CS1204: Databases for Digital Humanities

Steven Prestwich
Department of Computer Science
National University of Ireland at Cork
tel. +353 21 420 5911
s.prestwich@cs.ucc.ie

Modifying tables

We know how to create and delete tables, and how to query them to see what data they contain. But how do we add information to them, that is type in attribute values? Also we need to delete information and modify it. (You'll also need this for labs.)

Of course not every user can modify a database (for example you can't modify the database used in labs) — only users with the right permissions.

There are 3 types of modification:

- insert rows into a table
- delete rows from a table
- update components (attribute values) of rows in a table

Let's see how to do each.

Insertion

We type

```
INSERT INTO R(A1,...,An) VALUES (v1,...,vn)
```

where `R` is the table, `A1...An` are its attribute names, and `v1...vn` are the values we want the attributes to have. For example

```
INSERT INTO StarsIn(title,year,name) VALUES
   ('The Matrix',1999,'Keanu Reeves')
```

Actually, as we're typing in values for all the attributes we don't need to list the attribute names. We can just type

```
INSERT INTO StarsIn VALUES ('The Matrix',1999,'Keanu Reeves')
```

Of course we have to make sure that we type in the right number of values, in the right order, as given in the table definition (when we typed `CREATE TABLE`).

Sometimes we don't want to specify all the attribute values in the inserted row. Then we need to tell SQL which attributes we're specifying values for. For example

```
INSERT INTO StarsIn(title,year) VALUES ('The Matrix',1999)
```

The value of the name attribute will then be a special value called `NULL`, meaning "no data" — we'll come back to this later.

Deletion

To delete a row from a table we type

```
DELETE FROM R WHERE (some condition);
```

This deletes every row from table `R` that satisfies the condition. For example

```
DELETE FROM StarsIn
WHERE title='The Matrix' AND year=1999 AND name='Keanu Reeves';
```

The database now no longer contains the information that Keanu Reeves starred in The Matrix.

The condition could specify many rows to be deleted. For example to delete from

```
FilmExec(name,address,cert,netWorth)
```

all executives worth less than a million:

```
DELETE FROM FilmExec WHERE netWorth<1000000;
```

If no executive is worth less than a million then no rows will be deleted.

Updates

Finally, here's how to modify the contents of rows in a table. We type

```
UPDATE R SET (attribute=value,...) WHERE (some condition);
```

This finds all rows in `R` satisfying the condition and assigns the given values to the given attributes. For example suppose someone entered the wrong year when they added Harrison Ford in Star Wars. We want to change this to 1978:

```
UPDATE StarsIn SET year = 1978 WHERE title='Star Wars' AND name='Harrison Ford';
```

But suppose the same person entered the wrong years for all actors in Star Wars. We can update all these rows at once:

```
UPDATE StarsIn SET year = 1978 WHERE title='Star Wars';
```

This would change the year of Star Wars to 1978 in every `StarsIn` row that mentions one of its actors.

If the `WHERE` clause matches no rows (for example if Star Wars isn't in the table at all) then no rows will be updated.

Lab assignment 1

Here's the first lab assignment. First we'll create a database for an auctioneer (estate agent, realtor) with data on houses, apartments, branches of the auctioneer, staff, etc. I'll give a description of each table you should create. This is for practice and to get a feel for SQL, and you won't be marked on it. But the next thing will be to write some queries for this database, which will be the assignment.

Type `CREATE TABLE` with the name of the table, each attribute you need, and the attribute type. Here are the tables.

`BRANCH(BranchNo, Street, Area, City, TelNo)`. This has information on the different branches of the estate agent: a branch number (B1, B2, B3...), the street it's on (`'123 Fake St'`), the area (`'Blackrock'`), the city (`'Cork'`) and the branch telephone number (`'021-123-4567'`). An example row:

```
('B2', 'South Mall', 'city centre', 'Cork', '021-123-4567')
```

You can make each attribute a string type, with enough characters to fit in the data. For example a city might be up to 10 characters long: `city CHAR(10)`.

