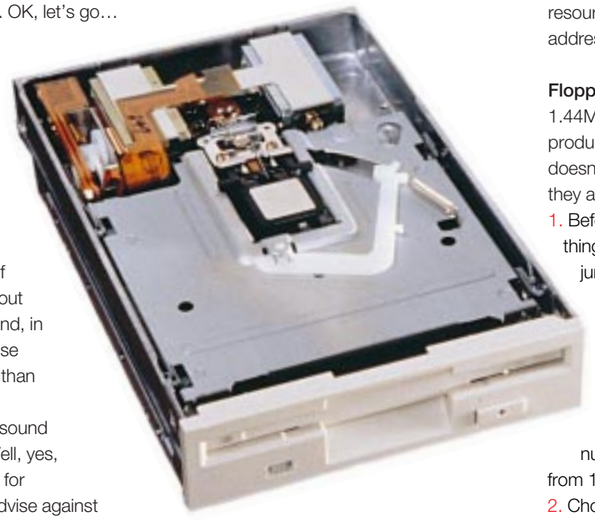
Drive on

Nearly there. It's all coming together now, and in part IV of his series on how to build your own computer, Roger Gann tells you how to install the expansion cards and fit the drives.

For those who came in during the interval, it's been a roller-coaster ride of a project! Yes, you've missed some well-crucial instalments. (You can obtain back issues of the magazine — see the panel, *Back Issues*, p214).

OK, to recap: in part one I covered the pros and cons of the build-it-yourself PC and led you through choosing the motherboard and the system unit. In the second part I looked at specifying all the components needed to build your PC; choosing a games PC for the simple reason that it represents the most powerful and well-specified PC you can currently buy. I then started to prepare the motherboard. Last month, we got around to putting in the motherboard and began installing the expansion cards. This month, I'll complete this and fit all the drives, the floppy and the hard disk. OK, let's go…

New breed of super-floppies: Iomega's Zip drive (top), with OR Technology's LS-120 (beneath)



Expansion cards

With so much I/O now integrated on the motherboard, the number of expansion cards required by a modern personal computer can often be counted on the fingers of one hand. I'm talking about the graphics card here and, in 99 out of 100 cases, these require no further action than simply plugging them in.

"OK, what about the sound card?" I hear you say. Well, yes, this is a prime candidate for installation but I would advise against fitting it at this juncture, simply because we won't be in a position at this stage to avail ourselves of the miracle that is Plug and Play, and believe me, this can make life a lot simpler. So hold off for a while until we've installed our operating system (Windows 95) and let it sort out the arbitration of the multitude of hardware resources, the IRQs, DMAs and I/O addresses needed by such devices.

Floppy disk drive

1.44Mb floppy drives are now generic products, almost unbranded, and so it doesn't really matter which one you buy as they are all much of a muchness.

1. Before fitting it in a drive bay, there's one thing to do: check that the Drive Select jumpers (or switch) is set to the first drive. A PC can support up to four floppy drives and this is a way of discriminating between them. It may be Drive 0 (DS0) or 1 (DS1) depending on the drive you've bought. Some start their drive numbering from 0, while others start from 1.
2. Choose a suitable 3.5in drive bay and unclip the blanking panel from the system case front bezel.
3. Slide the floppy drive into the bay and when it is flush with the front panel, insert the mounting bolts and tighten them up.
4. Plug in the power lead (it will be the smaller of the two types of power-lead plug).
5. Using the 34-way ribbon cable which came with the motherboard, plug one end into the connector at the back of the drive and the other into the floppy interface connector on the motherboard. Ensure that the coloured edge of the ribbon cable aligns with Pin 1 at both the drive and motherboard ends.

This cable may have an extra connector for a second floppy disk drive. It may also have a "twist" in the middle of the ribbon cable, between the two FDD connectors. This twist is a production convenience for PC manufacturers because it lets them set all their floppy disk drives to be the second option (e.g. DS1 or 2) and they let the choice of connector used (i.e. before or after the twist) determine whether it's a Drive 1 or 2.

In this case, with your drive set to Drive 1, make sure you use the connector that precedes the cable twist. Past experience has taught me that, for reasons unknown, sometimes you have no choice but to set the floppy to Drive 2 and use the twisted part of the cable in order to get the floppy drive to be recognised. So prepare for a degree of faffing around when getting the system to recognise the drive.

6. With the drive installed, enter CMOS Setup and specify that you have a 3.5in floppy installed. Most BIOSes default to this anyway so you may not have to change anything.
7. Quit Setup, saving your settings, and reboot. At this point it makes a lot of sense to test that we can boot the PC from our new floppy drive before proceeding further. So bung in a bootable floppy and hit the Reset button, crossing your fingers in the meantime.

The Quantum Bigfoot, like most IDE hard disks, requires a 5.25in drive bay

Other floppy drives

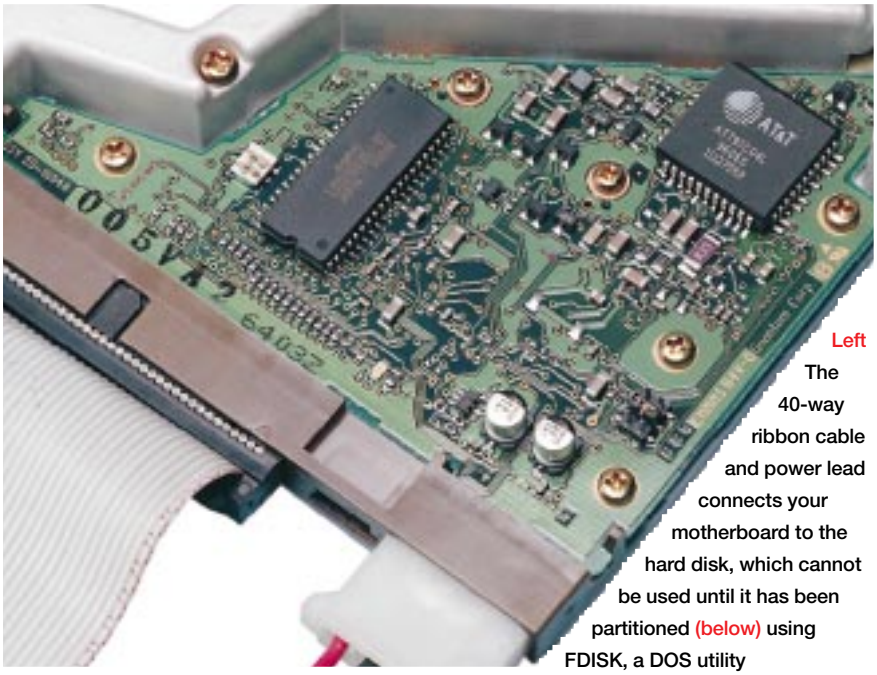
If you have a really modern motherboard you may find that it supports the new breed of super-floppies as bootable devices — I'm talking about Zip and LS-120 drives, here. At present, internal Zip drives (about £90 ex VAT) require a SCSI host adapter in order to work, which is outside the scope of this project. However, the LS-120 (about £160 ex VAT) is an ATAPI device and can be easily attached to the second IDE channel on our motherboard. It's also directly supported by the OSR2 release of Windows 95.

Externally an LS-120 looks just like a normal 1.44Mb/3.5in floppy drive but at the rear it has a 40-way rather than a 34-way connector. It also has jumpers for Master and Slave settings. If you are going to fit an LS-120 then it's probably best to attach it to the second IDE channel as a master, and slave the CD-ROM off it.

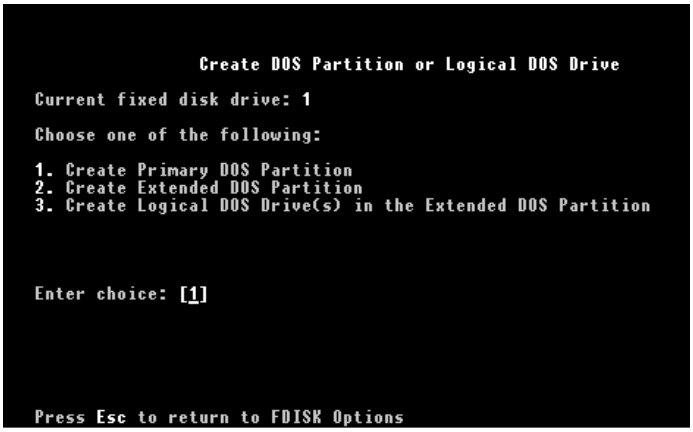
The hard disk

1. Check out the new drive's fixings and where it's going to fit in the system case. It won't need an externally accessible drive bay so look first at installing it in an internal bay. Most, but not all, IDE hard disks are 3.5in devices and so this shouldn't be a problem, but if you opt for a Quantum Bigfoot drive it will have to go in a 5.25in bay, which will probably mean losing an external drive bay.
2. Make sure you have the right mounting hardware, too; things like bolts or rails. Mount the drive rails if needed. If your new drive came with its own mounting screws, use them. Avoid using just any old bolts: if they're too long, they can damage the drive's electronics.
3. Power down the PC. Install the new drive in its bay — slide it in and tighten the mounting bolts.
4. Attach the power cable and the 40-way

p214 ➢



Left The 40-way ribbon cable and power lead connects your motherboard to the hard disk, which cannot be used until it has been partitioned (below) using FDISK, a DOS utility



ribbon cable to the new drive and to the primary IDE channel on the motherboard.

Some ribbon cable connectors have a "polarised" shroud, with a little notch which prevents the cable from being inserted incorrectly. If yours isn't, again make sure that the coloured stripe on the ribbon cable goes to Pin 1 of the connector on the motherboard and the hard disk. The power lead is always notched to ensure correct insertion.

5. Power up the PC. The new hard disk hasn't been "initialised" yet and thus is incapable of booting. So at this point press the DEL key (or whatever) to access your CMOS Setup program.

6. You now have to tell the CMOS the "geometry" of the new drive: that is, the number of heads, cylinders and sectors per track, of the new drive. You can either do this manually, keying the figures into the "User Definable" section, but it's much easier to select the option that auto-identifies the new drive. Save the new CMOS settings and quit Setup.

7. Restart the PC, this time with the system floppy in the drive. Now, even though your PC can recognise your new disk, you still can't actually use it until you perform two additional, related operations: partitioning and formatting (see also "Big drive partitioning", above).

Partitioning is carried out using the venerable DOS utility, FDISK:

- Boot from the system floppy.
- Run FDISK and select option 1 from the menu "Create DOS Partition".
- Select option 1 from the second menu "Create Primary DOS partition".

For reasons of storage efficiency avoid the temptation to make a single humungous partition occupying the entire hard disk. Make it no bigger than 511Mb: this will give you a cluster size of eight 512-byte sectors (or 4Kb), which is tolerable.

By contrast, if you create a single partition on a 2.1Gb disk you'll wind up with 32Kb clusters, which is a profligate waste of disk space.

Sadly, while the OSR2 version of Windows 95 supports FAT32, which overcomes this slack space problem, it isn't commercially available. The retail version of Windows 95 still supports the inferior FAT16 file system and so this is the regime to which we must adhere.

8. Press ESC a couple of times to back out of FDISK and reboot the PC using the system floppy.

9. You can now format the new primary partition on the hard disk. Type:

```
FORMAT C:/S <CR>
```

Not only will this format the drive, making it usable, but the /S switch copies the system tracks across. Depending on the size of the drive this could take a couple of minutes.

10. The hard disk should now be bootable so you no longer need the system floppy to boot with: test it by hitting the Reset button.

And that's enough drives for this month. In the next part of the series, I'll perform the topping-out ceremony of this building project: installing the CD-ROM drive, installing the operating system, and tweaking the system to make sure we get the most bangs for our buck.

Big drive partitioning

If you've bought a really big drive (<2Gb) you should be aware that the largest partition possible under FAT16 is 2.1Gb or 2,146,959,360 bytes. Hence, drives larger than this have to be partitioned into smaller logical drives using FDISK.

- Assuming you've created a 511Mb primary partition, load FDISK again and this time select option 2 from the second menu, Create Extended Partition.
- You can then devote the remaining space on the drive to the "Extended Partition".
- From this you can create "logical" drives.

For example: say you had a 2Gb hard disk with a 511Mb primary partition; you could create a 1.5Mb extended partition and from this carve out three 511Mb logical drives.

Back Issues

See page 12, or this month's *PCW* CD-ROM, for how to obtain Parts I, II and III of the Workshop, which appeared in our June, July and August '97 issues. The final part (V) appears next month (October issue).

PCW Contact

Roger Gann can be contacted by post c/o *PCW* at the usual address or via email at hardware@pcw.co.uk.

Cable junction

In the third part of his Workshop series on building your own PC, Roger Gann explains how to install the motherboard, how to connect the cables, and how to check that all is well so far.

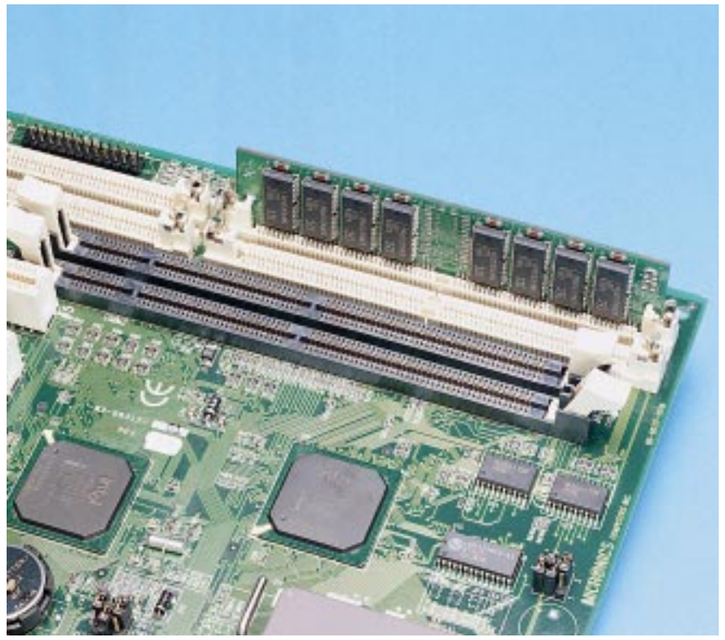
Last month I looked at the pros and cons of the build-it-yourself PC and led you through the process of choosing the motherboard and the system unit, and specifying all the components needed to build your PC. I chose a games PC for the reason that it represents the most powerful PC available. I then began preparing the motherboard.

This time I'm going to describe the final installation of the motherboard, and begin installing the expansion cards.

Install the new motherboard

1. Give the new motherboard a final once-over to make sure you've correctly set all the jumpers. Check that the SIMMs are correctly mounted in their slots and that the CPU has been orientated properly.
2. If you have a "cache-on-a-stick" module, make sure it's plugged into its socket. If you have a mini-tower system case, lay it on its side so the area where the motherboard will go is at the bottom.
3. Slide the new motherboard into position in the base of the case so the stand-offs engage in the corresponding locating holes in the floor: the position of the keyboard socket (and the hole in the casing for it) will help you locate the motherboard correctly.
4. Slide the motherboard as far to the right as possible (this may entail a little wiggling).
5. Make sure all the stand-offs have engaged in the base. Your system case may have several threaded nuts on the base — align the remaining holes in the motherboard with these and fit the supplied bolts to secure the motherboard in place.

Sometimes the system case may lack mounting holes that correspond with holes in the motherboard. This can be a problem, especially at the front on the motherboard which may be unsupported as a result and may flex alarmingly when inserting cards or processors. The solution is to install stand-offs in the appropriate holes but to cut off the lugs that fit the holes in the system case, so the stand-offs merely act as legs.



Make sure you check all your motherboard components thoroughly before mounting it in its case

Cabling

Having fixed the motherboard in place in the system case, our next task is to connect the myriad cables. These are the power cables, the cables which drive the LED displays, the loudspeaker and the reset button.

■ First in line are the two power cables, the two chunky female plugs with multi-colour cables attached. They will be labelled P8 and P9. By the rear right-hand corner of the motherboard you'll find the male socket for these plugs — it's probably white or cream and about 50mm long.

The plugs are identical, and while it's impossible to fit them facing the wrong way, it's certainly possible to put the one that belongs on the left, on the right, and vice-versa. Luckily, there's an easy way to orientate them correctly; fit them so that the black cables on each plug go together.

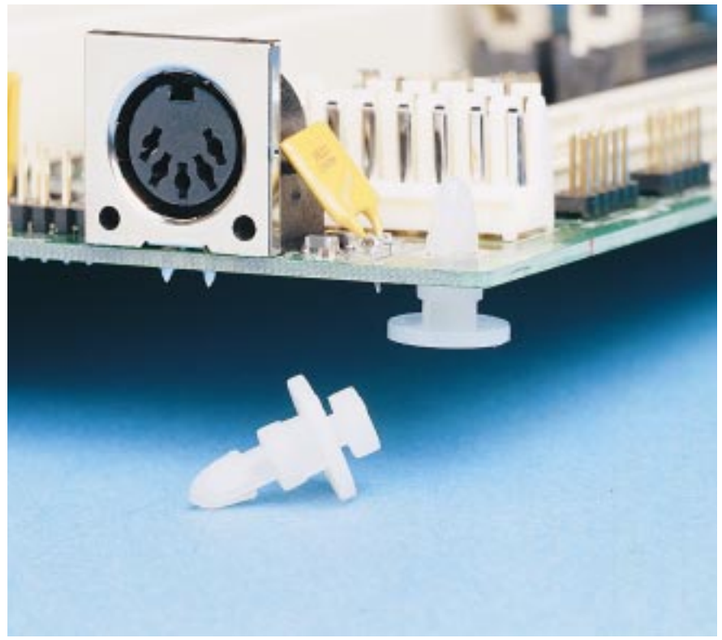
■ If you have an ATX motherboard, its power connector is slightly different: it's a single 20-pin socket and you'll need an ATX-style case that has a PSU (Power Supply Unit) with the appropriate connector.

■ The most fiddly task of all is fitting the little multicoloured cables. Typically, these are positioned along the front edge of the motherboard. Your case will have quite a few of these and will include connectors for these cables: keylock, reset, power, turbo LED, and switch and speaker.

There's no earthly reason why the number of leads and plugs attached to the case should match the number of connectors on the motherboard, so occasionally you'll have to use your noddle when ne'er the twain shall meet. For example, the leads can be grouped together to form just two or three plugs. If this is a problem because the motherboard connectors are not so arranged, it's OK to split the plastic plug into two separate plugs with a sharp knife. Pay close attention to the schematics provided in the motherboard manual and make sure you get the orientation of the plugs right. Often the connectors are tightly crammed together and badly labelled.

■ If you're fitting a 486 processor, you should now fit the heatsink which just clips on over the processor. If you are fitting a DX4 or a Pentium, you will need a combination cooling fan and heatsink — and this will need to pick up power from somewhere. Modern motherboards often have a power connector specifically for the fan, located adjacent to the ZIF socket, while others may have a fly lead which will take its power from one of the peripheral power leads.

Your PSU may make no special provision for the cooling fan, so you may have to buy a special lead and plug it in to one of the normal power leads. Buy a "Y" power lead adapter to avoid sacrificing a precious power lead.



If the system case lacks mounting holes for your motherboard, slice the standoffs in half so that they sit flat on the base

I/O ports

All modern motherboards now feature integrated I/O (input/output). Rather than use dedicated I/O cards which hog expansion slots like a serial/parallel/floppy/IDE card, these ports are now built into the motherboard. So the next step is to

p242 ➢

Check that the CPU is correctly mounted in its ZIF socket

commission your new PC's basic I/O.

