

SQL: Database Modifications

So far, we have focused on queries, but of course we also need to be able to change the content of a database. Let's look at how to insert and delete rows, and also how to change rows that are already in a table.

Insertion of literal values

We've already seen examples where we literally give the values to be inserted. For example, we define the schema for a small table below. (We'll learn the full details of the Data Definition Language part of SQL shortly, but this example is simple enough to make intuitive sense.)

```
csc343h-prof=> CREATE TABLE Ages(name TEXT, age INT);  
CREATE TABLE
```

Here we insert literal values into the table:

```
csc343h-prof=> INSERT INTO Ages VALUES  
csc343h-prof-> ('Amna', 21), ('Zach', 25), ('Miriam', NULL), ('Ben', 0);  
INSERT 0 4
```

We can confirm that the values are there:

```
csc343h-prof=> SELECT *  
csc343h-prof-> FROM Ages;  
  name | age  
-----+-----  
 Amna  |  21  
 Zach  |  25  
Miriam |  
 Ben   |   0  
(4 rows)
```

Insertion of computed values

Another way to insert rows is to *compute* the values to be inserted. For example, here we find everyone who has taken a first-year course and insert them into the table using 19 for their age (pretty silly):

```
csc343h-prof=> INSERT INTO Ages  
csc343h-prof-> (SELECT DISTINCT firstname, 19 AS age  
csc343h-prof(> FROM Student JOIN Took USING (sID)
```

```
csc343h-prof(> JOIN Offering USING (oID)
csc343h-prof(> WHERE cnum <= 199);
INSERT 0 5
```

We can see the five new rows:

```
csc343h-prof=> SELECT *
csc343h-prof-> FROM Ages;
  name  | age
-----+-----
Amna    |  21
Zach    |  25
Miriam  |
Ben     |   0
Leilani |  19
William |  19
Afsaneh |  19
Avery   |  19
Homer   |  19
(9 rows)
```

Notice that **INSERT INTO** added the new rows to what was already in the table.

Insertion with default values

If we are inserting rows and don't have a value for one or more of the attributes, we can omit it from the insertion and the DBMS will use a default value in that column. We accomplish this by naming the attributes that we *are* providing values for. We could have used this feature in the previous example rather than assume that all first-year students are 19.

Let's look at a new example. Suppose we want to create a table to track invitations and RSVPs for an event for scholarship winners. We might define the table as follows:

```
csc343h-prof=> CREATE TABLE Invite (
csc343h-prof(>   name TEXT,
csc343h-prof(>   campus TEXT DEFAULT 'StG',
csc343h-prof(>   email TEXT,
csc343h-prof(>   age INT);
CREATE TABLE
```

(Notice that we have defined a default value for **campus**.)

In the following query, we provide values for **name** and **email**, and let the DMBS fill in default values for the other columns:

```
csc343h-prof=> INSERT INTO Invite(name, email)
```

```
csc343h-prof-> (SELECT firstname, email
csc343h-prof(> FROM Student
csc343h-prof(> WHERE cgpa > 3.4);
INSERT 0 2
```

Since the `campus` attribute has a default value (`StG`) defined in its schema, this value is used; and since `age` does not, `NULL` is used for it:

```
csc343h-prof=> SELECT *
csc343h-prof-> FROM Invite;
  name   | campus | email | age
-----+-----+-----+-----
William | StG    | will@cs | 
Leilani | StG    | lani@cs | 
(2 rows)
```

Deletion

To delete some rows from a table, we simply specify the condition that must be satisfied in order for a row to be deleted.

For example, let's find all the failing grades:

```
csc343h-prof=> SELECT *
csc343h-prof=> FROM Took
csc343h-prof=> WHERE grade < 50;
  sid | oid | grade
-----+-----+-----
99132 | 14 | 39
 157 | 11 | 39
11111 | 17 | 46
11111 | 14 | 40
11111 | 15 | 0
11111 | 16 | 17
11111 | 34 | 45
(7 rows)

csc343h-prof=> -- There are 57 rows in total in the table, including
csc343h-prof=> -- the 7 that we found above containing failing grades.
csc343h-prof=> SELECT count(*)
csc343h-prof=> FROM Took;
 count
-----
    57
(1 row)
```

