Creating and Modifying Data

Objectives

- Write queries that creates tables.
- Write queries to insert, modify, and delete records.

So far we have only looked at how to get information out of a database, both because that is more frequent than adding information, and because most other operations only make sense once queries are understood. If we want to create and modify data, we need to know two other pairs of commands.

The first pair are `create table` and `drop table`. While they are written as two words, they are actually single commands. The first one creates a new table; its arguments are the names and types of the table's columns. For example, the following statements create the four tables in our survey database:

```
create table Person(ident text, personal text, family text);
create table Site(name text, lat real, long real);
create table Visited(ident integer, site text, dated text);
create table Survey(taken integer, person text, quant real, reading real);
```

We can get rid of one of our tables using:

```
drop table Survey;
```

Be very careful when doing this: most databases have some support for undoing changes, but it's better not to have to rely on it.

Different database systems support different data types for table columns, but most provide the following:

integer	a signed integer
real	a floating point number
text	a character string
blob	a "binary large object", such as an image

Most databases also support Booleans and date/time values; SQLite uses the integers 0 and 1 for the former, and represents the latter as discussed earlier. An increasing number of databases also support geographic data types, such as latitude and longitude. Keeping track of what particular systems do or do not offer, and what names they give different data types, is an unending portability headache.

When we create a table, we can specify several kinds of constraints on its columns. For example, a better definition for the `Survey` table would be:

```
create table Survey(
    taken   integer not null, -- where reading taken
    person  text,             -- may not know who took it
    quant   real not null,    -- the quantity measured
    reading real not null,    -- the actual reading
    primary key(taken, quant),
    foreign key(taken) references Visited(ident),
    foreign key(person) references Person(ident)
);
```

Once again, exactly what constraints are avialable and what they're called depends on which database manager we are using.

Once tables have been created, we can add and remove records using our other pair of commands, `insert` and `delete`. The simplest form of `insert` statement lists values in order:

```
insert into Site values('DR-1', -49.85, -128.57);
insert into Site values('DR-3', -47.15, -126.72);
insert into Site values('MSK-4', -48.87, -123.40);
```

We can also insert values into one table directly from another:

```
create table JustLatLong(lat text, long text);
insert into JustLatLong select lat, long from site;
```

Deleting records can be a bit trickier, because we have to ensure that the database remains internally consistent. If all we care about is a single table, we can use the `delete` command with a `where` clause that matches the records we want to discard. For example, once we realize that Frank Danforth didn't take any measurements, we can remove him from the `Person` table like this:

```
delete from Person where ident = "danforth";
```

But what if we removed Anderson Lake instead? Our `Survey` table would still contain seven records of measurements he'd taken, but that's never supposed to happen: `Survey.person` is a foreign key into the `Person` table, and all our queries assume there will be a row in the latter matching every value in the former.

This problem is called referential integrity (../../gloss.html#referential-integrity): we need to ensure that all references between tables can always be resolved correctly. One way to do this is to delete all the records that use `'lake'` as a foreign key before deleting the record that uses it as a primary key. If our database manager supports it, we can automate this using cascading delete (../../gloss.html#cascading-delete). However, this technique is outside the scope of this chapter.

> Many applications use a hybrid storage model instead of putting everything into a database: the actual data (such as astronomical images) is stored in files, while the database stores the files' names, their modification dates, the region of the sky they cover, their spectral characteristics, and so on. This is also how most music player software is built: the database inside the application keeps track of the MP3 files, but the files themselves live on disk.

Challenges

1. Write an SQL statement to replace all uses of `null` in `Survey.person` with the string `'unknown'`.

2. One of our colleagues has sent us a CSV (../../gloss.html#csv) file containing temperature readings by Robert Olmstead, which is formatted like this:

   ```
   Taken,Temp
   619,-21.5
   622,-15.5
   ```

   Write a small Python program that reads this file in and prints out the SQL `insert` statements needed to add these records to the survey database. Note: you will need to add an entry for Olmstead to the `Person` table. If you are testing your program repeatedly, you may want to investigate SQL's `insert or replace` command.

3. SQLite has several administrative commands that aren't part of the SQL standard. One of them is `.dump`, which prints the SQL commands needed to re-create the database. Another is `.load`, which reads a file created by `.dump` and restores the database. A colleague of yours thinks that storing dump files (which are text) in version control is a good way to track and manage changes to the database. What are the pros and cons of this approach? (Hint: records aren't stored in any particular order.)

Key Points

- Database tables are created using queries that specify their names and the names and properties of their fields.
- Records can be inserted, updated, or deleted using queries.
- It is simpler and safer to modify data when every record has a unique primary key.