■ Because expansion cards are no longer needed, the various sockets for these I/O ports have to be separately mounted on the chassis. You should find a set of these ports (a parallel and a pair of serial ports complete with grey ribbon cable and sockets) included with the motherboard.

■ Sometimes these are mounted on an expansion slot blanking plate, which makes installation a cinch — you just substitute a spare blanking plate for this one. Try to use the blanking plate closest to the power supply to avoid losing the use of a precious PCI. Make sure the lower socket doesn't foul up anything on the motherboard (they tend to be located low down on the blanking plate).

■ You may have to use cut-outs in the rear panel of the system case to mount your ports. These are held in place by a bit of solder and a light tap with an old screwdriver will dislodge them. Undo the bolts on the sockets, offer up the socket from inside the case and tighten up the bolts. They can be a fiddle to tighten because they're mounted so close to the socket itself. Install the parallel two serial ports in this way.

■ Complete the job by plugging the ribbon cable attached to each port into the appropriate header connector on the motherboard. Pin 1 should be marked on the motherboard. Align the coloured edge of the ribbon cable with Pin 1. Better motherboards use shrouded connectors with a polarising notch which prevents you from inserting the motherboard plug incorrectly. (Note that some motherboards don't have a joystick port, relying on the fact that most sound cards come with one.)

■ Another connector you might come across is the PS/2 mouse connector. There should be a motherboard connector close to the keyboard socket and the mouse port is almost certainly supplied on a blanking plate. Check the Pin 1 alignment as above when plugging in the data cable.

Fit the peripherals

That completes the cabling phase and we can start fitting the peripherals at long last. But we won't get carried away; we'll start with the graphics card. The reason for this is that I want to do a little preliminary system checking and make sure that what I've done so far is OK before proceeding any further. So, unpack your graphics card, again remembering to discharge any static electricity before touching any electronic components on it.

1. The card will probably be a PCI card, so fit it in a free PCI slot. Undo the bolt securing the blanking plate at the end of the slot and remove the plate.
2. Hold the card firmly by its top edge and press its edge connector firmly into the expansion slot (it may be a tight fit and you may have to use a modicum of steady force). Tighten up the bolt to stop the card from flapping around.
3. Now, without putting the cover back on (but taking the usual safety precautions), plug in the keyboard and monitor to their respective ports at the back of the system unit. Plug in the mains lead to the system unit and hit the on/off button.

Is it happening?

With luck, things should start to happen: the power LED should light, the CPU cooling fan should spin and the floppy disk drive should make a little grinding noise ("perform a seek" in technical jargon). The motherboard's BIOS should now perform its POST (Power On Self Test). It will count through the installed memory as it checks it, so watch this.

There's no boot device so your new PC cannot boot yet, but you can enter CMOS setup (typically, by pressing the DEL key at boot time) and set useful things like the time and date. This also confirms that the basic PC is a working machine.

If nothing happens, check the mains lead, the two mains plugs and the miniature multicoloured cables. Also check that the graphics card is plugged in and that the monitor is connected to it. Check the CPU in its ZIF socket. If the LEDs light up, but nothing happens on-screen and instead you hear a series of beeps, you have a fundamental error condition: listen to the pattern of the beeps and then consult the motherboard manual to see what error condition they indicate. Often it is a problem with main memory or with the graphics card, so try re-seating them in case it is a poor contact.

Next month, we will fit the remaining peripherals.

Back Issues

See p12, or this month's *PCW* CD-ROM, for how to obtain Parts I and II of the Workshop, which appeared in the June and July '97 issues.

PCW Contact

Roger Gann can be contacted by post c/o *PCW* at the usual address or via email at hardware@pcw.co.uk.

The games that PCs play

Why not build your own PC? It's easier than you imagine. Here, in Part II, Roger Gann considers a high-power spec, prepares the motherboard, and shows how to fit the SIMM and stand-offs.

Last month we looked at the pros and cons of building your own PC and how to go about selecting those most anonymous of components, the system case and the motherboard. This time around I'll deal with specifying the ultimate games PC, from the ground up.

When I tell you that a top games PC is probably the most powerful, this side of a dedicated graphics workstation, it gives you some idea of what is possible when you have the luxury of being able to build your own PC. But if you don't actually need that much power, you'll be able to scale down the specification accordingly. I'll also be covering the first stage of assembly — the preparation of the motherboard.

You might think that the most powerful PCs are invariably to be found on business desktops? Wrong. The most powerful PCs are those dedicated to playing games, and these are found in the home. If you're serious about games-playing on a PC, then you'll need a serious PC on which to play them. And I'm talking about *really* serious, modern games, which incorporate all manner of multimedia effects: sound, animation, video and 3D rendering; these test the capabilities of a PC like few other programs can. It may sound preposterous, but Intel is aiming the latest and fastest 200MHz MMX Pentiums at the home market simply because games run better on the fastest processor you can buy.

The processor

So, the good news is that as far as games are concerned, you now have two choices. Your first port of call is the 200MHz Pentium MMX. The Pentium Pro is undoubtedly a more powerful processor, but it's not so hot when it comes to running 16-bit apps and doesn't have MMX support (although the Pentium II will), so we'll shortlist the 200MHz Pentium MMX. Although not many games support MMX at present, the list is growing, and it's a must-have for the serious games and multimedia user.

The other hot candidate for fast game play is the recently announced AMD K6 processor. This is faster than the Intel Pentium and in some cases is faster than the Pentium Pro. It's also about 25 percent cheaper than the equivalent Pentium and available in a 233MHz version too, making it both cheap *and* fast. Add to this the fact that there's no performance penalty when running 16-bit apps, that it supports MMX and that it is Socket 7 compatible, and the K6 begins to look very interesting.

Because the K6 is so new, make sure the motherboard you buy supports this advanced CPU. (*See last month's column, which dealt with selecting a motherboard*.)

Storage: hard disks & CD-ROM drives

Although the most powerful games will run almost exclusively from CD-ROM, the hard disk still has an important role to play, so you should ensure that you've got the fastest-possible hard disk subsystem. The vast majority of new PCs these days come equipped with Enhanced IDE hard disks and, thanks to the modern miracle of Mode 4 PIO, can belt out data at a cracking pace of up to 16Mb/sec. Very soon, though, motherboards will appear which support Ultra DMA EIDE, delivering 33Mb/sec throughput.

OK, so the EIDE transfer rate is good; but there's a price to pay, and that is extremely high CPU utilisation rates. But thanks to PIO, the CPU alone has to supervise every chunk of data that is hoovered up off the disk and this takes up plenty of its time; time which could be better spent actually running your game.

Games purists should avoid Enhanced IDE hard disks altogether, good though they are, and instead plump for SCSI because it is clever and handles all data transfers itself, thus not wasting processor power.

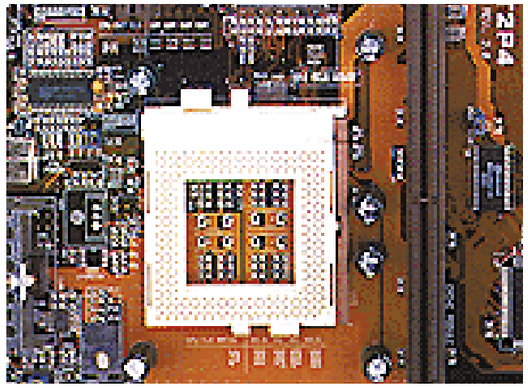
Here, I'd recommend something like the Adaptec AHA-2920 or 2940 host adaptor cards coupled to a Seagate Hawk Fast SCSI-2 drive. Or, if you're very keen, go for Ultra Wide SCSI such as the AHA-2940UW plus an Ultra Wide SCSI drive. This combination can deliver phenomenal throughput in the region of 20Mb/sec to 40Mb/sec.

For the same reasons, you'd have to choose a SCSI CD-ROM as well. Even though most modern CD-based games are mastered on four-speed drives, it is a good idea to get a faster drive. Right now, the fastest CD-ROM drives are 12- and 16-speed models. Unfortunately, these are mostly IDE/ATAPI drives which make severe demands on the CPU to deliver this kind of performance: SCSI CD-ROM drives don't, so I'd recommend these above their ATAPI counterparts any day. Sadly, they're also considerably more expensive, but I would bite the bullet and opt for something like the new Plextor 12/20Plex drive which can deliver data at 3,000Kb/sec.

Graphics cards

Without doubt, if you want the most realistic-looking games you'll need a graphics accelerator which supports 3D graphics. Unfortunately, the development

Step by Step — Motherboard, SIMM and stand-offs



Preparing the motherboard

1. First, get your toolkit together. You'll need:
- a Phillips screwdriver;
- an electrician's screwdriver; and
- a pair of fine needle-nose pliers.

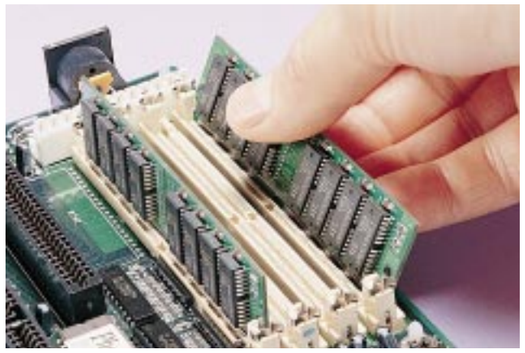
2. Take a moment to examine your new motherboard and read through its (no doubt sparse) manual. Check whether there's anything important of which you should be aware.

3. Most motherboard manuals are invariably terse and techie but you should try to identify the positions of important components and jumpers.

4. If the processor is supplied loose, fit it into the ZIF socket on the motherboard:
- lift the socket lever clamp;
- orientate the CPU so that the bevelled corner on the processor (a.k.a. Pin 1) aligns with Pin 1 on the ZIF socket; and
- lower the CPU in, then lower the lever to clamp the chip into its socket.

Most modern motherboards support a variety of processors from Intel, AMD and Cyrix, and you normally have to move a fair number of jumpers to configure the motherboard for the particular type of processor you're using.

You'll also have to configure the motherboard for the clock speed of the CPU. Many Pentium motherboards are festooned with these tiny black jumpers which are often poorly laid out (from a point of view of ease of use). Mercifully, though, jumpers are on the way out and the latest motherboards are entirely software-configurable from the CMOS Setup menu.



Fitting a SIMM

While access to the motherboard is so easy, take the opportunity to fit the SIMM memory. You will probably have planned to buy EDO SIMMs but if both your motherboard chipset and your budget support it, consider buying Synchronous DRAM (SDRAM) instead — it's a tad faster than EDO.

1. SIMMs are notched at one end to prevent them being inserted incorrectly in their sockets. Find the notched end and locate the

p256 ➢

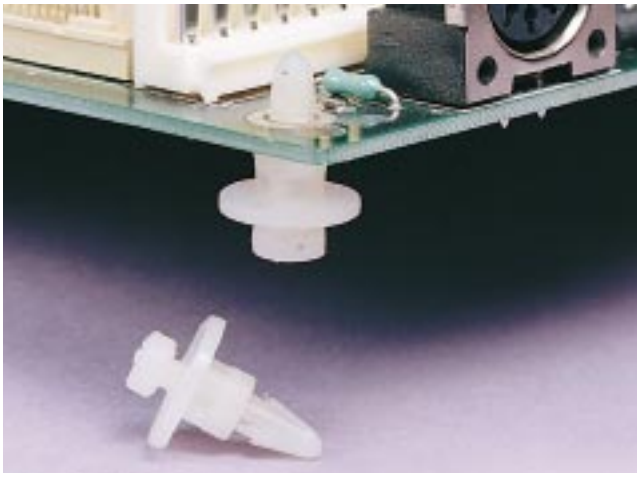
corresponding key in the SIMM socket.
2. Insert the SIMM module, at a shallow angle, into the first SIMM socket (they'll be numbered), then gradually rotate it until it's vertical and the side clips have snapped into place.
3. Do the same for the second SIMM (they have to be fitted in pairs).

Fit the stand-offs

Our final task for this month is to fit the plastic stand-offs to the motherboard. These plastic legs both secure the motherboard to the system unit and insulate it.

• The motherboard will have a number of holes through which the plastic stand-offs are pushed. These stand-offs then locate in tapered "key-holes" in the floor of the system unit. The problem here is that the two sets of holes seldom match up exactly and there will probably be more holes in the system unit base than there are on the motherboard itself.

• At this point you have to carefully work out which holes in the motherboard match up with the corresponding holes in the base of the system unit and *only* fit stand-offs in these holes.



■ **Next month:** We'll actually fit the motherboard and the rest of the peripherals.

Motherboard check list

Your minimum motherboard specification should look something like this:

- At least a 166MHz Pentium MMX (or equivalent) CPU.
- At least four SIMM slots. Maybe one DIMM slot.
- At least 256Kb of pipeline burst-mode secondary cache memory.
- A Triton 430HX or 430VX chipset.
- At least three PCI and four ISA slots.
- On-board I/O (e.g. EIDE, floppy, serial and parallel ports).
- A PnP BIOS of reputable brand such as AMI, Award, MR or Phoenix.

You should be able to get a motherboard to this specification for about £275 (ex VAT). Don't forget the memory: at the time of writing, 16Mb SIMMs cost about £55 and you'll need a pair of them, making £110 (ex VAT).

and take-up of 3D games has been hindered by Microsoft's dilly-dallying over Windows 95 graphics standards. Ordinary DOS games, which are always looking for maximum performance, access the graphics hardware directly; something that was *verboten* under Windows 3.1x. As a result, most DOS games couldn't run under Windows, or if they did, ran so slowly as to make them unplayable.

That settles it — it's DirectX

After much to-ing and fro-ing, Microsoft finally settled on DirectX, a video standard which permitted games to run under Windows 95. This standard embraces DirectDraw, DirectVideo, DirectSound and Direct3D, among others. All are supposed to simplify and speed operating-system access to hardware devices by providing direct access with as little driver overhead as possible.

Most 3D (and 2D) cards now ship with DirectX drivers but it's an emerging technology and you should check Microsoft's web site, which can be found at www.microsoft.com, to see whether updates are available.

DirectX and especially Direct3D are important because until recently games were, in the absence of a common 3D standard, specific to a particular graphics accelerator. Once Direct3D becomes ubiquitous, you'll be able to play any Direct3D game on any 3D graphics card.

Games developers will no longer have to account for what 3D acceleration hardware you might possess, and 3D acceleration hardware vendors will not have to worry about what games you have. So you should check for Direct 3D support when making your choice of graphics accelerator.

3D hardware

So what 3D hardware is available? The most popular 3D accelerators are based around S3's ViRGE and ViRGE/VX chipsets; cards like the entry-level Diamond Stealth 3D. As well as offering 3D rendering, they can all be hooked up to S3's Scenic/MX2 hardware MPEG decoder and, potentially, to other multimedia components via S3's Scenic Highway local-peripheral bus.

Because 3D cards deal with a third dimension, they require far more memory than a 2D card and you should be thinking about 4Mb or maybe even 8Mb of display memory for these.

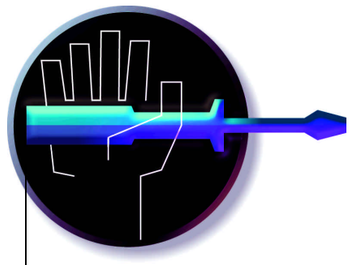
Then there are the dedicated 3D processors, like the VideoLogic Apocalypse 3D or the Diamond Multimedia Monster cards. These work in conjunction with existing 2D cards to deliver high-quality 3D graphics. They add realism to 3D objects by using transparency effects, lifelike shadows, shading, fogging and search-lighting. These cards are fast, too, because they perform the complex 3D calculations on-card and don't bog-down the CPU. Priced at less than £150, these 3D add-ons are worthy of an appearance on your shortlist.

Back Issues

See page 12 or this month's *PCW* CD-ROM for details of how to obtain Part I of this Workshop, which appeared in the June 1997 issue.

PCW Contact

Roger Gann can be contacted by post c/o *PCW* at the usual address or via email at rgann@mcgilivray.win-uk.net.



The home-brewed PC

Why not try building your own PC? It's fun, and you can learn a lot about what makes computers tick while you're tinkering. Roger Gann starts off with a list of the bits and pieces you'll need.

In my Hands On Hardware column over the past year or so, I've looked at the many ways you can upgrade your PC's hardware, from adding a SCSI host adaptor to replacing your motherboard. Over the next four months I'll be taking you through a much larger upgrade project: building your own PC from scratch. I'll take you through the A to Z of building a decent quality entry-level Pentium PC, complete with an Enhanced IDE hard disk and a CD-ROM drive.

Let's face it, assembling your own PC isn't a popular pastime, and doesn't begin to rival gardening or fishing as a hobby. But building a DIY PC can be a lot of fun and it's very instructive: at the end of it you'll have a much clearer understanding of PC internals and how they work together. You'll also be better placed to troubleshoot hardware problems in future. By building it yourself, you can opt for the piecemeal approach, spreading the purchase of the components over a period of time. You'll have the benefit of a PC built to your exact specification and to your standards of workmanship.

That's the good news. There's bad news, too. For a start, most PC assemblers already build PCs to your precise specification. More importantly, you're unlikely to beat them on price by opting for the DIY PC. You'll be buying components individually, while PC manufacturers will buy *en masse* so their unit costs will be much lower. Factor in the time spent building it, add up the cost of bundled software so often included with PCs sold today, and you'll see that, overall, it will be cheaper to buy a complete PC. So there are no savings to be had from building it yourself: but we're not doing it for the money, are we? Then there's the warranty. Escom owners may have a view on the value of service warranties, but any warranty is better than no warranty; and when you brew your own PC, you're on your own.

Degrees of difficulty

To be honest, building a PC isn't for everyone, and if you're the technically timid sort that finds fitting a graphics card an ordeal, you should stop reading here and quickly turn to the next feature, as building a PC from scratch would be a big mistake.

However, if you've attempted any of the more adventurous upgrades I've covered over the months, such as swapping a motherboard or fitting a hard disk, building a PC really isn't substantially any more complex than this. Rocket science isn't involved: there are just more bits to fit and a tad more preparation and planning. Once you've started, will it take ages to complete? No, is the short answer. With all the bits in front of you, a simple PC can be assembled in less than an hour.

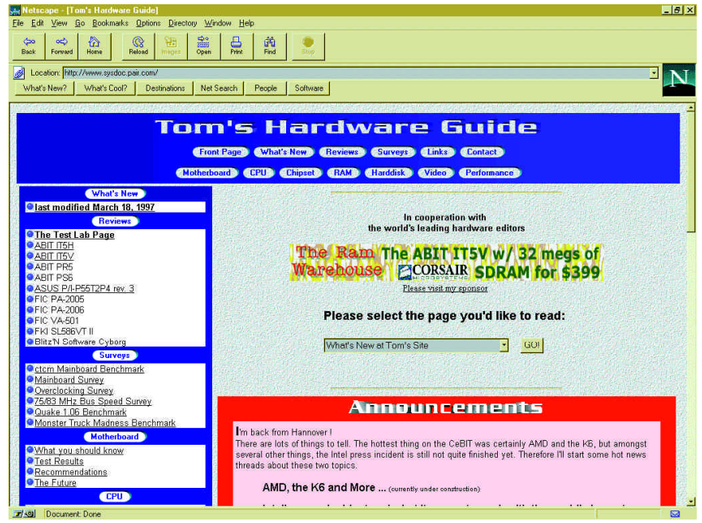
Sadly, there seems to be little in the way of books on the subject. Incredible as it may seem, until recently there were no books at all in the Computer Manuals catalogue on this subject. Now there's one, called, funnily enough, *Build Your Own PC*. But no fear: over the next four months I'll be giving you the low-down on the whole process, from start to finish.