Here, we delete those 7 rows that have failing grades:

```
csc343h-prof=> DELETE FROM Took
csc343h-prof-> WHERE grade < 50;
DELETE 7

csc343h-prof=> -- There are 50 rows left.
csc343h-prof=> SELECT count(*)
csc343h-prof=> FROM Took;
count
-----
      50
(1 row)
```

To delete *all* rows from a table, we just omit the **WHERE** condition:

```
csc343h-prof=> DELETE FROM Took;
DELETE 50

csc343h-prof=> -- There are no rows left.
csc343h-prof=> SELECT count(*)
csc343h-prof-> FROM Took;
count
-----
      0
(1 row)

csc343h-prof=> -- There really aren't!
csc343h-prof=> SELECT *
csc343h-prof-> FROM Took;
sid | oid | grade
-----+-----+-----
(0 rows)
```

Updates

To change the values of certain attributes in certain rows we use an **UPDATE** statement with this general form:

UPDATE *table* **SET** *list of attribute assignments* **WHERE** *condition on tuples*;

For example, here we update the (at most) one row with **sID** 999999:

```
csc343h-prof=> SELECT * FROM Student;
sid | firstname | surname | campus | email | cgpa
-----+-----+-----+-----+-----+-----
99132 | Avery     | Marchmount | StG    | avery@cs | 3.13
98000 | William  | Fairgrieve | StG    | will@cs | 4.00
00000 | A. Smith  | A. Smith   | UTSC   | a-smith@ | 2.00
```

```

99999 | Arsanen | Ali | UTM | aali@cs | 2.98
157 | Leilani | Lakemeyer | UTM | lani@cs | 3.42
11111 | Homer | Simpson | StG | doh@gmail | 0.40
(5 rows)

```

```

csc343h-prof=> UPDATE Student
SET campus = 'UTM'
WHERE sID = 99999;
UPDATE 1

```

```

csc343h-prof=> SELECT * FROM Student;
  sid | firstname | surname | campus | email | cgpa
-----+-----+-----+-----+-----+-----
99132 | Avery | Marchmount | StG | avery@cs | 3.13
98000 | William | Fairgrieve | StG | will@cs | 4.00
157 | Leilani | Lakemeyer | UTM | lani@cs | 3.42
11111 | Homer | Simpson | StG | doh@gmail | 0.40
99999 | Afsaneh | Ali | UTM | aali@cs | 2.98
(5 rows)

```

(We know there can't be more than one row updated because `sID` is the primary key of this table.)

In this example, we update possibly many rows. Before we do the update, let's see all the grades of one particular student:

```

csc343h-prof=> SELECT *
csc343h-prof=> FROM Took
csc343h-prof=> WHERE sID = 99132;
  sid | oid | grade
-----+-----+-----
99132 | 1 | 79
99132 | 16 | 98
99132 | 31 | 82
99132 | 11 | 99
99132 | 14 | 39
99132 | 15 | 62
99132 | 34 | 75
99132 | 5 | 98
(8 rows)

```

Now we raise several of this student's grades to 100.

```

csc343h-prof=> UPDATE Took
csc343h-prof=> SET grade = 100
csc343h-prof=> WHERE sID = 99132 AND grade > 95 AND grade < 100;
UPDATE 3

```

And here we see that the updates were indeed made:


```

csc343h-prof=> SELECT * FROM Took WHERE sid = 99132;

```

```
CSC343H1-prot=> SELECT * FROM took WHERE stu = 99132,  
  sid | oid | grade  
-----+-----+-----  
99132 | 1 | 79  
99132 | 31 | 82  
99132 | 14 | 39  
99132 | 15 | 62  
99132 | 34 | 75  
99132 | 16 | 100  
99132 | 11 | 100  
99132 | 5 | 100  
(8 rows)
```

Further details

For further information about database modifications, see the [chapter on Data Manipulation](https://www.postgresql.org/docs/9.5/static/dml.html)  [\(<https://www.postgresql.org/docs/9.5/static/dml.html>\)](https://www.postgresql.org/docs/9.5/static/dml.html) in the PostgreSQL Documentation.