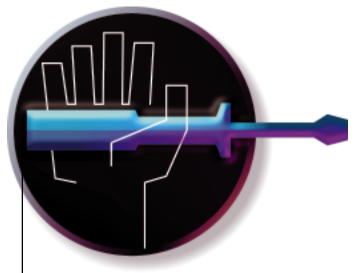

State of the **union**

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
```

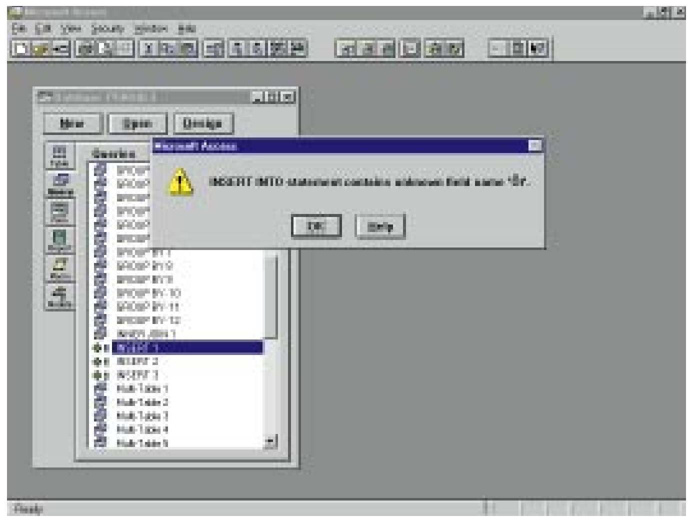

Fig 3 **Error message generated when the first INSERT command is used too frequently!**

```
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

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