Choosing components

Choosing some of the most important components of your home-brewed PC will be particularly tough. OK, you'll have the reviews at the front of *PCW* to guide you when you come to choose hard disks, graphics accelerators, monitors and the like. But you'll be on your own when it comes to such things as motherboards and cases, as these are invariably ignored when it comes to product reviews in any computer magazine. I guess cases are just too dull and motherboards too anonymous and unbranded to bother with.

Assuming you've got no spare hardware lying around, you'll need the following:

- 3.5in floppy drive (£15)
- 2Gb hard disk (EIDE) (£165)
- Eight-speed CD-ROM drive (£65)
- 72-pin EDO or SDRAM SIMM memory (16Mb) (£50)
- PCI graphics accelerator, e.g. Matrox Mystique or VideoLogic GrafixStar 600 (£100/£80)
- 15in 0.28mm dot-pitch SVGA display (£250)
- 102-key AT Windows 95 keyboard (£25)
- Mouse (£20)
- System case with power supply (£50)
- Motherboard and P166 CPU (£300)
- Any software, i.e. Windows 95



Check out Tom's Hardware Guide for some seriously detailed hardware info

Choosing the system case

Between them, the case and the motherboard amount to the foundations of your PC, so it pays to thoroughly check out what's available. These parts may look as alike as peas in a pod in the ads, but believe me, they aren't. With system cases, it's important to actually see the case and open it up. This way, you can judge for yourself just how easy it is to use and whether it meets your needs. If possible, don't buy "blind" (off-the-page); buy in person. Case ergonomics should play a big part in your choice but you'll probably only discover its shortcomings after you've bought it. An example of this is my PC's tower case. I had to remove the entire motherboard just to be able to undo a pair of bolts in order to swap

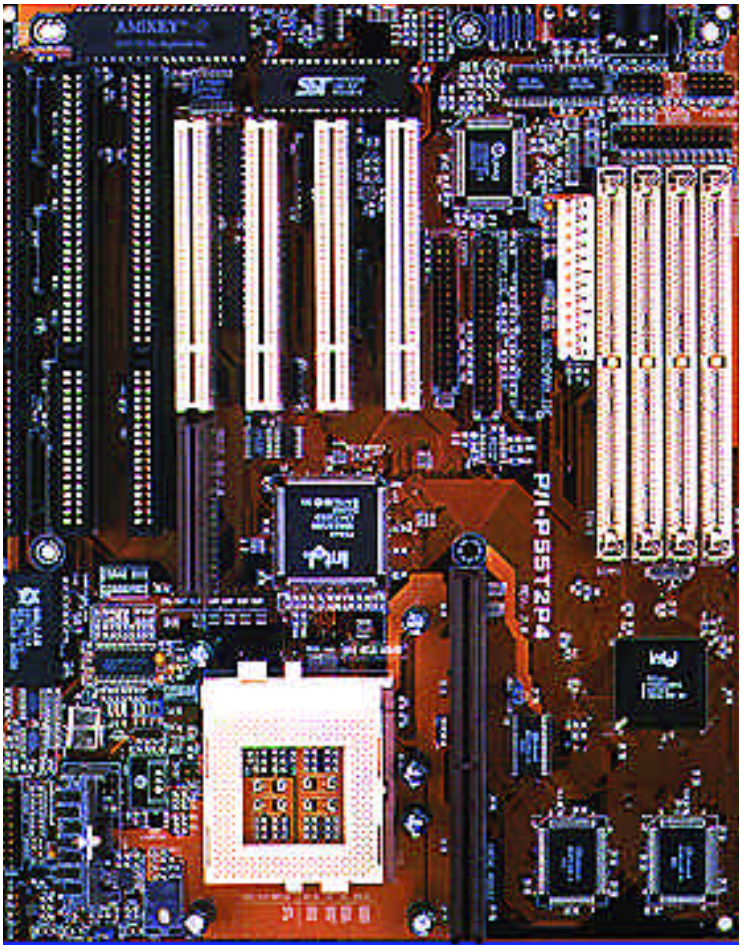
p240 ➢

a sickly hard-disk drive.

Flicking through the ads reveals that a good selection of system cases are available. However, they're all much of a muchness and fall into four broad types: normal desktop, slimline desktop, compact desktop/mini tower, and full tower. Prices start at about £35 and top out at about £85 for a full tower, although if you want a case that complies with the Euro CE safety standard you can add roughly a tenner to these prices. Wherever possible, try and get the biggest system unit possible. Not only will this give you maximum expansion potential, but it also makes access to the internal components easier. Unless you specifically want a slimline case, go for the larger case. In the end, the final decision boils down to expansion potential: if you want to fit a lot of drives, buy a tower case. If not, buy a desktop or mini-tower case. Don't forget, you get what you pay for: pay peanuts for a case, and you'll get something of flimsy construction and awkward to use.

All "Baby AT" motherboards will fit standard cases, but watch that the slimline case doesn't require an expansion card riser or "tree" so that cards can be fitted horizontally. If it does, be sure to select a motherboard that has these features and, most importantly, fits the case. And if you're fitting an ATX-style motherboard, make sure you buy a case designed to take ATX form-factor boards.

Most cases come with a 150W or 200W power-supply unit (PSU) as standard, but this might be a bit light for a well-stocked full tower. Ask how many power connectors the PSU has (the more the merrier), and what sort they are. It should have two types: the standard Molex, and the mini power connector. Most PSUs have power leads for only four peripherals, but try and find one with six. Ask whether it comes complete with all the fixings and accessories, things like printed circuit-board (PCB) supports, mounting bolts and drive rails. Consider at this point whether you want to fit a removable hard-disk tray.



The Asus P/I-P55T2P4 motherboard recommended in Tom's Hardware Guide pushes the new Intel 430HX chipset to its full potential

Choosing the right motherboard

If choosing a simple thing like a system case isn't straightforward, choosing a motherboard to go inside it is tougher still. The motherboard is, of course, the heart of your PC and, thus, is a fairly technical piece of kit. They are mainly sold as virtually unbranded, generic devices, each one near enough identical to its neighbour. There are such things as motherboard "best buys", but in the absence of proper product reviews who's to know? Sadly, there's no comfort to be derived from relying upon brand names to guide you. With the exception of Intel, you probably won't have heard of the major motherboard players: people like Asus, Abit, ECS/EliteGroup, Gigabyte, Micronics and SuperMicro.

So what should you be looking for in the ideal Pentium motherboard? Well, there's a veritable laundry list of desirable features that should appear on your checklist. There are the obvious ones like the form factor (Baby AT or ATX), the number of PCI and ISA slots, and the nature of the on-board I/O it has (EIDE, fast serial and enhanced parallel ports). There are other less obvious but just as important features. These include having a Flash BIOS (which permits software upgrading), the number of SIMM slots (usually four but sometimes eight), and is there a DIMM slot? Does it have an IR port or support for Universal Serial Bus?

On the techie side, you should check the board supports a wide variety of processors, including Cyrix and AMD CPUs. It should have an adjustable CPU voltage regulator (Standard/VRE/MMX), support EDO and SDRAM (particularly the latter) and should have a modern, up-to-date chipset. If it's an Intel chipset, it ought to be a Triton 430VX, HX, or the just-released TX.

So which motherboard is best? Luckily, there is an excellent web site that conducts benchmarking tests on motherboards which you can refer to. Tom's Hardware Guide (www.sysdoc.pair.com) contains an absolute goldmine of technical info plus hints and tips about PC hardware, and is well worth a visit. There you'll find various motherboard "Top Tens". For example, for 430HX boards, Tom Pabst recommends the Asus P/I-P55T2P4 and Abit IT5H boards, and for Triton 430VX boards, the Abit IT5V. Boards like these not only offer jumperless "Soft Menu" configuration but can also run the bus at 75MHz or even 83MHz (as opposed to 66MHz) for the latest generation of fast CPUs. It's worth searching Yahoo on the keyword "motherboards": you'll find all the motherboard manufacturers with an internet presence.

And that's all for this month. In part two, I'll be looking at what you'll need if you want to build the ultimate games platform, plus the first step, installing the motherboard. ■

Getting automated

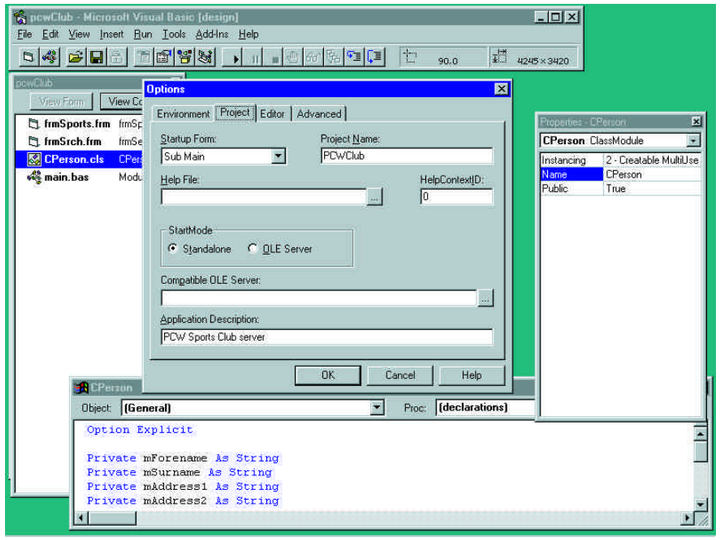
Tim Anderson does clever stuff with automation servers in the final part of the workshop. Plus, polishing the Sports Club database and adding some essential new functions.

Last month's workshop demonstrated how a Visual Basic class can be plucked out of the standalone version of VB and, with care, planted into Microsoft Office as a Visual Basic for Applications class. That is one way of re-using code, but an even better approach is to create objects that can communicate with any number of different applications without the need to recompile. With the rise of PC networks, and now the internet, this kind of software is the way of the future.

Under Windows, the way to achieve this is by using Microsoft's Component Object Model, or COM. This is the technology beneath OLE and ActiveX, and Visual Basic programmers can use it without knowing the detail of how it all works. For example, what if the PCW Sports Club wants to get at membership details not only via Word, but also from other programs like accounts and desktop publishing packages?

A key part of the Sports Club application is the CPerson class module which describes a club member. By making a few changes, that class module can become an automation object which exposes its properties and methods to other applications. The steps are as follows:

1. Open the CPerson module and press F4 to reveal the class properties. Set the Instancing property to CreateTable MultiUse, and the Public property to True.
2. Add a module to the project using Insert - Module. In the module, create a Sub Main.
3. On the Tools menu, in Options - Project, change the project name to PCWClub, and put a few words of description in the Application Description field. Finally, change the Startup Form to Sub Main.



Two key steps in creating an automation server are: first, the properties for the class to be exposed; and second, the project options

The project name must be unique to your automation server. The full class name of objects in your server will be of the form PCWClub.MyClass.

By taking these simple steps, you have created an automation server that lets other applications create and control objects of the CPerson class. All that remains is to register the class in the system registry. If you run the application in VB's development environment, it will be registered temporarily. If you build an .EXE or OLE DLL, it will be registered permanently. Then you can write code like this in Excel:

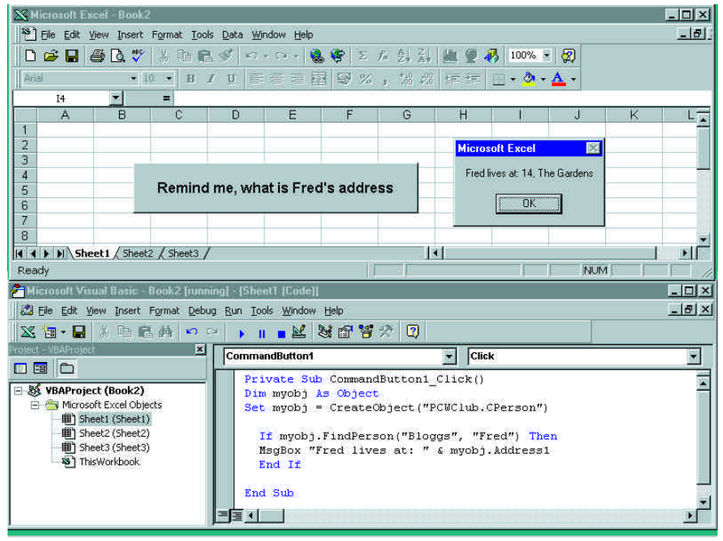
```
Dim myobj As Object
Set myobj = CreateObject
("PCWClub.CPerson")

  If myobj.FindPerson("Bloggs",
"Fred") Then
MsgBox "Fred lives at: " &
myobj.Address1
End If
```

Although applications such as Excel function both as automation servers and standalone, most VB applications will be one or the other. Often, VB automation servers have no user interface, since this is provided by the client application. The workshop example, though, is designed to work in both guises. The trick is to use Sub Main to detect whether the application is running standalone or as an automation server. Here is the code:

```
Sub Main()

If App.StartMode <>
vbSModeAutomation Then
frmSearch.Show
End If

End Sub
```

You may wonder what Sub Main does when running as a server? The answer is, nothing at all. The only thing the server application does is to expose its classes so other applications can create and control objects. For testing, you can simulate this mode by setting the startmode in Project Options to OLE Server. Run the application and then minimise VB. Next, run another instance of VB, open the References dialog and check the PCW Sports Club server. Now you can test the server by creating objects of the CPerson class.

If you have run a compiled VB automation server such as PCWCLUB.EXE, it will be entered permanently in the system registry. Once you have finished testing, it is good practice to remove it. You can do so by running it from a command line with the /UNREGSERVER parameter. DLLs are unregistered using REGSVR32.EXE which is found in the System directory. Run it without parameters to see the switches.



Excel is able to get at Sports Club details by using a VB automation server

Automation servers are powerful but do present some new programming challenges. The section of the Visual Basic manual called "Creating OLE Servers" is essential reading.

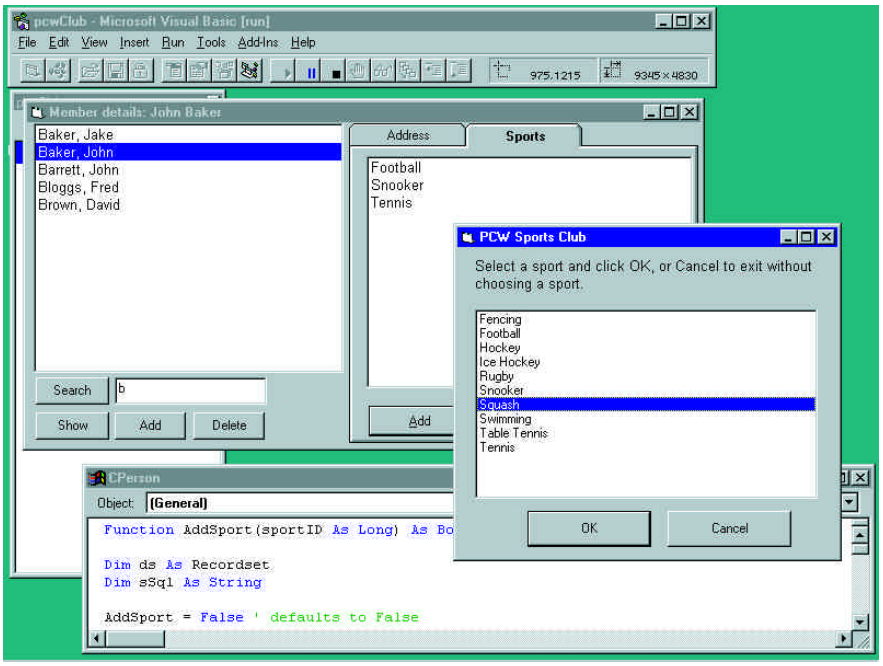
Adding the essentials

The Sports Club database is also used as a standalone application, and the version in last month's workshop is lacking some essential features. First, there is no way to add or delete members; and second, you cannot add or remove sports from the list which applies to each member.

The thinking behind the design of this simple application is that interface code belongs in the main form, while database code belongs in the CPerson class so that it can be used in other applications or as an automation server. The natural approach is to create new public methods for CPerson that give this new functionality. For example, here is code to add a new member:

```
Function CreateNew(sSurname As
String) As Long
' creates a new person in the
database
Dim lId As Long
myRecordSet.AddNew
myRecordSet!surname = sSurname
lId = myRecordSet!ID
myRecordSet.Update
Me.load (lId)
End Function
```

p260



Adding a sport to a member's list of interests is achieved via a simple dialog

The ID field is a counter, which means that the JET database handles the business of ensuring that the new member has a unique number. There is an issue, though, about how to cope with users who change their mind.

What if someone starts to create a new member, and then wants to back out and leave things as they were? One possibility is to call the AddNew method, but not to call Update until the user confirms the action. Unfortunately, bullet-proofing the application so that Update is only called after AddNew or Edit is prone to error. The easier approach is to minimise the time when JET has unsaved changes in its copy buffer. In this application, clicking the Add button creates a new member with the surname "Unnamed". If the user wants to cancel the addition, it is just a matter of clicking Delete.

The DeletePerson method is a little more involved. The problem is that there may be other records, in the SPORTLINK table, which refer to the member being deleted. To maintain data integrity, these records also need to be removed. Database objects have an Execute method which is an ideal solution. Execute takes an SQL command and applies it to the database. For example:

```
sSql = "DELETE * from SPORTLINK
where SPORTLINK.MEMBERID = " &
Str$(lId)
myDB.Execute sSql, dbFailOnError
```

The code at form level also has some work to do. When a member is deleted, the name must be removed from the list currently displayed, and the other fields on the form updated as necessary.

To make sense of adding sports to a member's list of interests, you need to throw a dialog listing the available sports. The dialog has a SportID property. To add a sport, the application takes these steps:

1. Show the Sports dialog modally, which means the user must either choose a sport, or cancel, before continuing.
2. When the OK button is clicked, the Sports dialog sets the SportID property to the currently selected sport.
3. Next, the program calls CPerson's AddSport method, passing the SportID as a parameter. AddSport creates a dynaset-type recordset which looks for records in the SPORTLINK table that match this member with the chosen sport. If the dynaset is empty, AddSport adds the required record. If it is not empty, AddSport reports that the member is already linked to that sport.
4. Finally, the program updates the form with the new list of sports.

Finishing touches

There is plenty more work to do in improving the Sports Club application. One professional touch is to enable and disable buttons according to whether or not they are applicable. For example, when no sports are listed, the Remove sport button should be disabled. Next, you can add keyboard shortcuts for mouse-free typing.

Another important area is error-handling, to prevent the program from crashing and to show the user informative messages when things go wrong. For instance, the database could become corrupted.

Finally, there is the issue of multiple users and what happens when two people try to update a record at the same time.

■ *All the code for this month's workshop is on the cover CD. And see Hands On Visual Programming (p301) for answers to queries concerning this workshop and other Visual Basic problems.*

Beating the OLE jargon

OLE has lots of strange jargon. and here are two examples that can cause confusion. Mastering these issues is important to make good use of the technology.

