCESS’ information is stored in tables. Tables are lists of information arranged in meaningful ways.

Click ‘New’ button. Creating a table consists of mainly defining the fields in the table. Each field has a name and a data type.

The *field name* identifies the data stored in a field. A field name can contain up to 64 characters including space.

The *data type* tells MS Access what kind of data goes in the field, such as text, numbers, date or currency. When you add a new field, ‘ACCESS’ automatically assigns it the text data type. If you want the field to have a different data type, choose the data type you want from the drop down list box in the Data Type column for the field.

WHAT IS A PRIMARY KEY?

A *Primary key* consists of one or more fields that uniquely identify each record in a table. An ID number of code often serves as the *Primary key* because this type of value is always different for each record. Access’s

Primary keys are especially flexible, because you can specify any field, not just the first field in the table. ‘ACCESS’ automatically creates an index for a table’s primary key. A *Primary key* is similar to a dbase unique index except that ‘ACCESS’ automatically keeps the index on the *Primary key* current and uses it to search and sort records. You can also create indexes for other fields you search or sort on often.

WHAT IS UNIQUE INDEX?

For example, suppose you had the personnel records for a large company stored on a database. There are over 50,000 employees within the company and it is necessary for you to find different employees record so that they can be updated. If the records were not indexed, every time you try to find the employee record for 'Mr Smith' you will need to read from the beginning of the database until you find 'Mr Smith'. This operation could take hours to complete.

However, if the records were stored or indexed in alphabetical order then finding the record for 'Mr. Smith' would be a great deal easier. What if they are 3 'Mr. Smith'? Which record would be the correct one? This highlights the reason why you should try and make your index (primary key) unique, so that every record is different and can be found easily. Staff numbers would be the ideal field to create as an index, because it is unique, no two employees' numbers are ever the same.

Therefore, if you need to find 'Mr. Smith's record and you know his staff number them the record could be located in a matter of seconds rather than hours.

FIELD PROPERTIES & VALIDATION

'ACCESS' automatically validates values based on a field data type; for example 'ACCEESS' will not allow text in a numeric field. You can set more specific rules for data validation rules.

WHEN SHOULD YOU VALIDATE DATA?

Data should be validated:

- When you want to cut down the amount of errors that can occur in data entry
- When you want data to be entered in a specific format only
- With fixed length (only xxx amount of characters can be entered)

For example if you were asking someone a question about their date of birth, you could set up a validation rule that only allowed them to enter their data in a specific format i.e. Day/Month/Year, and only allowed them to enter a date which makes them over the age of 18. This will ensure validity of data and cut down the number of errors made.

SETTING FIELD PROPERTIES

In the lower portion of the table window, you can set properties for the current field. For example, you can set the size of text or numeric fields. This is also where you define validation rules and default values to assist in the data entry and prevent mistakes in your data.

- Click in the field name row
- Type in the word surname and press the Return key
- The cursor will move automatically to the *Data Type* field

In the lower portion of the window, change the value in the field size box to 20 and press the *Return Key.*

- No close the table without saving any changes and open the *Employee Details* table.

SETTING A PROMARY KEY AND SAVING YOUR TABLE

Every table in your database should have a *Primary Key*. A field with a counter data type makes a perfect *Primary Key.* You know the field will contain a unique value for each record because 'ACCESS' automatically enters sequential numbers in that field for each new record of data.

SETTING the PRIMARY KEY

- Click the *Table Object* button
- Open the *Employee Details* table
- Change the *Employee Details Table* to *Design View* by clicking the *Design View* button
- Click the *Primary Key* button on the toolbar

'ACCESS' puts a key symbol in the row selector for the Staff number field. This symbol tells you that the *Staff Number* field is the table's primary key.

SAVING THE TABLE

- Click *File menu*
- Click *Save* command

ADDING RECORDS IN A TABLE DATASHEET

There are two methods of viewing a table, one is *Design View* (the one we have just been using), and the other is *Datasheet View*. To switch to Datasheet view you use the view buttons on the left side of the toolbar to switch back and forth between the views.

From the *Toolbar* slick the *Datasheet View* button.

You can now add records to the table in *Datasheet View*.

ENTERING A RECORD

NB: 'ACCESS' puts the cursor in the first field of the table (Surname), use the *Page Down* key on your keyboard to move to the next blank record.

- Type in your Surname
- Press *Tab* to move to the *Forename* field
- Type your forename and press *Tab* to move to the *Date of Birth* field
- Type in your Data of Birth
- Continue entering the data. When you have come to the *Overtime* field Type *Yes* and then stop.

SAVING A RECORD

To save a record press *Tab* to move to the next record.

When you move to the next record, 'ACCESS' automatically saves the data in the first record. You don not have to do anything else to save the record.

ADDING MORE RECORDS

Now add two more records to the *Employee Details* table. Use your family of friends for the two records.

CLOSING the TABLE

- Click *File Menu*
- Click *Close* Command
- *Close* the database

THE CONCEPT OF NORMALISATION

Data values should not be stored twice. If they are, it is a waste of space and more importantly, leads to updates only being applied to some of the copies of the value.