`STAFF(StaffNo, FName, LName, Address, TelNo, Job, Gender, Salary, BranchNo)`. This is information about the staff in the estate agent: a staff number for each (`'SL21'`, `'SG16'`...), their first name (`'John'`), their last name (`'Smith'`), their address, their telephone number, their job title, their gender (`'M'`, `'F'`), their salary (`25000`), and the branch number where they work (`B2` etc). An example row:

```
('SL33', 'Mary', 'Jones', '10 New St, Ballincollig',
 '021-765-4321', 'manager', 'F', 28000, 'B2')
```

These are strings, apart from salary which is an integer (INT).

`PROPERTY(PropertyNo, Street, Area, City, Type, Rooms, Rent, OwnerNo, StaffNo, BranchNo)`. This is information about the properties: a property number for each house or apartment (`'PA14'`, `'PL94'`...), its address (street, area, city), its type (`'house'`, `'apartment'`), how many rooms it has (an integer), the rent (an integer), an owner number (which I'll describe later), the staff number of the agent handling it (the same staff number as in the `STAFF` table), and the branch number of the branch handling it (the same branch number as in the `BRANCH` table). An example row:

```
('PA14', 'Old St', 'Blackrock', 'Cork', 'apartment', 3, 450,
 'CO46', 'SL33', 'B2')
```

All these can be strings except for number of rooms and the rent (both integers).

`RENTER(RenterNo, FName, LName, Address, TelNo, TypePreference, MaxRent)`. Information about the renters: a number for each (`'CR76'`, `'CR56'`...), names, addresses, telephone numbers, which type they prefer (`'house'`, `'apartment'`: the same options as in `PROPERTY`), and the maximum rent they're prepared to pay (an integer). An example row:

```
('CR76', 'Mike', 'Donovan', '8 Barrack St, Cork', '021-333-4444',
 'house', 550)
```

All strings except for maximum rent (integer).

`OWNER(OwnerNo, FName, LName, Address, TelNo)`. Information about the owners of the properties: a owner number (`'CO46'`, `'CO93'`...), names, addresses, telephone numbers. An example row:

```
('CO46', 'Mary', 'McAleese', '1 Main St, Dublin', '01-222-3333')
```

All these values are strings.

`VIEWING(RenterNo, PropertyNo, Date, Comment)`. Information about what happened when properties were viewed: which renter viewed which property, on which date, and what their comments were (`'too small'`,`'no dining room'`...). An example row:

```
('CR76', 'PA14', '10-04-2000', 'Damp on walls')
```

All strings except the date: it's better to use the SQL `DATE` type.

Once you have these tables you can start entering data. After that, the assignment is to access this database, and get some information out of it using queries — the exercise is really to write the queries. Here are the queries you have to write:

1. Find the name and address of staff members earning less than 20000.
2. Find the name and address of male staff members earning less than 20000.
3. Find all information on female staff members who do not have a telephone number.
4. Find the name and address of renters requesting an apartment for less than 500 (euros per month).
5. Find the name and address of renters living in the Cork area (phone number starting 021) requesting an apartment for less than 500.
6. Find the property number and address of apartments in Cork having more than 3 rooms and costing between 400 and 500.
7. Find the property number, address and rent of apartments handled by either branch B3 or B7. Sort the result by city and by area.
8. Find all information on property owner Carol Farrell.

Deadline. To be announced. To submit it, you should just upload a list of SQL queries to Canvas (please use plain text, Word or pdf):

```
(1) SELECT...FROM...WHERE;
(2) SELECT...
...
```

I don't need to see the results of the queries, or the contents of your tables: these are just for your practice.

Here are some tips:

- All these are pretty straightforward queries, each querying just one table. Later on you'll be querying more than one table at a time, to piece together the information you need.

- Please type in at least some information matching these queries, for example have a property owner called Carol Farrell.

- "Having no telephone number: the `TelNo` attribute is the empty string. We'd use a `WHERE` clause including the condition `TelNo=''`.