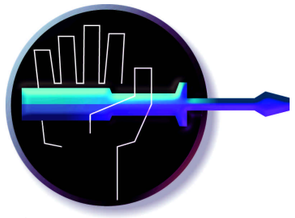
First, you will see references to in-process and out-of-process servers. In-process servers are DLLs which run in the same address space as the calling application, whereas out-of-process servers run in their own address space. This is a decision you take when building a VB executable. In-process servers have substantially better performance but introduce more programming restrictions.

Second, there is the matter of early or late binding. Binding is the process of locating the properties and methods which the client application calls. If you use variables declared as Object in the client application, then these identifiers are not resolved until runtime. This is called late binding. On the other hand, if you use an OLE-type library to resolve these identifiers at compile time, the code will execute faster. This is early binding. To use early binding in Visual Basic, open the Tools - References dialog and check the type library required. Then, declare variables of the specific class required, rather than the generic Object. This is much faster and also enables you to detect errors in parameters, properties or method names when the application is compiled. Another bonus is that you can use an object browser to inspect the interface of available classes.

Naturally, the best performance combines both techniques — that is, in-process servers called with early binding.

PCW Contacts

Tim Anderson welcomes your comments and queries. Write to the usual *PCW* address, or email freer@cix.co.uk.



First class letters

With VB you can use Word to create letters that virtually write themselves. Tim Anderson shows you how. Plus, how to delegate in Visual Basic to achieve the benefits of inheritance.

The "*PCW* Sports Club" is continually sending letters, reminders of coming events, subscription invoices, sympathy for broken bones and the like. The secretary has been running the Visual Basic application to look up the address and then using **Alt-Tab** to switch back and forth from Word while she copies it across. It is time to make her life a bit easier.

The first thought was to use VB's **Clipboard** object to copy addresses. This is easy: just add a **CopyAddress** method to the **CPerson** class, as in Listing 1.

But Windows can do better than that. It is possible to automate far more of the process of getting addresses into Word. Word has a mail-merge wizard that works fine for bulk mailings, but for *ad hoc* letters a custom solution is needed.

Here, I will show you how to create a Word letter wizard for the sports club (see screenshots, Figs1-4). The wizard is for Word 97, since earlier versions do not support Visual Basic. (As an aside, it is possible to do something similar in earlier versions, using the WODBC.WLL Word add-in and getting at data through ODBC. Another possibility is to automate the WordBasic object from a VB application. But Word 97 makes it easier.)

The plan is to create a Word macro using Visual Basic for Applications, accessing the same SPORTS.MDB database.

Because this tutorial is based on VB 4.0, you cannot import the form in Word. The good news, though, is that the **CPerson** class module can be reused, as is. The procedure is as follows:

1. In Word, open the Visual Basic editor. Choose **Tools, References**, and check the Microsoft DAO 3.0 (or higher) object library.

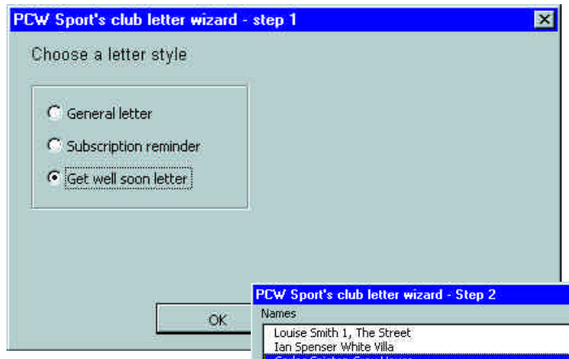


Fig 1 (left) The letter wizard begins by asking which template is needed. You can easily add further options

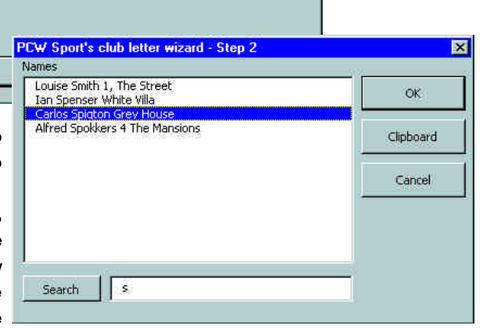
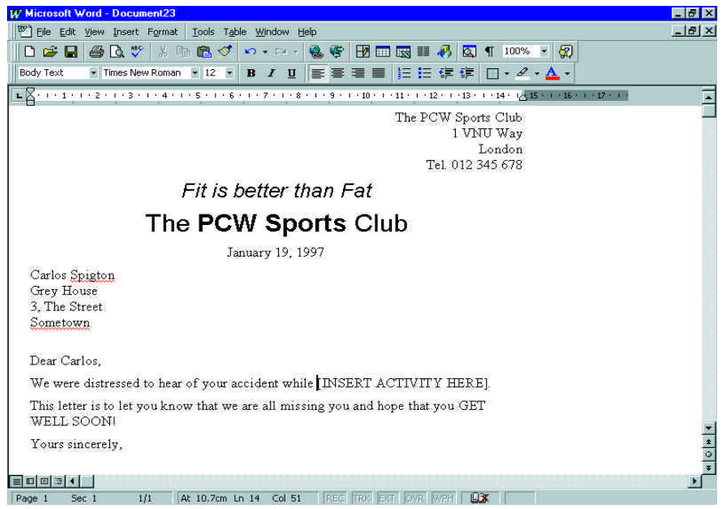


Fig 2 (right) The next stage is to choose a name to write to

Fig 3 (below) In the third step, the wizard has placed the address details into a new document based on the chosen template



Listing 1

```
Sub CopyAddress( )
‘ copies address to Windows clipboard
Dim sAddress As String
Dim cr As String * 2 ‘ fixed-length

cr = Chr$(13) & Chr$(10)

If mForename <> “” Then
sAddress = mForename & “ “ & mSurname & cr
Else
sAddress = mSurname & cr
End If

If mAddress1 <> “” Then
sAddress = sAddress & mAddress1 & cr
End If

...

Clipboard.SetText sAddress, vbCFText
End Sub
```

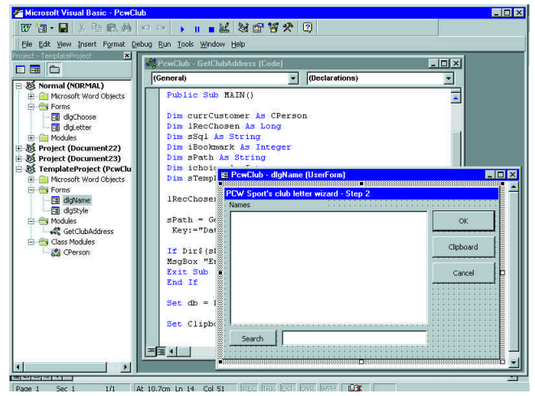


Fig 4 Editing the VBA macro from Word is very like working with standalone VB

This enables Word to use the same data access objects as VB 4.0.

2. Insert a new module into the Normal project. This means the macro will be stored in NORMAL.DOT. Call the macro **GetClubAddress** and give it a **Sub Main**.

3. Insert two new userforms. These will be steps one and two of the letter wizard.

4. Name the first userform **dlgStyle**, and put two or more option buttons on it, along with **OK** and **Cancel** buttons. Give the form a **Choice** property using a memory variable and property procedures. This is for choosing a letter template.

5. Name the second userform **dlgName**, and put on it a listbox, an editbox and three command buttons. This is for choosing a name for the letter.

6. In the Declarations section of **GetClubAddress**, declare a public database object. For the example code, I have also declared some convenient constants. Then in **Sub Main**, open the SPORTS.MDB database using code like:

```
Set db = DAO.OpenDatabase(sPath &
 “\SPORTS.MDB”)
```

sPath is a variable to store the path to the database file. (See below for how to get this path from the system registry.) **Sub Main** also creates a new **CPerson** object.

The **Main** procedure continues by opening **dlgStyle** to obtain a choice of template, and then **dlgName** to get the ID of name in the **Members** database. At each point, the user has an option to cancel. The code for the **dlgName** dialog is almost the same as that used in the main VB 4.0 application, the main difference being that VBA has no data control so you have to create a recordset in code. When a member ID has been retrieved, the record is loaded into the **CPerson** object.

7. The final step is to start a new document based on the chosen template. The templates must be pre-designed with bookmarks where the name and address information is needed. The wizard finishes by inserting the fields in the bookmark positions and then exits. Controlling Word from VB in this way is not difficult using Word's new object model. For example: p258 ➢

Delegating your inheritance

■ A common criticism of Visual Basic is that it doesn't support inheritance. If all your programming has been done in Visual Basic, which is probably true of the majority of VB programmers, this may not mean much to you. Fortunately, it's easy to explain. A class, both in VB and other object-orientated languages, defines an object. In VB, every class starts from scratch without any properties or methods. By contrast, C++, as an example, lets you begin a class definition like this:

```
class monkey : public animal
```

The result is that the monkey class inherits the properties and methods of the animal class. The monkey class just needs to add specialised code that describes monkeys; the generic animal code comes for free.

Although VB does not support inheritance, there are other ways of achieving some of the benefits. It is possible to contain one class within another. Then you can implement properties and methods of the parent class by calling the properties and methods of the contained class. This is called delegation, and the properties and methods of a class are called its interface. For example, the tutorial application has a CPerson class. Imagine you wanted to create a CEmployee class which used the properties and methods of CPerson. Here is how you can do it:

1. Insert a new class module and set its name property to CEmployee.
2. In the declarations section, put:

```
Private m_person As CPerson
Private m_wage As Currency
```

3. In the initialise section put:

```
Set m_person = New CPerson
```

4. Create a CEmployee interface that calls the CPerson interface. For example:

```
Public Property Get surname( ) As
 String
surname = m_person.surname
End Property
```

5. Add new properties and methods specific to CEmployee. For instance, you must expose the wage property.

The fourth step (*above*) is tedious, but beats re-coding all the functionality of CEmployee in CPerson. It could be automated by a VB Wizard. In Visual Basic 5.0 this approach to object-orientation is built into the language, with a new Implements keyword which guarantees that all the methods of the contained class are implemented by the outer class. You can implement the interface of any ActiveX automation server. Finally, there is nothing to stop you implementing several interfaces in a single class.

Delegation works, but it is neither as intuitive nor as elegant as traditional inheritance. For the moment, though, this is the VB way. It ties in with ActiveX, the component model which is becoming more powerful and pervasive as Windows evolves. VB may not be the fastest or most thoroughly object-orientated language out there, but Microsoft does ensure that it stays up to date with the latest ActiveX developments.

```
Documents.Add (sTempLate) ‘ starts
a new document based on the given
template

ActiveDocument.Bookmarks(“name”).
Select ‘ sets cursor to the “name”
bookmark in the new document

Selection.InsertAfter Trim
(currCustomer.forename & “ “ _
& currCustomer.surname) ‘ inserts
text at the cursor position
```

Problem-solving

There are a few things to notice about this joint Visual Basic and Word project. Although Word VBA is downward compatible with VB 4.0, there are some objects which are available in VB but not VBA. One example is the global App object which, in Word, is the Application object.

The original **CPerson** class used **App.Path** to discover the location of SPORTS.MDB. This strategy fails in any case, when the code runs in other applications. A better idea is to use a registry entry, using VB's **GetSetting** command. The registry entry is created by the main VB 4.0 application when it first runs. This way, the data can easily be found by any Windows application.

Another catch is that VBA has no Clipboard object, so CPerson's CopyAddress method does not compile in Word. The workaround is to declare a public Clipboard variable as a **DataObject**: VBA's private version of the clipboard. To demonstrate, there is a Clipboard button on the **dlgNames** form which uses the DataObject's **PutInClipboard** method to transfer text to the read clipboard.

Enhancing the wizard

There are plenty of ways you can improve on the Letter Wizard. For instance, you can add database fields for things like job title and salutation. You could increase the range of templates on offer. For the subscription template, you could write code to check a person's outstanding balance and insert the amount into the letter. By adding the bulk of the code to a shared class module like CPerson, you can easily reuse it in VB 4.0 or in other VBA applications such as Excel.

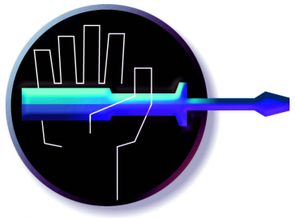
Installing the example code from the *PCW* CD

When you unpack the tutorial code from our cover-mounted CD, you will find a VB 4.0 project and a Word 97 template. To install the example code, copy PCWCLUB.DOT into your Word templates directory. Then start a new document based on this template. If you then choose Tools, Macro, Visual Basic Editor, you will find the example macros. Choose Tools, Macro, Run, to run the macro. You can also copy the macros into NORMAL.DOT if you want, by using Tools, Templates and Add-ins, Organizer. Finally, the macro will not run without a registry setting for the data path. To create this setting, run PCWCLUB.EXE.

■ *Next month: Back to native Visual Basic for the final stage in the PCW Sports Club application.*

PCW Contact

Tim Anderson welcomes your comments and queries. Write to the usual *PCW* address, or email pcw@vnu.co.uk.



A sporting chance

Visual Basic gives the mythical *PCW* sports club the database treatment in the second of our workshops, conducted by Tim Anderson. He also studies the life of a VB object.

Last month's workshop showed how to create a simple database application for the *PCW* sports club. It was a flat-file database, which means all the data was stored in a single table, like a card-index. At the sports club, though, it is important to know which sports a member has signed up for. A member can sign up for any number of sports, and each sport is played by any number of members. This is a classic many-to-many relationship, but it is not always obvious how best to store this kind of information.

One strategy would be to add several fields to the table of members, for Sport1, Sport2, Sport3, etc. Another possibility is a notes field with the sports entered line by line. Both these ideas are fatally flawed. Although they seem easy, they are actually inefficient and inflexible. For example, what happens when you want a list of all the footballers? You would end up with a horrible keyword search and probably get inaccurate results.

The correct approach is to analyse the data into three tables. The first one is the table of members. Next there is a table of sports, which for the moment has just two fields, Sport and ID. The third table records which member belongs to which sport. SportLink again has two fields, MemberID and SportID. If you view the table on its own it will look like a meaningless string of numbers, but in combination with the other tables it makes sense. Later you might want to add other fields to SportLink, perhaps a Role field which contains information like "Goalkeeper" or "Captain". It is important to grasp this principle, which is a great way to store all kinds of data.

The main form needs adapting to display this new information. Since a member may sign up for any number of sports, these are best displayed in a listbox control. To keep the form from getting cluttered, a good tip is to use a tab control as well. Visual Basic 4.0 comes with two: a TabStrip which is for Windows 95 only, and the SSTab which comes as both a 16-bit and 32-bit OCX. In this example SSTab is used. The tabs work at design time, making it easy to lay out the form. When a tab is selected, controls placed on it belong to that tab. Controls placed on the form itself will show through all the tabs. Fig 1 shows buttons for adding and removing members from particular sports, but these are not yet enabled.

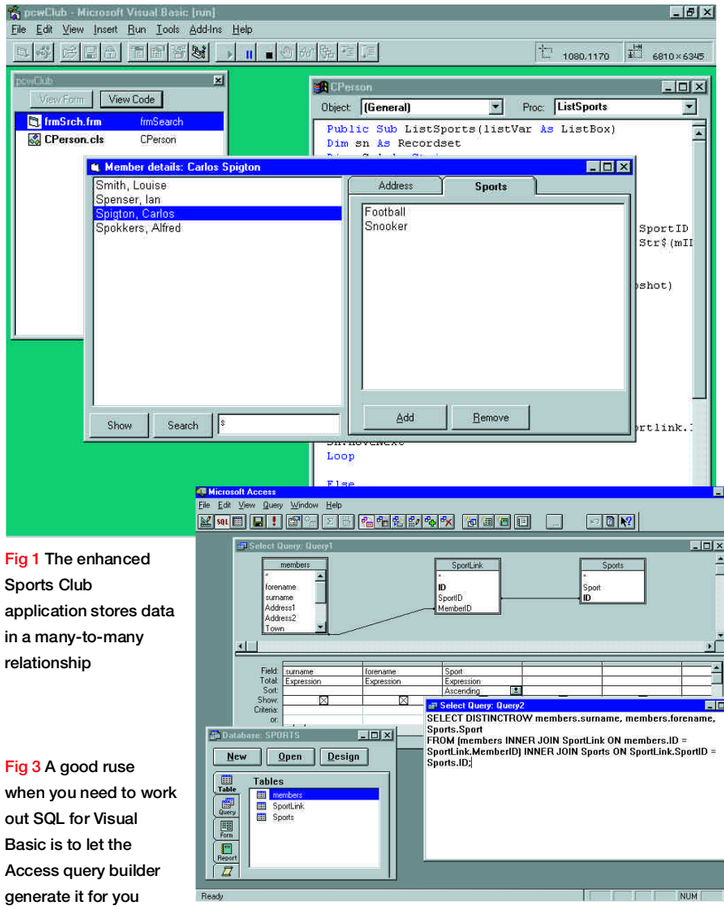


Fig 1 The enhanced Sports Club application stores data in a many-to-many relationship

Fig 3 A good ruse when you need to work out SQL for Visual Basic is to let the Access query builder generate it for you

The next step is to write code to display the list of sports for each member. One idea is to add a ListSports method to the CPerson class. The ListSports method takes a listbox control as a parameter. It searches the database to get the list of sports and adds them to the listbox. Doing it this way means that if a list of sports is needed at some other point in the application, it will not be necessary to rewrite the code. All you need do is to supply the ListSports method with an available listbox. The code for CPerson.ListSports is in Fig 2.

Much of this code is similar to that used last month for searching the members table. The main difference is that the SQL query for extracting data from two tables is more complex. If you followed *PCW*'s recent SQL workshop, you will have no problem. If not, notice that the SQL string includes several sections:

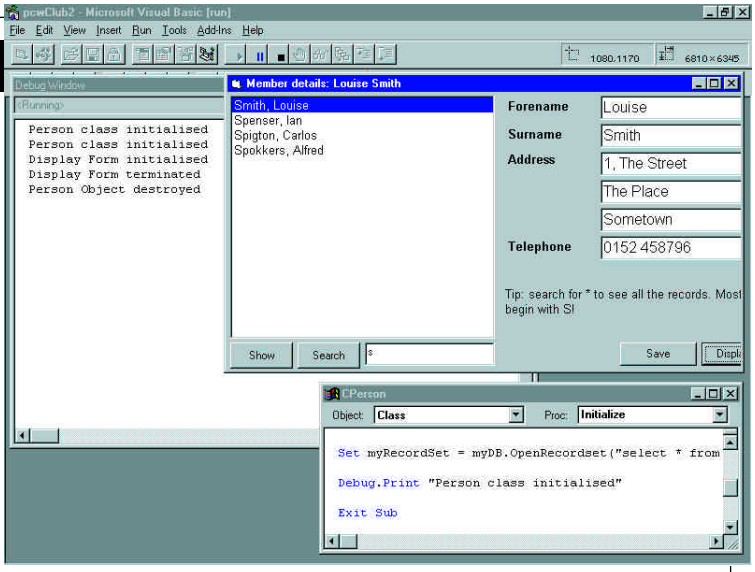
1. Which fields to extract — SELECT * for all fields.
2. How the two tables are linked — the first part of the WHERE clause.

p268 ➢

Fig 2 The code for CPerson.ListSports

```
Public Sub ListSports(listVar As
ListBox)
Dim sn As Recordset
Dim sSql As String

listVar.Clear

' build up the SQL command
sSql = "SELECT * FROM Sports,
Sportlink "
sSql = sSql & " WHERE Sports.ID =
Sportlink.SportID "
sSql = sSql & "AND
Sportlink.MemberID = " & Str$(mID)
sSql = sSql & " ORDER BY
Sports.Sport"

Set sn = myDB.OpenRecordset(sSql,
dbOpenSnapshot)

' now fill the list box
If Not (sn.BOF And sn.EOF) Then
' there are records

sn.MoveFirst

Do While Not sn.EOF
listVar.AddItem (sn!SPORT)
listVar.ItemData(listVar.NewIndex)
 = sn![SPORTLINK.ID]
' the square brackets and table
 name are needed because
' there are two different ID fields
\ in the result set
sn.MoveNext
Loop

Else

listVar.AddItem "None"

End If    End Sub
```

The life of a VB object



Judicious use of Debug.Print can help track the lifetime of VB objects

Introduced in Visual Basic 4.0, class modules are a way to create user-defined objects. For example, the Sports Club application has a CPerson class with properties and methods. These constitute the interface which a person object presents to the application. Whenever your other code has to interact with a person object, it does so through this interface. If the interface stays the same, you can change or improve its implementation (the code which drives those properties and methods) with no danger of breaking the application. If you add to the interface, those new features are available wherever a Person object is referenced.