Each record is a collection of data items, which have some unique key. In any one record, the data items can only have one value for each key.

If a formal system has a set of repeating entries on one form, then each repeating entry becomes a record in a table.

Some business applications can be implemented using only one table, for example a stock file of a mailing database.

Many other applications use several tables; stock monitoring with records of purchase orders being delivered and despatches being sent out.

The correct system for an internal telephone directory giving departments, employees, rooms and extension numbers needs six tables to cope with such concepts as one room having more than one telephone extensions or two employees sharing the same extension.

End-users require an understanding of these concepts before they can design systems, which are capable of solving a problem. The target of the developer should be that, other people could use system without hours of explanation, or volumes of documentation

THIS IS SELDOM ACHIEVED – usually due to lack of system design.

The data that is to be stored in tables need to be thought about. One factor that needs to be taken into consideration when deciding what tables will be needed is the reality of the system. How many repeating lines mean a new table is needed?

A simple example of this is a list of companies and their telephone numbers. There could be a field for the switch board number, a fax number and an emergency number. This means that we can use one record per company, with three telephone number fields. If one company has fifty external telephone lines then we should NOT allocate fifty field for the numbers in one record; we should start a new table with one field for each telephone number, and another filed to link the telephone number table to the company table. WE do not normally want to repeat the company name in the company name in the telephone number table, as this would be a repetition. If each company has a unique identifier, then this can be stored in the telephone number table.

We have already mentioned that each table normally has a unique key ‘ACCESS’ allows the key to be constructed from two fields such that the combination of values from the two fields is unique. It is NOT compulsory to have a key for a title; the existence of a key serves more that one purpose though.

KEY VALUES

'ACCESS' ensures that no two records in the same table can have the same key. If we use a 'part number' to identify a line of stock, then this must be kept unique. If two different lines of stock had the same key, then chaos could be caused.

The key also creates a sequence for the records. However, the user should not try to use values for a key to sequence a report. The data can be presented in different sequence in different printouts without using any keys.

The key also provides quick access to a record, which is important if there are hundreds of records in a file. This feature is used frequently by 'ACCESS' when the user is interacting with more than one table at once, via a form, a report or a key.

CHOOSING A KEY

Sometimes the choice of a field for a key is self-evident, as with example such as Employee Details (Payroll number) or stock file (Part Number). If the table is a list of companies, then the choice is not so obvious. If the companies were customers, then a Sales Ledger system would specify an account number for each. If they are a mix of useful contacts such as Bank, Inland Revenue, Software support, Consultant and so on, then there is no obvious key.

In this situation, an artificial key needs to be created. The old fashioned way to do this was to sue some letters of the company name, a letter to give their relationship to our organisation to our organisation and some geographical reference. This style of solution is now largely discredited, and it is more common to use an artificial numbering system such as A001, A002 and so on. The length of the code should be as short as possible, but providing enough possible values to cope with any predictable number of records.

An alphabetical key provide many more possible values than numeric, but care should be taken to avoid problems such as A 1 not being the same as A1 or A 1.

Sometimes it is desirable to use two fields for a key. If we wish to keep records of daily usage of pool cars, then we could use the combination of the car registration numbers and the date. This would not permit the storage of two records for the same car for the same day though. A system to record quality wines in stock will need to have a list of the wines, and then a second table indicating the number of wines in stock for each year. In this case, the key to the wine table could be vineyard number, and the key to the second table would be the combination of the vineyard number and the year.

It is always possible to create a new field to act a key in one table, and store the other fields as non-key data items.

It is important to realise that a value that is unique key in one table is often stored in some other tale. It will not usually be a unique key to the other table, it may be part of a two-part key or it may be a non-key field.

COMPANIES and PEOPLE

If we need to set up a database of people and the organisations they work for, then it should be apparent that we will need two tables. One table will give the name of the organisations, and address, the telephone number and so on. The second table will need to give the name of the people, their extension or mobile telephone number, their position in the organisation and so on.

We must establish a unique key for each organisation, it must have a name and we must decide its type. A three-character text field, used with a letter and two numbers has more than two thousand values, assuming we only use capital letters.

The table with data concerning people does not have an obvious key, but it is NOT essential to have a key. It would be very wrong to say that the organisation code was the key; this would mean that each value could only be entered once and so data could only be kept about one person for each organisation.

The files are therefore:

COMPANY	Company ID	A	3
	Company Name	A	30
	Address1	A	20
	Address2	A	20
	Post Code	A	8
	Phone	A	20
PEOPLE	Company ID	A	3
	Surname	A	15
	Forename	A	10
	Title	A	5
	Work Ext.	A	12
	Site	A	10

Possible problems?

No key, so how do we know which record is which?

Site: - this could be duplicate data, as the same will appear for each person that works there. True, but is it very likely that a site will change its name?

If a person works at a site, what is the site phone number? Again, we can put the whole number rather than just the extension.

What happens if two organisations share the same phone number: - put the extension in the phone number.

