SQL in brief

NOTES ON QUERIES

ULRICH WOLF

Getting started with relational databases can be easier than you might think. Once you have created your data model it has to be converted to SQL. The following two pages will help SQL novices to do just that.

The "Structured Query Language" SQL is the lingua franca of the database world. Almost all relational databases understand SQL these days, and it is not difficult to learn, either. The language originated in the IBM research laboratories, where it was developed in the seventies. There has been a common standard from the American standardisation authorities, ISO and ANSI, since 1992: SQL-92. Most databases introduced in this magazine conform to a subset, Entry Level SQL-92.

There are two aspects to the tasks of database languages: creation of data structures (data definition) and data manipulation. Purists therefore distinguish between "Data Definition Languages (DDL)" and "Data Manipulation Languages (DML)". SQL however is both.

The following article will therefore be covering both aspects of the language. We will be take a data model as an example, and we will be converting using PostgreSQL. The version used must be 7.0 or later, as earlier versions cannot cope with foreign keys, which are a part of this data model.

To enable us to communicate with the database we need a front end. In the simplest case that will be the program *psql*; however, simple graphical or Web-based interfaces exist as well. Whichever we are going to use, we have to start by building a data structure, that is to say, by creating tables or relations. We will now create the simplest table:

```
CREATE TABLE vendor (
        vid INTEGER,
        vname VARCHAR(40),
        vaddress VARCHAR(100),
        vcity VARCHAR(40),
        vzip VARCHAR(7),
        vtel VARCHAR(40),
        PRIMARY KEY (vid));
```

As you can see, the syntax is simple: the name ("attribute" in database language) is followed by the data type, and optionally by the field length. Afterwards we have the option of determining which attribute is to be used as the primary key. None of the entries are case-sensitive, upper case is used here only to differentiate between command syntax and attributes. Each SQL command ends with a semicolon and can extend over several lines. If it does, the *psql* prompt will change from *database name*=> to *database name (->*, to indicate that the command is not complete.

In the relation pc below, a foreign key indicated by the keyword *references* appears in addition to the primary key. We have also introduced the data type *DATE*:

```
CREATE TABLE pc (
        pcid INTEGER,
        vid INTEGER REFERENCES vendor,
        type VARCHAR,
        purdate DATE,
        PRIMARY KEY (pcid));
```

As we are only concerned with syntax, these two tables should be sufficient. Now it's time to provide some content for them. In most cases the command *INSERT INTO* is used for this purpose:

Listing 1: Two small tables with simple queries

```
osadmin=> select * from pc;
pcid     | vid      |       type          |  purdate
---+---+-------------+------
   1     |   4      | Desktop PC          | 20/01/1999
   2     |   4      | Laptop              | 24/12/2000
   3     |   5      | Sun server          | 06/08/1992
(3 rows)
osadmin=> select * from vendor;
 vid     |  vname   |    vaddress         |   vcity     | vzip          |   vtel
---+-----+----------+------+---+------
   1     | Red Hat  | 10 Alan Turing Road | Guildford   | GU2 7YF       | 01483300169
   2     | SuSE     | Theobald Street     | Borehamwood | WD6 4PJ       | 02083874088
   3     | VA Linux | Whitehill Way       | Swindon     | SN5 6QR       | 08702412813
   4     | IBM      | PO Box 41           | Portsmouth  | PO6 3AU       | 0990426426
   5     | Sun      | Guillemont Park     | Camberley   | GU17 9QG      | 01276451440
(5 rows)
```

```
INSERT INTO vendor VALUES (
  1,'Red Hat','10 Alan Turing Road',
  `Guildford','GU2 7YF
  '01483300169');
```

Postgres also provides the *COPY* command, which unfortunately does not conform to the standard. This is an easy way of reading in data from ASCII files that have for example been exported from a spreadsheet.

Ultimately it is not data entry options that distinguish databases, but the diversity of query facilities, in the case of SQL using the *SELECT* command. In its simplest form *SELECT * FROM table;* returns a complete table with all computers and all vendors, as in Listing 1. That is of course not sufficient. Listing 2 therefore presents a more useful query, which returns all computers manufactured by Sun, together with the purchase dates. As you can see, there is only one. Nested queries are possible in almost all database systems. In MySQL they have to be written differently; this is described in the comparison between Postgres and MySQL elsewhere in this issue.

The *SELECT* command is the heart of SQL and offers an enormous range of possibilities: sorting by specific criteria, grouping of results, pattern matching with *LIKE* and many more, which are beyond the scope of this article. We will just give one more short example of how entries can be amended with a combination of *UPDATE* and *SELECT*, after all, that is part and parcel of database functionality. In the admittedly somewhat contrived case that we only remember the purchase date of the server, but now the telephone number of the vendor, who we do not know has changed, we can update it as shown in Listing 3.

Finally two options for getting rid of data. Complete tables are removed without any fuss using *DROP tables;*, individual rows with *DELETE* where conditions can be set in the same way as for *UPDATE*. A simple example: The Notebook has been sold and we are going to remove its record from the database:

```
DELETE FROM pc
   WHERE type = 'Laptop';
```

There are a number of pitfalls to watch out for, especially something called cascading deletion when using foreign keys. Nevertheless, these few glimpses of SQL should be enough for you to take your first steps, and then get more deeply involved if necessary. The subject can be more exciting than you might think, and SQL has hardly become unfashionable, despite object orientation and XML. There is no shortage of literature on SQL. ■

Listing 2: Using SELECT

```
osadmin=> select type, purdate from pc where vid =
osadmin-> (select vid from vendor
osadmin(> where vname = `Sun');
         type     |  purdate
-------------+-----
 Sun server       | 06/08/1992
(1 row)
```

Listing 3: UPDATE combined with SELECT

```
osadmin=> update vendor SET vtel = '01252421730' WHERE
osadmin-> vid =
osadmin-> (SELECT vid from pc where purdate = '06/08/1992');
UPDATE 1
osadmin=> select * from vendor
osadmin-> ;
 vid     |  vname   |       vaddress       |   vcity     | vzip       |   vtel
---+-----+----------+-------+---+------
   1     | Red Hat  | 10 Alan Turing Road  | Guildford   | GU2 7YF    | 01483300169
   2     | SuSE     | Theobald Street      | Borehamwood | WD6 4PJ    | 02083874088
   3     | VA Linux | Whitehill Way        | Swindon     | SN5 6QR    | 08702412813
   4     | IBM      | PO Box 41            | Portsmouth  | PO6 3AU    | 0990426426
   5     | Sun      | Guillemont Park      | Camberley   | GU17 9QG   | 01252421730
(5 rows)
```

Info

SQL Tutorial: http://dblabor.f4.fhtw-berlin.de/morcinek/sqltutor

PostgreSQL Homepage: http://www.postgresql.org

For creating online Postgres databases and "playing": http://pgdemo.acucore.com

■