The first thing to understand is the lifetime of an object. Unlike other variables, you can't simply DIM a CPerson object and then refer to its properties. Objects must be instantiated. For example, this gives an error:

```
Dim Myperson as CPerson
Myperson.surname = "Baxter"
```

Error — object variable not set. Instead you need code like this:

```
Dim Myperson as Cperson
set Myperson = New CPerson
Myperson.surname = "Baxter"
```

Or if you start with:

```
Dim Myperson as New Cperson
```

VB will instantiate the object when first referenced.

The question of instantiation is important because it does not just allow you to start using the object. It fires an event, Initialize. All class modules have this event predefined. It is extremely valuable, since you can do things like setting default values for properties, or opening a link to a database, or instantiating other subsidiary objects as required. Sadly, Initialize cannot take parameters, making it less useful than it should be. There is another, similar event called Terminate, which occurs when the object is destroyed.

But when is the object destroyed? It is destroyed when there is no longer any active reference to it in your code. This feature is designed to make it easy to manage objects, but can get confusing. If you have an object variable declared in a procedure, it goes out of scope and the object is destroyed when the procedure finishes. But if you have assigned the object to another variable which is still in scope, the object is not destroyed: the listing (left) illustrates the point. So far, not too difficult. It's harder when your objects are more ambitious. Perhaps you want a CPerson to have a Display method which creates and shows a form. VB forms are just another kind of class, and the obvious approach would be like this:

```
public sub Display()
Dim myform As New DisplayForm
' ... code to fill the fields
myform.Show
```

But to encapsulate things you will want a Hide method which disposes of the form. That means keeping a reference to the form in the CPerson class, so the DisplayForm variable needs to be scoped to the class. It is likely the form will need to interact with its corresponding CPerson object, so you give the form a Person property. The references start to proliferate, and neither the DisplayForm nor the Person object will be destroyed until the last one goes out of scope or is set to Nothing. If your Hide method was like this:

```
Unload myform
```

that would not destroy the form object. In turn, the form object would prevent the Person object from being destroyed, because it still has an active reference to it. You have to add the line:

```
Set myform = Nothing
```

to clean it up properly. The conclusion is that you need to watch the lifetime of VB objects closely or they could stick around for longer than they are wanted.

Destroy all objects!

```
Dim Myperson As New CPerson
Myperson.surname = "Baxter"
Set FormPerson = Myperson
' assumes FormPerson is scoped to the form
' both now refer to the same object
Set Myperson = Nothing ' Object is NOT destroyed
Set FormPerson = Nothing ' Object is destroyed
```

3. An additional restriction — the second part of the WHERE clause after AND. This ensures that only data for the current member is extracted.
4. An ORDER BY clause to sort the results. Working out SQL acceptable to JET, the VB database engine, can be tricky. A good ploy is to build a query in Access, then cut-and-paste the generated SQL code (Fig 3). Note that often, more than one SQL expression will product the same result, sometimes with performance differences.

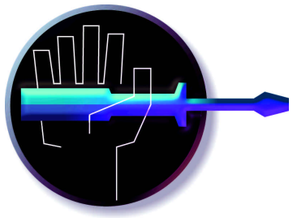
■ *All the code for this month's VB workshop is on this month's cover CD.*

■ *Next month: Visual Basic, inheritance and delegation.*

PCW Contact

Tim Anderson welcomes your comments and queries. Write to the usual *PCW* address, or email freer@cix.co.uk

NEW!



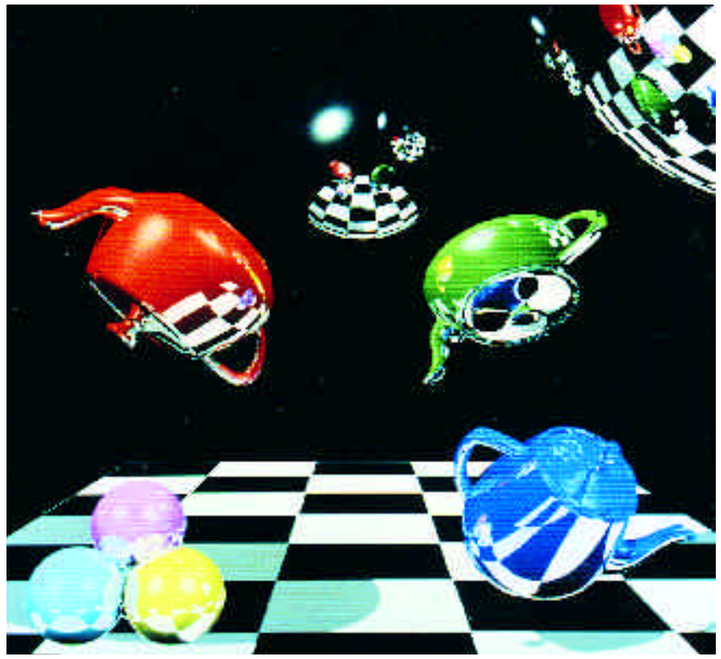
Object of the exercise

Tim Anderson takes you through the first in a four-part teach-in about Visual Basic. You'll learn how to get to grips with VB objects and snap together a powerful database application using a few lines of code.

Like it or loathe it, you can hardly avoid it — Visual Basic is the most popular Windows development language. It is also the macro language of Microsoft Office, and with Microsoft now willing to license it to third parties, VB will more frequently appear in third-party products such as the Visio charting package. So time spent learning Visual Basic (VB) soon repays the effort, giving you program control over many powerful applications.

This workshop will show you how to make the most of VB: including data access, automation of other applications like Word and Excel, and using ActiveX components for rapid application development. And along the way you will build a useful application. The software is for managing a sports club but could easily be adapted for a contact manager, book or CD collection, customer database and many other purposes.

Our workshop uses Visual Basic 4.0. It makes use of features introduced in that version so you will not be able to follow the workshop using older editions. A little knowledge of VB is assumed, so complete beginners are advised to become familiar with the product before starting on the workshop.



Objects in focus

Visual Basic makes extensive use of objects. What is confusing, though, is that the word is used in several different ways. Here are three kinds of VB objects:

1. **Internal objects and controls.** For example, there is a global App object which has useful properties like Title and Path. There are also VB's built-in controls like command buttons and text boxes, represented by objects with properties, methods and events. These objects are VB's essential building blocks.
2. **OLE objects.** These include ActiveX controls, also known as OCX controls, and applications like Excel which expose functionality in the form of objects you can access from Visual Basic. The advantage of OLE objects is that they are system-wide and not just limited to one application.
3. **User-defined objects.** You create these by inserting class modules into your project. You can also customise forms by adding your own properties and methods.

If you have used VB at all, you will already have worked with the first two kinds of object but may not have used the third. It is possible to write major VB applications without using them, especially if either the application or the developer started in Visual Basic 3.0, where they did not exist. In fact, the Visual Basic environment does not encourage you to use them.

The obvious way to build an application is to draw buttons and controls onto a form, setting their properties and writing code for their events. With that approach you may not see the need to define your own objects. It is worth making the effort. Here are three reasons why:

1. Object-orientated programs are more robust and easier to debug. One reason for this is that you can avoid global variables, which are notoriously error prone, and use object properties instead.
2. Well-designed objects can be used in more than one application.
3. To exploit the power of OLE (Object Linking and Embedding) you need to define objects that can be made available to other applications.

This workshop will explore how to make use of VB's class modules, which create user-defined objects, in order to derive these benefits.

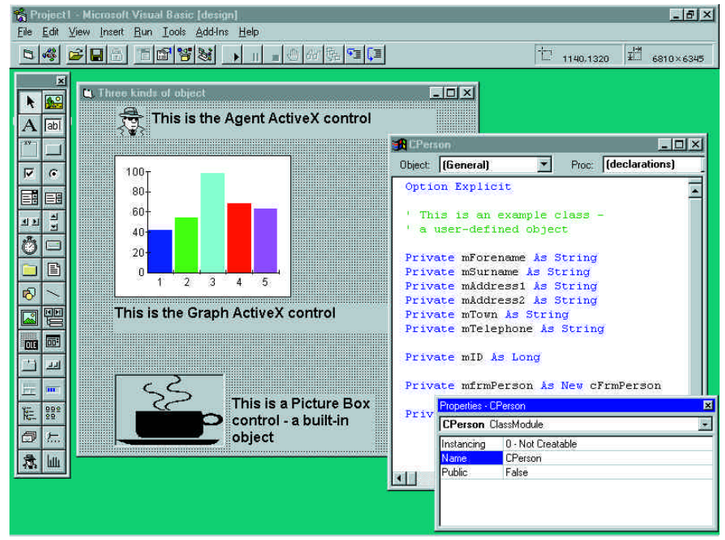


Fig 1 Not all VB objects are the same. This application shows three kinds: built-in, OLE, and user-defined

Building a database application

Anyone can build a Visual Basic database application. Just place a data control on a form, set the databasename and recordsource properties, add some bound text boxes to display the fields, and it's done. There is even an add-in that will do it for you; the data form designer. The typical result is shown in Fig 2.

The speed of development is impressive, but in other respects applications built like this are poor. For a start, a visible data control is not the world's most stylish graphical interface. Worse, it encourages a navigational approach to viewing data. If the visible record is not the one you want, click Next until you find it. It may work for half a dozen records but it's hopeless for large database tables. It is also fundamentally at odds with the set-based strategy of SQL, the native query language of VB's database engine. Additionally, working with bound controls increases the risk of inadvertently changing the data. All these problems can be overcome by adding code for searching, validation, and so on. Another option is to use an entirely different approach.

A particularly powerful technique uses a listbox and a text box to create a database searcher. The user types one or more letters into the text box and presses Enter. The listbox then fills with all the matching records. By double-clicking an entry in the list, the full details of the record can be displayed. It allows control over the precision of the search, and it is fast, with no need to enter criteria into a search dialogue.

Will the real VB stand up?

Visual Basic exists in various forms. The standalone product comes in three editions: Standard, Professional and Enterprise.

The Standard edition is cheap but not all that cheerful. It is fine for learning the Basic language but data access is limited to the data control. Few custom controls are included and it is unsuitable for creating applications for distribution. It works only on Windows 95 or NT. The Professional edition fills the gaps, includes a 16-bit version, full data access, important OLE features and a wide range of custom controls. For general-purpose work, the Professional edition is all you need. The Enterprise edition adds features for client-server work and team development.

That leaves two other types of VB. Visual Basic for Applications is the version that ships with Microsoft Office and now a number of third-party applications, too. VBA in Office 95 has no forms engine, which limits its power, but VBA 5.0 in Office 97 is almost the same as the standalone version. The main difference is that you cannot compile a standalone executable. VB Script is a stripped-down language for Internet Explorer. Microsoft hopes that other web browsers will adopt VB Script, too, although so far this has not happened.

p248 ➢

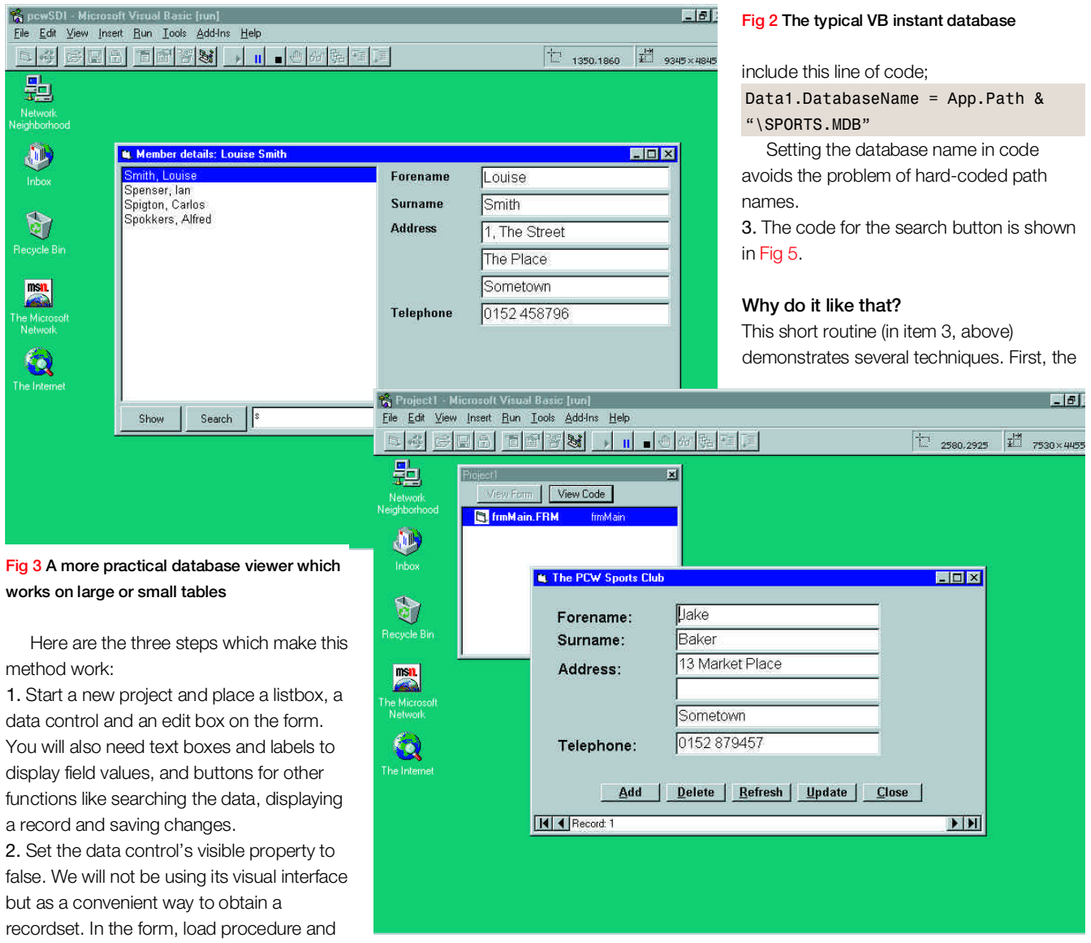


Fig 3 A more practical database viewer which works on large or small tables

Here are the three steps which make this method work:

1. Start a new project and place a listbox, a data control and an edit box on the form. You will also need text boxes and labels to display field values, and buttons for other functions like searching the data, displaying a record and saving changes.
2. Set the data control's visible property to false. We will not be using its visual interface but as a convenient way to obtain a recordset. In the form, load procedure and include this line of code;

Fig 2 The typical VB instant database

```
Data1.DatabaseName = App.Path &
"\SPORTS.MDB"
```

Setting the database name in code avoids the problem of hard-coded path names.

3. The code for the search button is shown in Fig 5.

Why do it like that?

This short routine (in item 3, above) demonstrates several techniques. First, the code uses SQL to create a dynaset-type recordset based on the text the user has entered. By adding the star character to the string and using the Like keyword, we find all the surname fields which begin with that string. JET, the Visual Basic database engine, is not case-sensitive, which simplifies matters. A nice feature is that the user can enter wildcards. For example, the string "??i" finds all surnames with a third letter i. Your users will think this is very clever, but it is VB's SQL that has done the work for you.

Second, the code uses a standard list box rather than the databound list box or the bound grid control. Using a databound control would save the few lines of code which fill the list. But unfortunately, the bound list control can only display one field, limiting its use. The databound grid is a viable option but is, frankly, overkill in view of what's required. In version 4.0, Microsoft enhanced VB's list box by adding the ItemData property and this is ideal since it lets you store an ID number against each item in the list. It is then easy to look up the correct record when the user selects the item.

The underlying principle is not to use a complicated ActiveX control where a simple, lightweight VB control will do just as well.

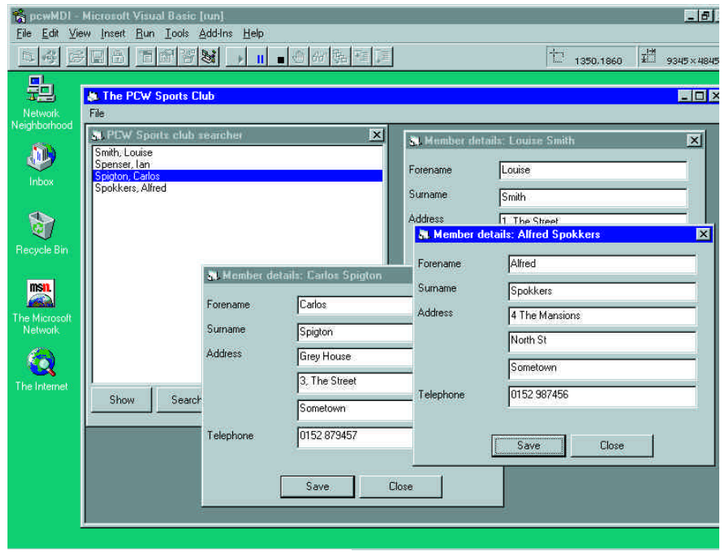


Fig 4 This alternative approach lets you view several records at once

Fig 5 Code for the search button

```
List1.Clear

' now do the search
Data1.RecordSource = _
"select * from members where members.surname like '" _
& Trim$(txSearch.Text) & "*' order by members.surname"
Data1.Refresh

' now fill the list box
If Not (Data1.Recordset.BOF And Data1.Recordset.EOF) Then
' there are matching records

Data1.Recordset.MoveFirst

Do While Not Data1.Recordset.EOF
List1.AddItem (Data1.Recordset!surname & ", " &
Data1.Recordset!forename)
List1.ItemData(List1.NewIndex) = Data1.Recordset!ID_NO
' stores the ID in the list box

Data1.Recordset.MoveNext
Loop

List1.ListIndex = 0 ' select first matching record
cbShow_Click ' show the first record

Else
' add code here to clear the form's fields, report no match, etc

End If
```

Putting objects to work

Not all the code is shown here (for reasons of space) but if you look at the example project on our cover-mounted CD you will notice a class module, CPerson.

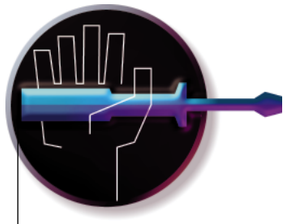
The application maintains an instance of the CPerson class and obtains member details by inspecting its properties. The Save button works by calling the save method of the currPerson object. This approach will bring several advantages as the application develops. For example, a weakness of traditional database forms is that they only show one record at a time. Fig 4 is a database application which uses an enhanced CPerson class that has the capability to display itself. That makes it easy to simultaneously view the details of several individuals.