How de we access the data, how do we find the company ID when adding new records etc.

Wait and see!!!

Once the tables are designed and the file definitions entered into 'ACCESS' then we can think about form design for data input. There are two alternatives for having data from both files on screen at once.

Firstly, it is possible to design a form for displaying / editing the data in the *People* table. This form can be used in conjunction with 'referential integrity'. What this means is that when a new record is added to the *People* table, it has automatically give the Company ID from the value in the record from the *Company* table which is on the screen. It also means that if the company ID for one record in the *Company* table is changed, then all the records with the matching values in the *People* table will be changed.

Alternately, a form can be designed for adding a record to the *People* table that represents the use with a view of the records in the company table when they need to fill in the field called *Company ID*.

There is no reason why both forms cannot be designed, and then the user can choose the one they prefer according to the job they are doing. Obviously, the former is very useful for adding a list of people who work at the same organisation. It is not possible to have both solutions implemented at once.

Once the data has been entered, it may worth creating a query which links the two tables together with an example element for the Company ID field in each of the two tables.

If the designer wants to make the system available to others who cannot cope with multiple table queries, then they can break the rules. Why not store the name of the organisation in the *People* table. To database designer, this is absolutely wrong. However, if organisations seldom change their name, then the duplicated data is only wasting space. If a PC has a hard disk of hundreds of megabytes, and there are only a few hundred records in the *People* table then the space is not important. Rules may exist to be broken, but the rule breaker must be aware of the consequences.

'ACCESS' assists the rule breaker by having the facility to automatically copy the data for the *Company name* to the *Company* table, as long as there is a field with the same field name in both tables. The above table definitions did not include this, but the can be added afterwards.

By the same logic, it could be argued that the phone number stored in the record for the *Company* table could be copied, if the argument is taken to the limit, then all the fields could be copied into both tables. This is then defeating the purpose of the data normalisation; it is better to design systems and provide users with necessary skills to do their jobs with data that is as good as possible.

NORMALISATION TECHNIQUES

The steps in Normalising data are:

- Convert the source data into a list of attributes;
- Put repeating groups of data into separate structures;
- Put data that is only dependent on part of the key into separate structures;
- Put data that is dependent on other non-key data into separate structures;
- Combine structures with identical keys.

The following source document will be used to illustrate the effect of each step.

SALES ORDER RECORD				
Order No.	**Order Date:**	**Order Value**	**Priority**	**Sales Person**
Customer No.	**Customer Name:**			
Invoice Address:		**Consignee Address:**		
Order Details				
Product Code	**Description**	**QTY**	**Price**	**Delivery Data**
Delivery Instruction:		**Special Instruction:**		

Convert the source data into a list of attributes
(UN-normalised form (UNF))

UNF

Order No.

Order Date
Order Value
Priority
Sales Person
Customer No.
Customer Name
Customer Address
Consignee Address
Invoice Address
Product Code
Description
Quantity
Price
Delivery Date
Delivery Instruction
Special Instruction

Put repeating groups f data into specific structure
(First Normal Form (FNF))

Order No.	*Order No.* *Product Code*
Order Value	
Order Value	Description
Priority	Quantity
Sales Person	Price
Customer No.	Delivery Data
Customer Name	
Invoice Address	
Consignee Address	
Delivery Address	
Delivery Date	
Delivery Instruction	
Special Instruction	

This is called the First Normal Form (FNF)

Put fields which are only dependent on part of the key into separate structures (Second Normal Form)

This step is also called '*removing partial dependencies'.*

The description attributes in the extracted group is only dependent on the product code part of the key, not the order No. So that is extracted to a new group. Its primary key is product code. In simple terms

this is saying that the description of a product will always be the same, so why not hold it once under the product code, rather that repeat it every time it is ordered.

There are now three groups of data:

Order No.	*Order No.* *Product Code*	*Product Code*
Order Date		Description
Order Value	Quantity	Price
Priority		
Sales Person		
Customer No.		
Customer Name		
Invoice Address		
Consignee Address		
Delivery Address		
Special Instruction		

The above example assumes that the price is per unit. The analyst would have to check that, this was the correct meaning and not, say price per unit x quantity ordered.

Put fields that are dependent on other non-key fields into separate structures.
(Third Normal Form (TNF)).

This step is known as '*removing mutual dependencies'*.
The fields '*Customer Address'*, and '*Invoice Address'* are only dependent on the customer number.
They are extracted to a separate group with *Customer Number* as the primary key.
Customer Number needs to remain in the original data group as a foreign key to provide a link to the extracted data. Again, in simple terms, the reasoning is that if the 'Customer Address' and 'Invoice Address' will always be the same, hold them only once with the master details of the customer. The foreign key is marked with an '*'.

Order No.	*Order No.* *Product Code*	*Product Code*	*Customer No.*
Order Date		Description	Customer Name
Sales Person	Quantity	Price	Invoice Address
*Customer No.	Delivery Date		
Consignee Address			
Delivery Instruction			
Special Instruction			
Priority			