- "Property owner Carol Farrell": first and last names are separate attributes, so you'll have to check that the first name matches `'Carol'` *and* the last name matches `'Farrell'`.

- "Sort the result by city and by area": we can order the output from a query. As an example, say we want to list the first name and telephone number of all renters in apartments, ordered by last name and amount of rent. Notice that we're ordering them on something that isn't being printed out!

  ```
  SELECT FName, TelNo
  FROM RENTER
  WHERE Type='apartment'
  ORDER BY LName, Rent;
  ```

  The effect of this query is to order the renters living in apartments by last name (in alphabetical order), and where more than one have the same last name they will be ordered by the rent they pay (in increasing order), then their first names and telephone numbers will be printed out.

- "Phone number starting 021": this is an example of pattern matching. As an example, say we want to find the first and last names of all staff whose last name ends in 's':

  ```
  SELECT FName, LName
  FROM STAFF
  WHERE LName LIKE '%s';
  ```

  Recall that `%` stands for any string (including the empty string), and _ stands for any single character.

- Please type in several rows to each table. But notice that some tables share attributes, for example `BranchNo` appears in `BRANCH`, `STAFF` and `PROPERTY`: a member of staff at the auctioneer works at a given branch, has a certain salary etc, and deals with certain properties. So please use a small number of branch numbers and use them in each table.

  Another example: `StaffNo` appears in `STAFF` and `PROPERTY`, so again use a few staff numbers and use them in both tables. Also `RenterNo` is in `PROPERTY` and `VIEWING`, and `PropertyNo` is in `PROPERTY` and `VIEWING`.

  But some attributes of the same name have nothing to do with each other, for example `TelNo` in `BRANCH` means the telephone number of the auctioneer branch, `TelNo` in `OWNER` means the telephone number of a property owner, `TelNo` in `STAFF` means the home telephone number of a member of staff, `TelNo` in `RENTER` means the current telephone number of a property renter. Don't get these confused.

You'll have all the SQL features that you need for this assignment from the lecture notes.

SQLite and labs

In the labs we'll use a version of SQL called SQLite: a simple, free library written in C (but you don't need to know C to use it) whose code is public domain. There's good documentation here:

```
https://www.sqlite.org/index.html
```

including instructions for setting it up.

SQLite is actually the most-used database engine in the world. It's built into all mobile phones and most computers, and is used in many popular applications. There are over a trillion SQLite databases in active use.

You can also install SQLite on your computer. As you probably have a variety of computers running Windows, Mac or perhaps Linux, I won't give details on how to install it. Please follow instructions at:

```
https://www.sqlite.org/quickstart.html
```

This gives a link to the code which you can download. It provides a Windows interface, also for Mac and Linux, and you can also access SQLite from VSCode (an integrated development environment that you might have already used in UCC) which gives you a nice interface. You're free to use whichever method works for you, but I'll be using commands that are entered into a command window. If you're not used to setting up software then I recommend this as the simplest approach. Please let me know if you have problems.

Once SQLite is installed you can use it. Assuming you're using a command window, type

```
sqlite3 test.db
```

where `test.db` is the name of the database you want to create (you can use other names). `sqlite3` (or `sqlite3.exe` on Windows) is a simple command-line program that allows you to manually enter and execute SQL statements, prompting you with `sqlite>`. For example the dialogue might look like this in Windows:

```
$ sqlite3 ex1
SQLite version 3.28.0 2019-03-02 15:25:24
Enter ".help" for usage hints.
sqlite> create table tab1(one varchar(10), two smallint);
sqlite> insert into tab1 values('hello!',10);
sqlite> insert into tab1 values('goodbye',20);
sqlite> select * from tab1;
hello!|10
goodbye|20
sqlite>
```

("`|`" separates the columns).

Notice that you must end a command with a semicolon `;`. If you don't, it will assume that what you're typing is still part of the same command, and keep prompting you. Nothing actually happens until you enter `;` then RETURN. You quit SQLite by entering an end-of-file character, typically control-d.