■ *Next month: A closer look at VB class modules.*

•PCW Contacts

Tim Anderson welcomes your VB comments and tips. Contact him either by post c/o *PCW* or email at visual@pcw.vnu.co.uk

For more information about Visual Basic, contact **Microsoft** on 0345 002000.



State of the **union**

In the final part of our four-part tutorial, Mark Whitehorn covers UNION, insert, update and delete commands.

I ended last month's tutorial by illustrating that while you can have all of the cars some of the time, and all of the people some of the time (in your SQL statement), what you really want to know is: can we have *all* of the people *all* of the time? The answer is "yes" but you need to make use of UNION.

UNION returns all of the records from two queries and displays them, minus any duplicates, in a single table. Thus:

```
SELECT CARS.Make, CARS.Model,
EMPLOYEES.FirstName,
EMPLOYEES.LastName
FROM CARS RIGHT JOIN EMPLOYEES
ON CARS.CarNo = EMPLOYEES.CarNo
UNION
SELECT CARS.Make, CARS.Model,
EMPLOYEES.FirstName,
EMPLOYEES.LastName
FROM CARS LEFT JOIN EMPLOYEES
ON CARS.CarNo = EMPLOYEES.CarNo;
```

produces:

Make	Model	FirstName	LastName
		John	Greeves
Aston Martin	DB Mk III		
Bentley	Mk. VI	Bilda	Groves
Ford	GT 40		
Ford	Mustang		
Jaguar	D Type		
Shelby	Cobra	Sally	Smith
Triumph	Spitfire		
Triumph	Stag	Fred	Jones

Clearly, the two answer tables that are produced by the separate SELECT statements must be compatible in order for the UNION to combine them sensibly. So:

```
SELECT CARS.CarNo, CARS.Model,
EMPLOYEES.FirstName,
EMPLOYEES.LastName
FROM CARS RIGHT JOIN EMPLOYEES
ON CARS.CarNo = EMPLOYEES.CarNo
UNION
SELECT CARS.Make, CARS.Model,
EMPLOYEES.FirstName,
EMPLOYEES.LastName
FROM CARS LEFT JOIN EMPLOYEES
ON CARS.CarNo = EMPLOYEES.CarNo;
```

attempts to put text and numeric data into the same field and should fail. (In practice, some RDBMSs will allow this and convert the resulting field to the lowest common denominator, such as text.)

However, the result shown in Fig 1 *(page 269)* may not be particularly meaningful.

The first example I gave for UNION (combining a LEFT and RIGHT join) serves as an excellent example. However, it isn't the only way in which it can be used. Suppose that you have another table of sales people who, for whatever reason, are stored in a separate table from the other employees. Take a look at the following:

SALESPEOPLE

EmployeeNo	FirstName	LastName	CarNo
1	Fred	Williams	1
2	Sarah	Watson	4
3	James	Hatlitch	6
4	Simon	Webaston	
5	Sally	Harcourt	
6	Martin	Boxer	
7	Trevor	Wright	7

You want to throw a party for all the employees, and to include those sales people with company cars (because they have volunteered to drive the employees home afterwards).

You can use:

```
SELECT FirstName, LastName
FROM EMPLOYEES
UNION
SELECT FirstName, LastName
FROM SALESPERSON
WHERE SALESPERSON.CarNo Is Not
Null;
```

to yield;

FirstName	LastName
Bilda	Groves
Fred	Jones
Fred	Williams
James	Hatlitch
John	Greeves
Sally	Smith
Sarah	Watson
Trevor	Wright

You can also use UNION to produce a list of all employees and sales people who have company cars:

```
SELECT DISTINCTROW
SALESPEOPLE.FirstName,
SALESPEOPLE.LastName, CARS.Make,
CARS.Model
FROM
(CARS INNER JOIN SALESPEOPLE
ON CARS.CarNo = SALESPEOPLE.CarNo)
UNION
SELECT DISTINCTROW
EMPLOYEES.FirstName,
EMPLOYEES.LastName, CARS.Make,
CARS.Model
FROM
(CARS INNER JOIN EMPLOYEES
ON CARS.CarNo = EMPLOYEES.CarNo);
```

FirstName	LastName	Make	Model
Bilda	Groves	Bentley	Mk. VI
Fred	Jones	Triumph	Stag
Fred	Williams	Triumph	Spitfire
James	Hatlitch	Ford	Mustang
Sally	Smith	Shelby	Cobra
Sarah	Watson	Ford	GT 40
Trevor	Wright	Aston Martin	DB Mk III

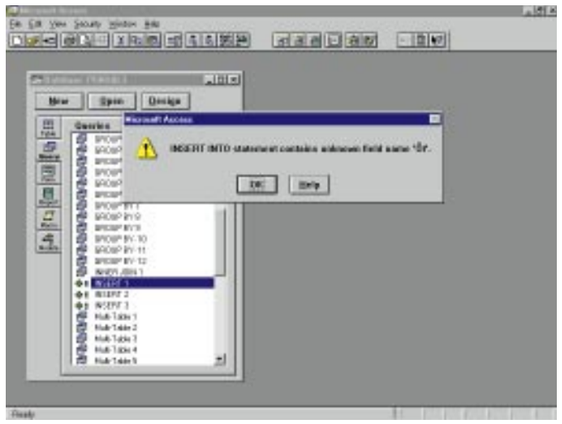


Fig 3 Error message generated when the first INSERT command is used too frequently!

SELECT summary

Suppose you import a table of data like this:

InvoiceNo	Foo
1	King
2	Baby Blue
3	Royal
2	Crested
5	Humbolt
2	Jackass

into a database and then try to make the field InvoiceNo into a primary key (*the Foo field is simply a shorthand representation of the boring information that would usually be displayed in an invoice*). This should fail because the field contains duplicate values. In this tiny table we can see them, but what if it had 50,000 records? With a little imagination, a query will find the errant records for us.

```
SELECT InvoiceNo, Count(InvoiceNo)
AS NoOfDuplications
FROM INVOICES
GROUP BY [InvoiceNo]
HAVING Count([InvoiceNo])>1;
```

InvoiceNo	NoOfDuplications
2	3

INSERT

Firstly, a brief note about the sample Access database which is provided. It is tempting to open each query as an SQL view, read it, and then look at the answer table by pressing the "Datasheet View button". This works for most of the examples provided but not for the INSERT queries. Press the "Run" button instead.

It is also worth bearing in mind that these queries will update the base tables, so you should be working on a copy of the database. In addition, remember that the tables have primary keys, so if you run the same INSERT query twice without deleting the additional record, the query will fail to run the second time.

As if all that weren't enough, please also note that I have encountered what appear to be "software anomalies" in using these queries in Access 2.0. The first example of an SQL INSERT statement will only run two or three times. Thereafter, even if the new record is dutifully deleted from the target table, the query will generate the error message shown in the screenshot, Fig 3. This is despite the fact that it hasn't been edited, or even opened for editing. Once this error message appears, the only way to get the query to run again is to delete the existing query, open a new one and type the SQL statement again.

SELECT is undoubtedly the most commonly used SQL statement, but we shouldn't forget the other members of the Data Manipulation Language (DML), INSERT, UPDATE and DELETE.

INSERT is used to add rows to a table. Thus the statement:

```
INSERT INTO SALES
VALUES (8, 1, "Jones", "Sofa",
"Harrison", 235.67);
```

This is not the only allowable construction. Indeed, Access will run this syntactical construction, but if you save the query, Access converts it to :

```
INSERT INTO SALES
SELECT 8, 1, "Jones", "Sofa",
"Harrison", 235.67;
```

Both constructions will add this record to the SALES table shown in Fig 2.

A slightly more verbose form is possible:

```
INSERT INTO SALES ( SaleNo,
EmployeeNo, Customer, Item,
Supplier, Amount )
SELECT 8, 1, "Jones", "Sofa",
"Harrison", 235.67;
```

which has exactly the same result. We can also add to specific fields:

```
INSERT INTO SALES ( SaleNo,
EmployeeNo, Customer, Amount )
SELECT 9, 1, "Jones", 235.67;
```

which adds a single record as shown in Fig 4.

But don't forget closure. Any operation that we perform on a table (or tables) in a relational database must have, as its result, another table. So suppose we write an INSERT statement like this:

```
INSERT INTO SALES
VALUES
(SELECT
        FROM SALES2
        WHERE SaleNo > 200);
```

The table SALES2 looks like that shown in Fig 5, and this SQL statement will add the five records for which [SaleNo] is greater than 200 to the SALES table.

Closure is important here because the statement within the brackets:

```
(SELECT
        FROM SALES2
        WHERE SaleNo > 200);
```

generates a table in its own right which is then INSERTED into SALES.

SQL is not always as standard as it should be. As another example, the syntax for this statement in Access is:

```
INSERT INTO SALES
SELECT *
FROM SALES2
WHERE SaleNo > 200;
```

UPDATE

The UPDATE command allows you to alter the values in fields. The general format is:

```
UPDATE tablename
SET Fieldname(s) = value
WHERE fieldname = value
```

although the WHERE condition is optional. Thus:

```
UPDATE SALES
SET Customer ="Smith";
```

will change Fig 6 to Fig 7.

As you might imagine, this command can be a little devastating in the wrong hands. The WHERE command generally limits its scope. So:

```
UPDATE SALES
SET Customer ="Smith"
WHERE Customer = "Simpson";
```

will act on the same initial table to produce that shown in Fig 8.

It is quite possible to use different fields in the SET and WHERE clauses. Thus:

```
UPDATE SALES
SET Customer ="Smith"
WHERE SaleNo < 5;
```

produces Fig 9.

Other variations are possible, and indeed common. For example:

```
UPDATE SALES
SET AMOUNT = AMOUNT * 1.1;
```

will update all the values in SALES.[Amount] by 10 percent, as in Fig 10. This sort of variant is particularly useful.

DELETE

The DELETE command allows you to alter

p268 ➢

the values in fields.

The general format of the command is:

```
DELETE FieldName(s)
FROM tablename
WHERE fieldname = value
```

although the WHERE condition is optional. Thus:

```
DELETE *
FROM SALES;
```

is a particularly powerful (not to say dangerous) statement since the output table looks like Fig 11. To be more specific, this command deletes the entire contents of the SALES table. *Please be aware of the consequences of any injudicious use of this command.*

More commonly (and less alarmingly) the command is used like this:

```
DELETE *
FROM SALES
WHERE [EmployeeNo] = 2;
```

which deletes two records and produces the table in Fig 12. Of course, closure comes into its own and we can write statements like:

```
DELETE *
FROM EMPLOYEES
WHERE EmployeeNo IN
(SELECT EmployeeNo
FROM SALES
GROUP BY EmployeeNo
HAVING COUNT (*) < 2);
```

which is neither friendly nor amiable, but effective in database terms. It deletes all employees from the EMPLOYEES table who have made fewer than 2 sales. The SALES table is unaffected, but one of our employees disappears from EMPLOYEES.

Bear in mind that this statement will try to remove employees who have performed badly, but the data dictionary may in fact prevent this deletion in order to preserve data integrity. This will depend upon whether Cascade Delete has been set between the two tables. In the sample database, the query will complete.

Summary

SQL is great, and if you spend any time at all with databases, it repays the effort required to learn it. One of the best ways to learn is to practise using it, which is why the sample database has 70 sample queries. However, you might also like to wile away your time on this brainteaser:

■ Question (and a free SQL diagnostic tool)

The two SQL statements below are perfectly legal. Both will run. The question is, which will be sensible? One of them will find all the records where the SaleNo is >200 and order the answer table by EmployeeNo and SaleNo. The other won't.

Q1

```
SELECT *
FROM SALES2
WHERE SaleNo>200
ORDER BY EmployeeNo, SaleNo;
```

or is it…

Q2

```
SELECT *
FROM SALES2
WHERE SaleNo>200
ORDER BY EmployeeNo AND SaleNo;
```

The only difference, to save you wasting time comparing them, is in the ORDER BY statement.

Answer: Q1 is correct and returns the table shown in Fig 13. Q2 returns the table in Fig 14 because it has a very odd construction:

```
ORDER BY EmployeeNo AND SaleNo
```

Despite appearances, this does NOT say "order the records by EmployeeNo and then by SaleNo". Instead, it says "evaluate the expression 'EmployeeNo AND SaleNo' for truth (the answer will come back as -1 [True] or 0 [False]) and then stack the records based on this value." You can prove this by adding the expressions which are being evaluated to the list of information that you want to see. Thus:

```
SELECT SaleNo>200 AS
['SaleNo>200'],
EmployeeNo AND SaleNo AS ['Emp AND
Sale'],
EmployeeNo, SaleNo, Customer
FROM SALES2
WHERE SaleNo>200
ORDER BY EmployeeNo AND SaleNo;
```

produces Fig 15. In all the records, the expression 'EmployeeNo AND SaleNo' happens to evaluate to -1, so the sorting has no effect.

If and when you come across an intractable SQL statement that runs but doesn't give you the answer you expect, then you can use SQL's own ability to show you the results of expressions as a diagnostic tool.

Figs 1-15

Examples to accompany part four of the SQL tutorial, covering the UNION, INSERT, UPDATE and DELETE commands, and the associated brainteaser.

Fig 1

Car No	Model	First Name	Last Name
		John	Greeves
2	Mk. VI	Bilda	Groves
3	Stag	Fred	Jones
5	Cobra	Sally	Smith
	Aston Martin DB Mk III		
	Bentley Mk. VI	Bilda	Groves
	Ford GT 40		
	Ford Mustang		
	Jaguar D Type		
	Shelby Cobra	Sally	Smith
	Triumph Spitfire		
	Triumph Stag	Fred	Jones

Fig 2

Sale No	Employee No	Customer	Item	Supplier	Amount
8	1	Jones	Sofa	Harrison	£235.67

Fig 3

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
1	1	Simpson	Sofa	Harrison	£235.67
2	1	Johnson	Chair	Harrison	£453.78
3	2	Smith	Stool	Ford	£82.78
4	2	Jones	Suite	Harrison	£3,421
5	3	Smith	Sofa	Harrison	£235.67
6	1	Simpson	Sofa	Harrison	£235.67
7	1	Jones	Bed	Ford	£453
8	1	Jones	Sofa	Harrison	£235.67
9	1	Jones			£235.67

Fig 2

Sale No	Employee No	Customer	Item	Supplier	Amount
3	2	Smith	Stool	Ford	£82.78
5	3	Smith	Sofa	Harrison	£235.67
213	3	Williams	Suite	Harrison	£3421
216	2	McGreggor	Bed	Ford	£453
217	1	Williams	Sofa	Harrison	£235.67
218	3	Aitken	Sofa	Harrison	£235.67
225	2	Aitken	Chair	Harrison	£453.78

Fig 6 — will change to…

Sale No	Employee No	Customer	Item	Supplier	Amount
1	1	Simpson	Sofa	Harrison	£235.67
2	1	Johnson	Chair	Harrison	£453.78
3	2	Smith	Stool	Ford	£82.78
4	2	Jones	Suite	Harrison	£3,421
5	3	Smith	Sofa	Harrison	£235.67
6	1	Simpson	Sofa	Harrison	£235.67
7	1	Jones	Bed	Ford	£453

…Fig 7

Sale No	Employee No	Customer	Item	Supplier	Amount
1	1	Smith	Sofa	Harrison	£235.67
2	1	Smith	Chair	Harrison	£453.78
3	2	Smith	Stool	Ford	£82.78
4	2	Smith	Suite	Harrison	£3,421
5	3	Smith	Sofa	Harrison	£235.67
6	1	Smith	Sofa	Harrison	£235.67
7	1	Smith	Bed	Ford	£453

Fig 8

Sale No	Employee No	Customer	Item	Supplier	Amount
1	1	Smith	Sofa	Harrison	£235.67
2	1	Johnson	Chair	Harrison	£453.78
3	2	Smith	Stool	Ford	£82.78
4	2	Jones	Suite	Harrison	£3,421
5	3	Smith	Sofa	Harrison	£235.67
6	1	Smith	Sofa	Harrison	£235.67
7	1	Jones	Bed	Ford	£453

Fig 9

Sale No	Employee No	Customer	Item	Supplier	Amount
1	1	Smith	Sofa	Harrison	£235.67
2	1	Smith	Chair	Harrison	£453.78
3	2	Smith	Stool	Ford	£82.78
4	2	Smith	Suite	Harrison	£3,421
5	3	Smith	Sofa	Harrison	£235.67
6	1	Simpson	Sofa	Harrison	£235.67
7	1	Jones	Bed	Ford	£453

Fig 10

Sale No	Employee No	Customer	Item	Supplier	Amount
1	1	Simpson	Sofa	Harrison	£259.24
2	1	Johnson	Chair	Harrison	£499.16
3	2	Smith	Stool	Ford	£91.06
4	2	Jones	Suite	Harrison	£3,763.10
5	3	Smith	Sofa	Harrison	£259.24
6	1	Simpson	Sofa	Harrison	£259.24
7	1	Jones	Bed	Ford	£498.30

Fig 11

Sale No	Employee No	Customer	Item	Supplier	Amount

Fig 12

Sale No	EmployeeNo	Customer	Item	Supplier	Amount
1	1	Simpson	Sofa	Harrison	£235.67
2	1	Johnson	Chair	Harrison	£453.78
5	3	Smith	Sofa	Harrison	£235.67
6	1	Simpson	Sofa	Harrison	£235.67
7	1	Jones	Bed	Ford	£453

Fig 13 — the correct answer

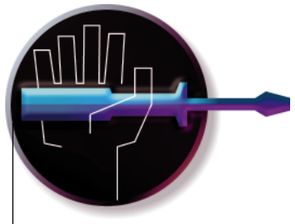
Sale No	Employee No	Customer	Item	Supplier	Amount
217	1	Williams	Sofa	Harrison	£235.67
216	2	McGreggor	Bed	Ford	£453
225	2	Aitken	Chair	Harrison	£453.78
213	3	Williams	Suite	Harrison	£3,421
218	3	Aitken	Sofa	Harrison	£235.67

Fig 14

Sale No	Employee No	Customer	Item	Supplier	Amount
225	2	Aitken	Chair	Harrison	£453.78
218	3	Aitken	Sofa	Harrison	£235.67
217	1	Williams	Sofa	Harrison	£235.67
216	2	McGreggor	Bed	Ford	£453
213	3	Williams	Suite	Harrison	£3,421

Fig 15

'SaleNo>200'	'Emp AND Sale'	EmployeeNo	SaleNo	Customer
-1	-1	3	213	Williams
-1	-1	2	216	McGreggor
-1	-1	1	217	Williams
-1	-1	3	218	Aitken
-1	-1	2	225	Aitken



Practical **joinery**

In part III of our SQL tutorial, Mark Whitehorn explains how multiple tables work together and highlights the distinction between left, right, inner and outer joins.

Last month I promised to continue dealing with the subject of working with multiple tables and how to use the SELECT statement to draw data from more than one. This month, I'll look at how it works and what it's doing.

The sample tables and the joins between them are shown in the two screenshots (Figs 1 & 2). In the sample Access database, which is included on our cover-mounted CD, I have removed the joins. Some of the SQL commands alter the sample tables so I have included extra versions of those, stored with the word SAFE after the name. Once you have run the SQL statement, you can delete the altered table and replace it with a copy of the "safe" version. This replacement process is easier if the joins are removed.

The only difference between these tables and the way they appeared in last month's issue is that John Greeves has lost his licence, so he is no longer allocated a company car. (This does not affect any of the examples shown in previous months.)

Note that in order to maintain consistency with my previous article, the first SQL statement this month is labelled as "Multi-Table 3" (not "Multi-Table 1") in the Access database provided on the cover-mounted CD-ROM. Last month, we looked at SQL which worked across multiple tables. The statement we finished with was:

```
SELECT SALES.Customer,
EMPLOYEES.LastName, SALES.Amount
FROM SALES, EMPLOYEES
WHERE SALES.EmployeeNo =
 EMPLOYEES.EmployeeNo;
```

which yields:

Customer	LastName	Amount
Simpson	Groves	£235.67
Johnson	Groves	£453.78
Simpson	Groves	£235.67
Jones	Groves	£453
Smith	Greeves	£82.78
Jones	Greeves	£3,421
Smith	Smith	£235.67

In order to see how this is working, we can add the EmployeeNo fields, from the two tables, into the answer table and remove the WHERE statement. (I've used synonyms for the tables to reduce the size of the table headings.)

```
SELECT S.Customer, E.LastName,
S.Amount, S.EmployeeNo,
E.EmployeeNo
FROM SALES S, EMPLOYEES E;
```

See the table in Fig 3 (page 258).

Without a WHERE clause, the answer table contains every record in the SALES table matched against every record in the EMPLOYEE table, giving 4 x 7 = 28 records. The WHERE clause ensures that we see in the answer table only those records in which the EmployeeNo in SALES matches the EmployeeNo in EMPLOYEES. This is logically reasonable since we are using the value in SALES.EmployeeNo to indicate which employee made the sale.

It is possible to join more than two tables by adding to the WHERE clause. For example:

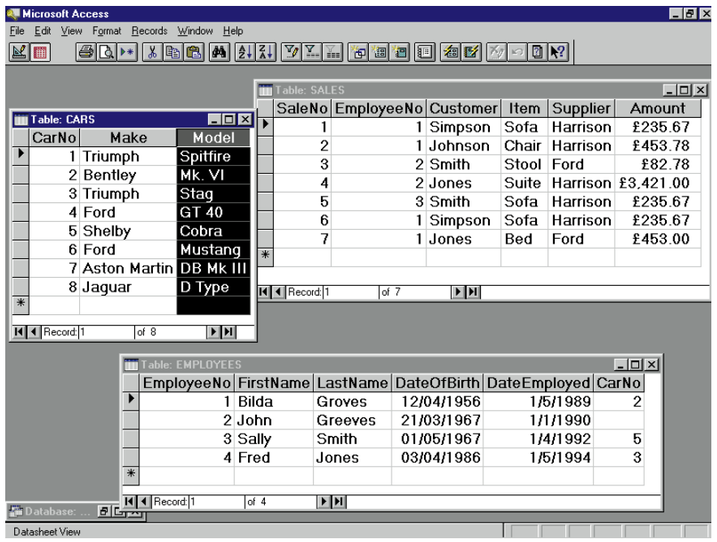


Fig 1 The tables used in the examples. I have set the dates to show four-digit years in response to email from readers worried about the coming of the millennium. In fact, Access stores all dates as four-digit years: it is just the default format which doesn't show them

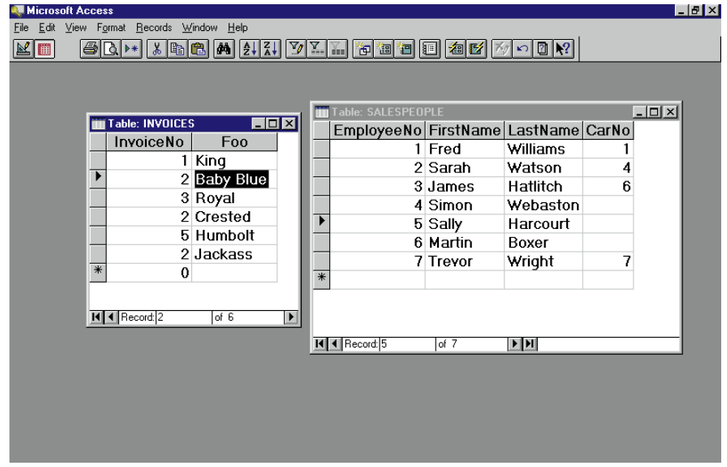


Fig 2 Two tables used in a couple of the later examples.The Foo field is simply a shorthand representation of the boring information that would usually be displayed in an invoice

```
SELECT SALES.Customer,
EMPLOYEES.FirstName, CARS.Make,
CARS.Model
FROM CARS, EMPLOYEES, SALES
WHERE EMPLOYEES.EmployeeNo =
SALES.EmployeeNo
AND EMPLOYEES.CarNo = CARS.CarNo;
```

Customer	FirstName	Make	Model
Simpson	Bilda	Bentley	Mk. VI
Johnson	Bilda	Bentley	Mk. VI
Simpson	Bilda	Bentley	Mk. VI
Jones	Bilda	Bentley	Mk. VI
Smith	Sally	Shelby	Cobra

Note that this query is finding the car driven by the sales person who dealt with a given customer, so it isn't supposed to present particularly meaningful information.

The most recent ISO standard for SQL (SQL-92) includes a new way of expressing joins such that:

```
SELECT SALES.Customer,
EMPLOYEES.LastName, SALES.Amount
FROM SALES, EMPLOYEES
WHERE SALES.EmployeeNo =
EMPLOYEES.EmployeeNo;
```

Customer	LastName	Amount
Simpson	Groves	£235.67
Johnson	Groves	£453.78
Simpson	Groves	£235.67
Jones	Groves	£453
Smith	Greeves	£82.78
Jones	Greeves	£3,421
Smith	Smith	£235.67

can be replaced by:

```
SELECT SALES.Customer,
EMPLOYEES.LastName, SALES.Amount
FROM SALES INNER JOIN EMPLOYEES
ON EMPLOYEES.EmployeeNo =
SALES.EmployeeNo;
```

This produces the same answer table and is generally considered to be more readable. However, it does raise another question: what is this INNER business?

Inner (natural) joins

Suppose your boss says: "Give me a list of all the cars and the sales person to whom each is currently allocated."

You are immediately tempted to use the SQL statement:

```
SELECT CARS.Make, CARS.Model,
EMPLOYEES.FirstName,
EMPLOYEES.LastName
FROM CARS INNER JOIN EMPLOYEES
ON CARS.CarNo = EMPLOYEES.CarNo;
```

but this will give the answer:

Make	Model	FirstName	LastName
Bentley	Mk. VI	Bilda	Groves
Triumph	Stag	Fred	Jones
Shelby	Cobra	Sally	Smith

which doesn't list all the cars because that delectable D-type Jaguar, for instance, hasn't been allocated to anyone.

In fact, your boss has phrased the question badly, since the original question assumes that every car is allocated to an employee and this is not the case. However, voicing your opinion about the inexact use of English is likely to be a CLM (Career Limiting Move). Better to keep quiet and find a query that will list all the cars and show what has been allocated to which lucky employees.

But before that, we'll have a look at what's wrong with the query shown above. By default, a join combines the two tables via fields that have identical values. This is known as a "Natural" or "Inner" join. However, if one or both of the fields contain unique values (I am using the term "unique" to mean that the values are found in one table but not the other) then the join ignores the records that are associated with these values. Thus, the table CARS has a delightful Aston Martin, CarNo = 7, but since there is no corresponding value in EMPLOYEES.CarNo, this fine automobile never appears in the answer table.

So instead of a Natural join, what you need to use here is an Unnatural join. Okay, I admit it, that was just to see if you were awake. It is really known as an "Outer" join.

Outer joins

There are two distinct types of Outer join, Left and Right.

The following SQL statement

```
SELECT CARS.Make, CARS.Model,
EMPLOYEES.FirstName,
EMPLOYEES.LastName
FROM CARS LEFT JOIN EMPLOYEES
ON CARS.CarNo = EMPLOYEES.CarNo;
```

yields:

Make	Model	FirstName	LastName
Triumph	Spitfire		
Bentley	Mk. VI	Bilda	Groves
Triumph	Stag	Fred	Jones
Ford	GT 40		
Shelby	Cobra	Sally	Smith
Ford	Mustang		
Aston Martin	DB Mk III		
Jaguar	D Type		

p258 ➢

Essentially, the substitution of LEFT JOIN for INNER JOIN has made all the difference.

The other sort of outer join is RIGHT, which simply ensures that every record in the table on the right-hand side of the join is included in the answer table, so

```
SELECT CARS.Make, CARS.Model,
EMPLOYEES.FirstName,
EMPLOYEES.LastName
FROM CARS RIGHT JOIN EMPLOYEES
ON CARS.CarNo = EMPLOYEES.CarNo;
```

yields:

Make	Model	FirstName	LastName
		John	Greeves
Bentley	Mk. VI	Bilda	Groves
Triumph	Stag	Fred	Jones
Shelby	Cobra	Sally	Smith

It is important to note that

```
SELECT CARS.Make, CARS.Model,
EMPLOYEES.FirstName,
EMPLOYEES.LastName
FROM EMPLOYEES LEFT JOIN CARS
ON CARS.CarNo = EMPLOYEES.CarNo;
```

produces exactly the same answer table, namely:

Make	Model	FirstName	LastName
		John	Greeves
Bentley	Mk. VI	Bilda	Groves
Triumph	Stag	Fred	Jones
Shelby	Cobra	Sally	Smith

In other words, the LEFT and RIGHT simply refer to the tables as named in the SQL statement. So

```
FROM EMPLOYEES LEFT JOIN CARS
```

and

```
FROM CARS RIGHT JOIN EMPLOYEES
```

will include all the employees and some of the cars:

```
FROM CARS LEFT
JOIN EMPLOYEES
```

and

```
FROM EMPLOYEES
RIGHT JOIN CARS
```

will include all the cars and some of the employees.

The upshot is that you can have all of the cars some of the time, and indeed, you can have all of the people some of the time. But what you really want to know is, can we have all of the cars and all of the people all of the time?

The answer, not surprisingly, is "Yes". But in order for that to happen, we need to make use of UNION and I'll be covering this and other topics in next month's column.

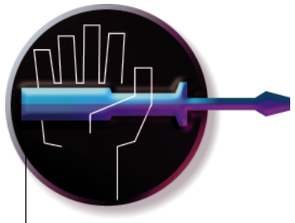
■ You will find the Access sample file in the Resources section on this month's cover-mounted CD.

Fig 3

Customer	LastName	Amount	S.EmployeeNo	E.EmployeeNo
Simpson	Groves	£235.67	1	1
Johnson	Groves	£453.78	1	1
Smith	Groves	£82.78	2	1
Jones	Groves	£3,421	2	1
Smith	Groves	£235.67	3	1
Simpson	Groves	£235.67	1	1
Jones	Groves	£453	1	1
Simpson	Greeves	£235.67	1	2
Johnson	Greeves	£453.78	1	2
Smith	Greeves	£82.78	2	2
Jones	Greeves	£3,421	2	2
Smith	Greeves	£235.67	3	2
Simpson	Greeves	£235.67	1	2
Jones	Greeves	£453	1	2
Simpson	Smith	£235.67	1	3
Johnson	Smith	£453.78	1	3
Smith	Smith	£82.78	2	3
Jones	Smith	£3,421	2	3
Smith	Smith	£235.67	3	3
Simpson	Smith	£235.67	1	3
Jones	Smith	£453	1	3
Simpson	Jones	£235.67	1	4
Johnson	Jones	£453.78	1	4
Smith	Jones	£82.78	2	4
Jones	Jones	£3,421	2	4
Smith	Jones	£235.67	3	4
Simpson	Jones	£235.67	1	4
Jones	Jones	£453	1	4

PCW Contacts

Mark Whitehorn welcomes readers' correspondence. He is at **m.whitehorn@dundee.ac.uk**



Group therapy

Having dealt with the basics, Mark Whitehorn delves deeper into SQL and shows you how to organise your records logically, in part II of our four-part tutorial.

Last month, we looked at the basic building-blocks of SQL and the ways in which they can be put together to elicit information from a database. With those commands alone you could pose an almost infinite series of queries, but SQL still has a whole range of tricks up its sleeve.

(*Last month's sample tables reappear in the screen shots here, so you won't have to fight with two magazines at once).*

Built-in Functions

SQL includes several simple statistical functions:

Function	
SUM	Total
COUNT	The number of occurrences
AVG	Average
MIN	Minimum
MAX	Maximum

Thus, it is possible (although not normal practice) to write SQL statements such as:

```
SELECT SUM(Amount)
FROM SALES;
```

Some systems will actually accept this. Access, for instance, generates a "dummy" field name (Expr1000) and yields the following table:

Expr1000
£5,117.57

It is common to explicitly name the field into which to place the output. For example:

```
SELECT SUM(Amount) "Sum of Amount"
FROM SALES;
```

or:

```
SELECT SUM(Amount) AS SumOfAmount
FROM SALES;
```

or even:

```
SELECT DISTINCTROW
SUM(SALES.Amount) AS SumOfAmount
FROM SALES;
```

which is how it appears in the Access dialect of SQL. All three of the above yield a table like this:

SumOfAmount
£5,117.57

The AS followed by a field name simply tells the SQL statement to put the data into a field of that name in the answer table.

It is permissible to mix two or more functions, for example:

```
SELECT SUM(Amount) AS SumOfAmount,
COUNT(Amount) AS CountOfAmount,
AVG(Amount) AS AvgOfAmount,
MIN(Amount) AS MinOfAmount,
MAX(Amount) AS MaxOfAmount
FROM SALES;
```

which yields the table shown in Fig 1.

It's also perfectly permissible to mix fields like this:

```
SELECT COUNT(Customer) AS
 CountOfCustomer,
AVG(Amount) AS AvgOfAmount
FROM SALES;
```

giving:

CountOfCustomer	AvgOfAmount
7	£731.08

These functions will even operate on fields which contain no data. If we amend the base table (for the sake of this example only) to be as shown in Fig 2, then the SQL statement above will give:

CountOfCustomer	AvgOfAmount
5	£282.76

The COUNT function finds only five values and AVG sums the values that it finds and then divides the result by six (i.e.the number of values in that particular field) rather than seven (the number of records).

However, these functions are designed to yield only a single figure each. Thus, SQL statements such as:

```
SELECT Customer,
AVG(SALES.Amount) AS AvgOfAmount
FROM SALES;
```

are illegal because SELECT Customer can (and in this case, would) have an output consisting of multiple records, while the second:

```
SELECT AVG(SALES.Amount) AS
AvgOfAmount
```

can only have an output of a single record.

Several SQL implementations provide more than the basic functions.

For example, Access also provides:

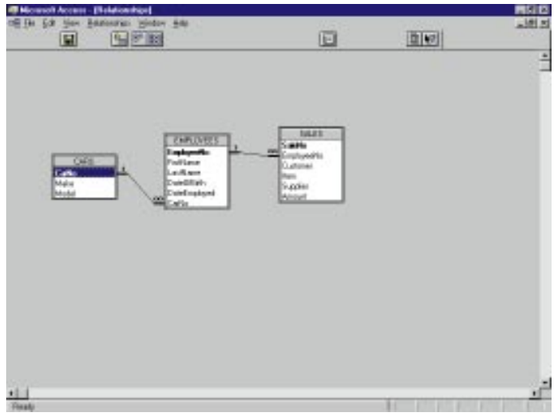
Function	
StDev	Standard Deviation
Var	Variance

Fig 1

SumOfAmount	CountOfAmount	AvgOfAmount	MinOfAmount	MaxOfAmount
£5,117.57	7	£731.08	£82.78	£3,421.00

Fig 2

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
1	1	Simpson	Sofa	Harrison	£235.67
2	1	Johnson	Chair	Harrison	£453.78
3	2		Stool	Ford	£82.78
4	2	Jones	Suite	Harrison	
5	3	Smith	Sofa	Harrison	£235.67
6	1		Sofa	Harrison	£235.67
7	1	Jones	Bed	Ford	£453.00



The relationship editor, showing the joins between the tables

It is just this kind of variation from the standard which demonstrates that SQL is still a fairly fluid standard.

GROUP BY — collecting information

So far, our generic SELECT statement looks like this:

```
SELECT field name(s)
FROM table name
WHERE condition(s)
ORDER BY field name(s)
```

We can expand it with:

```
SELECT field name(s)
FROM table name
WHERE condition(s)
GROUP BY Field name(s)
ORDER BY field name(s)
```

Last month we looked at the command ORDER BY, which provides a way of presenting information in ascending or descending order. Further control over your answer data is given by GROUP BY. The syntax is:

```
GROUP BY Field name(s)
```

To illustrate its usefulness, we'll consider the simple statement below:

```
SELECT AVG(Amount) AS AvgOfAmount
FROM SALES;
```

AvgOfAmount
£731.08

This averages the values found in the [Amount] field for all records in the SALES table. Suppose you want to examine the records which refer to customer "Simpson"? You'd use WHERE, as follows:-

```
SELECT AVG(Amount) AS AvgOfAmount
FROM SALES
WHERE Customer = "Simpson";
```

AvgOfAmount
£235.67

Now, suppose you want to do this for each customer. An inelegant, brute-force solution would be to run the query multiple times, once each for each customer. A clever solution is to get the SQL statement to group the records together by the name of the customer and then apply the AVG function to the values in the groups.

We can visualise the process as follows: going from the data shown in Fig 3, to that shown in Fig 4; and then to this, which is a full but compact summary of the required information:

Customer	AvgOfAmount
Johnson	£453.78
Jones	£1,937.00
Simpson	£235.67
Smith	£159.23

The SQL statement required to perform this magic is impressive:

```
SELECT Customer, AVG(Amount) AS
AvgOfAmount
FROM SALES
GROUP BY Customer
Order BY Customer;
```

The GROUP BY clause can be used more simply than this. For example:

```
SELECT Customer
FROM SALES
GROUP BY Customer;
```

produces:

Customer
Johnson
Jones
Simpson
Smith

At first, it appears that this is the same as:

```
SELECT DISTINCT Customer
FROM SALES;
```

which yields the same answer table, but adding another field demonstrates the difference. Thus:

```
SELECT DISTINCT Customer, Amount
FROM SALES;
```

produces:

Customer	Amount
Johnson	£453.78
Jones	£453.00
Jones	£3,421.00
Simpson	£235.67
Smith	£82.78
Smith	£235.67

whereas:

```
SELECT Customer, Amount
FROM SALES
GROUP BY Customer;
```

fails to run. Why? To answer this, we must look at what the SQL clauses are trying to achieve. The command:

```
SELECT Customer
FROM SALES
GROUP BY Customer;
```

essentially says "Sort the records in the SALES table so that identical values in the Customer field are together. Then 'crush together' the records with identical Customer values so that they appear to be one record." Thus:

```
SELECT Customer, Amount
FROM SALES
GROUP BY Customer;
```

Fig 3

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
1	1	Simpson	Sofa	Harrison	£235.67
2	1	Johnson	Chair	Harrison	£453.78
3	2	Smith	Stool	Ford	£82.78
4	2	Jones	Suite	Harrison	£3,421.00
5	3	Smith	Sofa	Harrison	£235.67
6	1	Simpson	Sofa	Harrison	£235.67
7	1	Jones	Bed	Ford	£453.00

Fig 4

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
2	1	Johnson	Chair	Harrison	£453.78
7	1	Jones	Bed	Ford	£453.00
4	2	Jones	Suite	Harrison	£3,421.00
6	1	Simpson	Sofa	Harrison	£235.67
1	1	Simpson	Sofa	Harrison	£235.67
5	3	Smith	Sofa	Harrison	£235.67
3	2	Smith	Stool	Ford	£82.78

p248

fails because there's a conflict (real in this case, potential in others) between the number of records that should be output.

```
SELECT Customer
FROM SALES
GROUP BY Customer;
```

will output four records:

Customer
Johnson
Jones
Simpson
Smith

while:

```
SELECT  Amount
FROM SALES;
```

will output seven records:

Amount
£235.67
£453.78
£82.78
£3,421.00
£235.67
£235.67
£453.00

Combining these two incompatible requests is impossible and SQL engines will refuse the statement. As you can see from the above, there is no obligation to combine GROUP BY with one or more of the functions. However, it is commonly done because we often only want to group records in order to be able to perform some type of manipulation on selections of records. It is perfectly possible to GROUP BY more than one field.

Thus:

```
SELECT Customer, Supplier,
 AVG(Amount) AS AvgOfAmount
FROM SALES
GROUP BY Customer, Supplier;
```

produces more groups than the SQL statement above that grouped by one field, because it is grouping those records which share the same value in Customer and Supplier. The answer table is this:

Customer	Supplier	AvgOfAmount
Johnson	Harrison	£453.78
Jones	Ford	£453.00
Jones	Harrison	£3,421.00
Simpson	Harrison	£235.67
Smith	Ford	£82.78
Smith	Harrison	£235.67

which raises another interesting question: how can you tell how many records are actually contributing to each group? One answer (but by no means the only one) is:

```
SELECT Count(*) AS NumberInGroup,
Customer, Supplier, AVG(Amount) AS
AvgOfAmount
FROM SALES
GROUP BY Customer, Supplier;
```

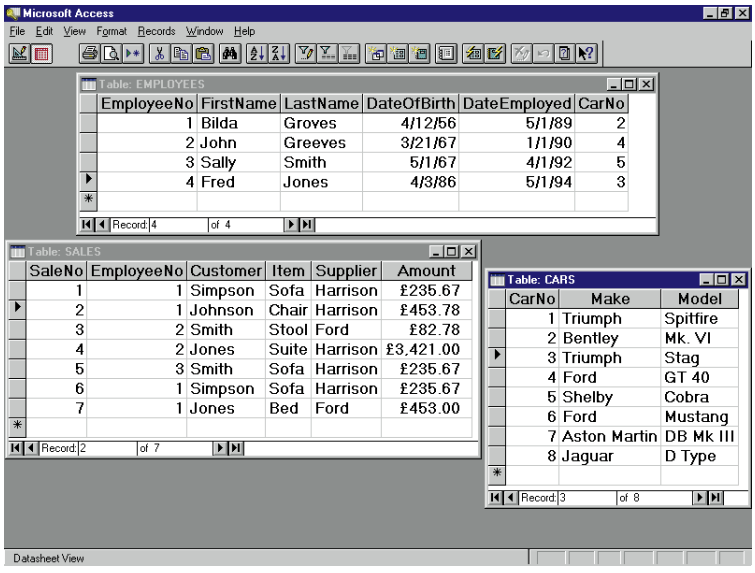
The only addition is the "Count(*) AS NumberInGroup" bit which simply says that the number of records in each group should be counted (Fig 5).

We could equally well use:

```
SELECT Count(Customer) AS
NumberInGroup, Customer, Supplier,
AVG(Amount)
AS AvgOfAmount
FROM SALES
GROUP BY Customer, Supplier;
```

which returns the same answer table.

GROUP BY is an incredibly powerful tool



The tables used in my examples

and it can be made even more so with the addition of HAVING.

GROUP BY and HAVING — Collecting information together

Whereas the GROUP BY clause puts records into logical groupings, the HAVING clause allows you to select the groups that you want to see based on values which appertain to that group. Consider the example given above.

```
SELECT Customer, Supplier,
AVG(Amount) AS AvgOfAmount
FROM SALES
GROUP BY Customer, Supplier;
```

Customer	Supplier	AvgOfAmount
Johnson	Harrison	£453.78
Jones	Ford	£453.00
Jones	Harrison	£3,421.00
Simpson	Harrison	£235.67
Smith	Ford	£82.78
Smith	Harrison	£235.67

Suppose, now the records are grouped in this way, that we are only interested in the groups where the average amount is £250 or more? The foolish solution is:

```
SELECT Customer, Supplier,
AVG(Amount) AS AvgOfAmount
FROM SALES
GROUP BY Customer, Supplier
ORDER BY AVG(Amount);
```

Customer	Supplier	AvgOfAmount
Smith	Ford	£82.78
Smith	Harrison	£235.67
Simpson	Harrison	£235.67
Jones	Ford	£453.00
Johnson	Harrison	£453.78
Jones	Harrison	£3,421.00

which, although it renders the desired values easy to find, nevertheless still leaves the job of actually locating them, up to the user. A much better solution would be:

```
SELECT Customer, Supplier,
AVG(Amount) AS AvgOfAmount
FROM SALES
GROUP BY Customer, Supplier
HAVING AVG(Amount) >= 250;
```

Customer	Supplier	AvgOfAmount
Johnson	Harrison	£453.78
Jones	Ford	£453.00
Jones	Harrison	£3,421.00

You can, of course, still order the groups:

```
SELECT Customer, Supplier,
AVG(Amount) AS AvgOfAmount
FROM SALES
GROUP BY Customer, Supplier
HAVING AVG(Amount) >= 250
ORDER BY AVG(Amount);
```

Customer	Supplier	AvgOfAmount
Jones	Ford	£453.00
Johnson	Harrison	£453.78
Jones	Harrison	£3,421.00

Working with multiple tables

So far, we have looked at using the SELECT statement with a single table. Clearly, since the relational model encourages us to split complex data into separate tables we will often find it necessary to recover data from two or more tables. To do this, we have to use the SELECT statement to draw data from both and the WHERE clause to form the joins.

Before we do, let's try querying the tables without using the WHERE clause.

```
SELECT SALES.Customer,
EMPLOYEES.LastName, SALES.Amount
FROM SALES, EMPLOYEES;
```

produces the data shown in Fig6.

Note that this SQL statement includes, for the first time, the table names when fields are being specified. Up to this point our SELECT statements have referred to single tables. Since field names within a single table must be unique, the field name alone allowed us to unambiguously identify the fields. However, field names can (and often are) shared by different tables. For example, both SALES and EMPLOYEES have a field called EmployeeNo. Therefore, the only way to identify a precise field uniquely is to use the table name as well. SQL syntax typically has the table name first in upper case, followed by a dot, followed by the field name in lower case.

SQL allows you to substitute temporary synonyms for table names:

```
SELECT S.Customer, E.LastName,
S.Amount
FROM SALES S, EMPLOYEES E;
```

which can shorten statements considerably but also tends to makes them less readable.

Note that the synonyms are defined in the FROM clause, but can still be used in the SELECT clause which tells you something about the way in which the SQL statement is read by the RDBMS.

To return to the multiple table query, if we were to add a WHERE clause as shown here:

```
SELECT SALES.Customer,
EMPLOYEES.LastName, SALES.Amount
FROM SALES, EMPLOYEES
WHERE SALES.EmployeeNo =
EMPLOYEES.EmployeeNo;
```

we get:

Customer	LastName	Amount
Simpson	Groves	£235.67
Johnson	Groves	£453.78
Simpson	Groves	£235.67
Jones	Groves	£453.00
Smith	Greeves	£82.78
Jones	Greeves	£3,421.00
Smith	Smith	£235.67

Referring to the base tables shows that this is a more useful answer table than the previous one.

How it works, and what it's doing, will be revealed next month. ■

Fig 6

Customer	LastName	Amount
Simpson	Groves	£235.67
Johnson	Groves	£453.78
Smith	Groves	£82.78
Jones	Groves	£3,421.00
Smith	Groves	£235.67
Simpson	Groves	£235.67
Jones	Groves	£453.00
Simpson	Greeves	£235.67
Johnson	Greeves	£453.78
Smith	Greeves	£82.78
Jones	Greeves	£3,421.00
Smith	Greeves	£235.67
Simpson	Greeves	£235.67
Jones	Greeves	£453.00
Simpson	Smith	£235.67
Johnson	Smith	£453.78
Smith	Smith	£82.78
Jones	Smith	£3,421.00
Smith	Smith	£235.67
Simpson	Smith	£235.67
Jones	Smith	£453.00
Simpson	Jones	£235.67
Johnson	Jones	£453.78
Smith	Jones	£82.78
Jones	Jones	£3,421.00
Smith	Jones	£235.67
Simpson	Jones	£235.67
Jones	Jones	£453.00

Fig 5

NumberInGroup	Customer	Supplier	AvgOfAmount
1	Johnson	Harrison	£453.78
1	Jones	Ford	£453.00
1	Jones	Harrison	£3,421.00
2	Simpson	Harrison	£235.67
1	Smith	Ford	£82.78
1	Smith	Harrison	£235.67

NEW!

Question time

Which database querying tool is text-based and reactionary, yet immensely adaptable and even a boon in some social circles? Why, SQL of course. In the first part of our new tutorial, Mark Whitehorn introduces the basics.

SQL stands for Structured Query Language, which is referred to either as its individual letters or is called "Sequel". It appears as if the former reference is more common in the UK and the latter in the US, but as the two are interchangeable don't let it be a cause of anxiety.

Despite many similarities to C, Pascal, BASIC *et al*, SQL is not a programming language. It is a data access language or data sub-language. As such, it is a very restricted language which deals only with how tables of data can be manipulated. It lacks many of the other features (such as the ability to write information to a particular place on the screen) which characterise a full programming language.

Using SQL

SQL is often described as a standard, but when you actually start using it you find that, like many standards, it's not as standard as all that.

The examples given here are in a generic form of SQL: you may well find discrepancies depending on the actual version used. For example, the generic DISTINCT becomes DISTINCTROW in Microsoft's Access. Having said that, the differences are not great, and should not pose serious problems.

The name itself ("SQL") is somewhat misleading as it implies that this sub-language is concerned exclusively with querying. In fact, the language is sufficiently rich to allow the user to perform many other operations such as creating tables, but it remains true that the most common usage of the language is to ask questions of a database. This part of the language comprises the Data Manipulation Language (DML) statements of SQL.

DML statements are, by convention, written in UPPERCASE. The first ones we'll look at are SELECT, FROM, DISTINCT and WHERE. The sample tables shown in Fig 1 will be used for the examples.

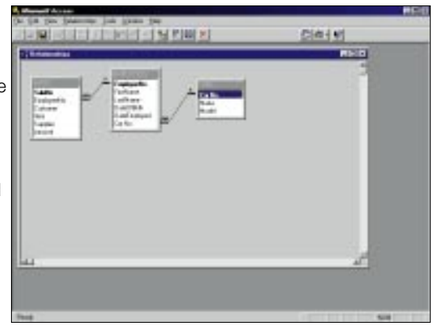


Fig 1 **The sample files used in my examples**

SELECT & FROM

The first statement, SELECT, is used to extract a collection of fields from a given table. FROM simply directs attention to the table in question. Therefore, the statement

```
SELECT SaleNo, Item, Amount
FROM SALES;
```

will yield the following;

SaleNo	Item	Amount
1	Sofa	£235.67
2	Chair	£453.78
3	Stool	£82.78
4	Suite	£3,421.00
5	Sofa	£235.67
6	Sofa	£235.67
7	Bed	£453.00

SQL doesn't eliminate duplicates by default, so:

```
SELECT Item, Amount
FROM SALES;
```

will yield

Item	Amount
Sofa	£235.67
Chair	£453.78
Stool	£82.78
Suite	£3,421.00
Sofa	£235.67
Sofa	£235.67
Bed	£453.00

DISTINCT

You can force SQL to remove the duplicates by using the statement DISTINCT, which dictates that all rows in the answer table must be unique. The query

```
SELECT DISTINCT Item, Amount
FROM SALES;
```

produces:

Item	Amount
Bed	£453.00
Chair	£453.78
Sofa	£235.67
Stool	£82.78
Suite	£3,421.00

WHERE

SELECT lets you choose the fields with which to work, and WHERE lets you choose the records.

```
SELECT Item, Amount
FROM SALES
WHERE Item = 'Sofa';
```

produces

Item	Amount
Sofa	£235.67
Sofa	£235.67
Sofa	£235.67

while

```
SELECT Item, Amount
FROM SALES
WHERE Item = 'Sofa' AND Customer =
'Smith';
```

yields

Item	Amount
Sofa	£235.67

All sorts of variations are already possible, combining SELECT and WHERE statements: as you can see from the last example, WHERE clauses can contain conditions.

Fig 2 One record found

EmployeeNo	FirstName	LastName	Date of Birth	DateEmployed
2	John	Greeves	21 March 1967	01 January 1990

Conditions

We'll digress here to cover the range of Conditions that are acceptable within a WHERE clause. Conditions typically consist of logical expressions which can be evaluated for truth; in other words, they are checked to discover whether they are true or false.

Thus if we use the SQL statement

```
SELECT EmployeeNo, FirstName,
LastName, DateOfBirth, DateEmployed
FROM EMPLOYEES
WHERE EmployeeNo = 2;
```

then we can expect the RDBMS to examine every record in the EMPLOYEE table, and place in the answer table only those records for which the condition

```
WHERE EmployeeNo = 2
```

is true. As you'd hope, this is only true for one record (Fig 2).

A condition is constructed from operators such as those shown in Fig 3.

The logical operators in Fig 4 have a lower priority than those above and are therefore processed after them, unless brackets are used to alter precedence.

The following SQL statement asks for a table of the items and amounts from the Sales table for sale numbers greater than six:

```
SELECT Item, Amount
FROM SALES
WHERE SaleNo > 6;
```

Item	Amount
Bed	£453.00

while this one only wants to see records for sofas for sale numbers greater than six;

```
SELECT Item, Amount
FROM SALES
WHERE Item = 'Sofa' AND SaleNo > 6;
```

There are none.

This next statement asks for all records for sofas, suites and beds, regardless of sale number:

```
SELECT Item, Amount
FROM SALES
WHERE Item IN ('Sofa', 'Suite',
'Bed');
```

Item	Amount
Sofa	£235.67
Suite	£3,421.00
Sofa	£235.67
Sofa	£235.67
Bed	£453.00

and this one adds a condition which specifies records for the same three pieces of furniture with sale numbers greater than six:

```
SELECT Item, Amount
FROM SALES
WHERE Item IN ('Sofa', 'Suite',
'Bed') AND SaleNo > 6;
```

Item	Amount
Bed	£453.00

Conditions are nothing if not logical, and rendering a series of conditions into plain English is a good way of understanding what it will do in practice.

ORDER BY

Another useful command is ORDER BY. It gives you control over the order in which records appear in the answer table generated by the query. You specify the field by which you want records ordered, as in the following statement:

```
SELECT Item, Amount
FROM SALES
WHERE Item = 'Sofa'
ORDER BY SaleNo;
```

where the records are ordered by the number of each sale, with the default being in ascending order. If you feel you want to specify this, the command is ASC, as shown below:

```
SELECT Item, Amount
FROM SALES
WHERE SaleNo > 6
ORDER BY Item ASC;
```

It's a perfectly acceptable statement, but it's tautological. The next statement:

```
SELECT Item, Amount
FROM SALES
WHERE SaleNo > 6
ORDER BY Item DESC;
```

will produce exactly the same data but will be sorted differently, as DESC, as you'll have gathered, sorts records in descending order. You can use sorts in both directions:

```
SELECT Item, Customer, SaleNo,
Amount
FROM SALES
WHERE SaleNo > 0
ORDER BY Customer ASC, Amount DESC;
```

Note that Amount doesn't *have* to be in the SELECT statement to be used for sorting the records in the answer table, although this would often be the case.

This will sort the customer records in ascending order, with the amounts each customer has spent shown in descending order.

Item	Customer	SaleNo	Amount
Chair	Johnson	2	£453.78

Fig 3 Operators

Symbol	Meaning	Example	Notes	Records returned from Employee table
=	Equal to	EmployeeNo = 2		1
>	Greater than	EmployeeNo > 2		2
<	Less than	EmployeeNo < 2		1
<>	Not equal to	EmployeeNo <> 2		3
>=	Greater than or Equal to	EmployeeNo >= 2		3
<=	Less than or Equal to	EmployeeNo <= 2		2
IN	Equal to a value within a collection of values	EmployeeNo IN (2, 3, 4)		3
LIKE	Similar to	LastName LIKE "Gr*"	Finds Greeves and Groves. Uses wildcards. Wild cards vary between SQL implementations.	2
BETWEEN…AND	Within a range of values, including the two values which define the limits	EmployeeNo BETWEEN 2 AND 4	Equivalent to: EmployeeNo IN (2, 3, 4)	3
IS NULL	Field does not contain a value	DateEmployed IS NULL		0

p254 ➢

Fig 4 Logical operators

Symbol	Meaning	Example	Notes	Records returned from Sales table
AND	Both expressions must be true in order for the entire expression to be judged true	SaleNo > 3 AND Customer = "Smith"	AND is evaluated before OR	1
OR	If either or both expressions are true, the entire expression is judged to be true	SaleNo > 3 OR Customer = "Smith"	AND is evaluated before OR	5
NOT	Inverts Truth	SaleNo NOT IN (2, 3, 4)	(just as well it isn't available for the real world!)	4

Suite	Jones	4	£3,421.00
Bed	Jones	7	£453.00
Sofa	Simpson	6	£235.67
Sofa	Simpson	1	£235.67
Sofa	Smith	5	£235.67
Stool	Smith	3	£82.78

Wild cards

Wild cards are used in SQL much as they are used elsewhere, for occasions where you want a range of data that fits a certain pattern. The variation below is not uncommon:

```
SELECT *
FROM SALES
WHERE SaleNo > 1;
```

In this case, the * symbol is used as a wild card, meaning "all Fields".

Fig 5 Records from SALES table

Sale No.	Employee No.	Customer	Item	Supplier	Amount
3	2	Smith	Stool	Ford	£82.78
4	2	Jones	Suite	Harrison	£3,421.00
5	3	Smith	Sofa	Harrison	£235.67

Sub-queries

The use of conditions can be expanded into sub-queries to add further refinement to queries. In the following example:

```
SELECT Customer
FROM SALES
WHERE EmployeeNo IN
   (SELECT EmployeeNo
    FROM EMPLOYEES
    WHERE DateEmployed > 12/5/89);
```

the statement inside brackets is known as a sub-query and would work perfectly happily as a query all on its own. (Incidentally, this is a good case where dialects of SQL differ. Access requires that the date be wrapped up in # symbols, thus the last line would read as

```
WHERE DateEmployed > #12/5/89#)
```

Any operation performed on a table (or tables) results in another table — one containing the answer. This is termed "closure" and it is an invariable rule. The aforementioned sub-query produces an answer table, shown here:

EmployeeNo
2
3
4

By looking at the answer table generated by the sub-query, we can see that the original statement in its full form can be simplified to:

```
SELECT Customer
FROM SALES
WHERE EmployeeNo IN (2,3,4)
```

and the records from the SALES table for which this is true are shown in Fig 5.

So the query actually yields:

Customer
Smith
Jones
Smith

We can eliminate the duplicate records by adding the word Distinct to the first line of the SQL command.

■ There will be more on honing your SQL skills in part 2 of this workshop next month.