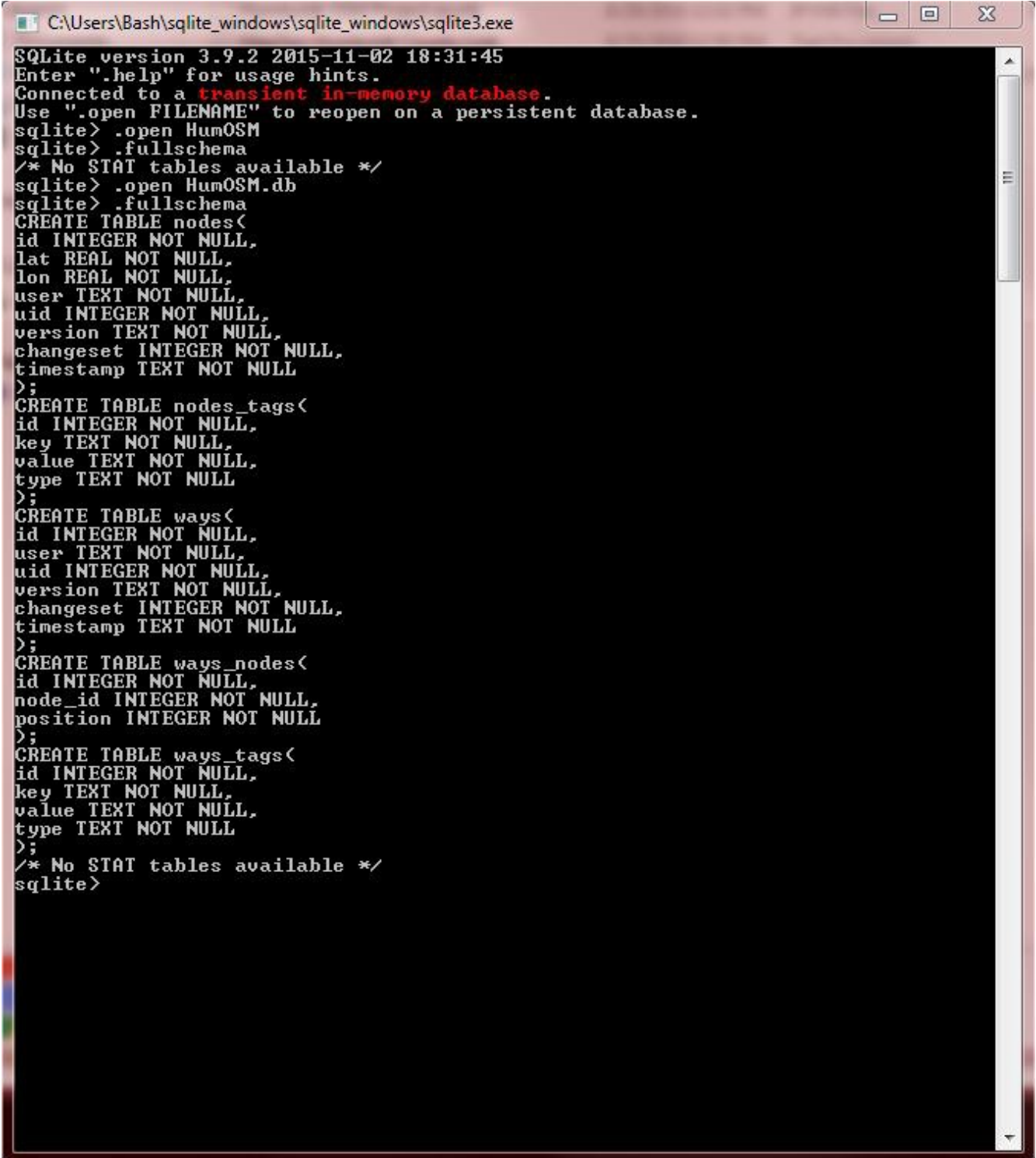


Constructing the HumOSM SQL Database

Our CSVs complete and in order, the next task is to pivot to SQLite3 to construct our database. The first step in creating most SQL databases is to write the schemata:



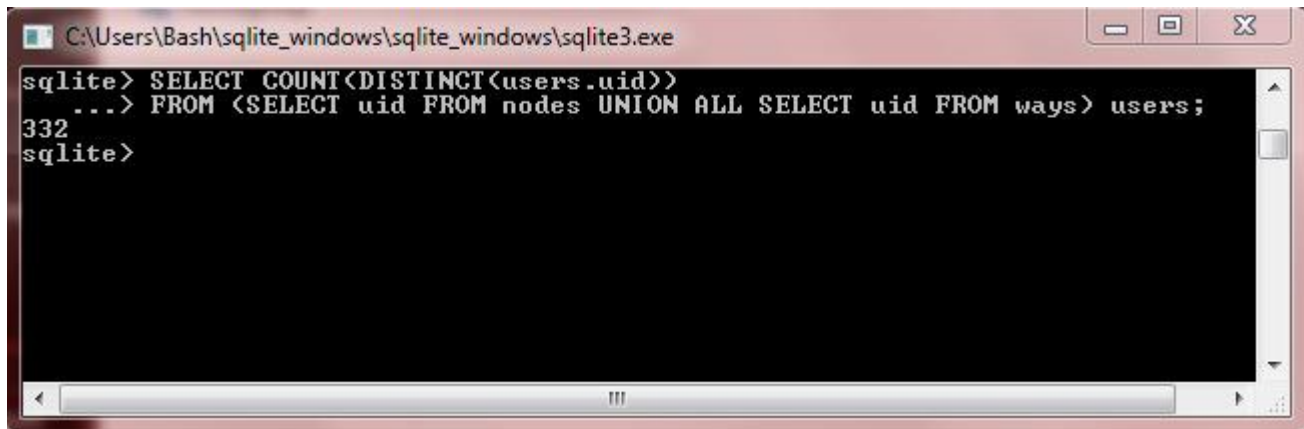
```
C:\Users\Bash\sqlite_windows\sqlite_windows\sqlite3.exe
SQLite version 3.9.2 2015-11-02 18:31:45
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open HumOSM
sqlite> .fullschema
/* No STAT tables available */
sqlite> .open HumOSM.db
sqlite> .fullschema
CREATE TABLE nodes(
id INTEGER NOT NULL,
lat REAL NOT NULL,
lon REAL NOT NULL,
user TEXT NOT NULL,
uid INTEGER NOT NULL,
version TEXT NOT NULL,
changeset INTEGER NOT NULL,
timestamp TEXT NOT NULL
);
CREATE TABLE nodes_tags(
id INTEGER NOT NULL,
key TEXT NOT NULL,
value TEXT NOT NULL,
type TEXT NOT NULL
);
CREATE TABLE ways(
id INTEGER NOT NULL,
user TEXT NOT NULL,
uid INTEGER NOT NULL,
version TEXT NOT NULL,
changeset INTEGER NOT NULL,
timestamp TEXT NOT NULL
);
CREATE TABLE ways_nodes(
id INTEGER NOT NULL,
node_id INTEGER NOT NULL,
position INTEGER NOT NULL
);
CREATE TABLE ways_tags(
id INTEGER NOT NULL,
key TEXT NOT NULL,
value TEXT NOT NULL,
type TEXT NOT NULL
);
/* No STAT tables available */
sqlite>
```

Once the data are imported into their respective tables, we may begin to call queries on our database!
How many rows of information do we have in each of our tables?

A screenshot of a SQLite3 command prompt window. The title bar shows the path 'C:\Users\Bash\sqlite_windows\sqlite_windows\sqlite3.exe'. The window contains the following text:

```
SQLite version 3.9.2 2015-11-02 18:31:45
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open HumOSM.db
sqlite> SELECT Count(*) as row_num FROM nodes
...> ;
2809878
sqlite> SELECT Count(*) as row_num FROM ways
...> ;
133188
sqlite> SELECT Count(*) as row_num FROM nodes_tags;
15598
sqlite> SELECT Count(*) as row_num FROM ways_tags;
1059831
sqlite> SELECT Count(*) as row_num FROM ways_nodes;
2965493
sqlite>
```

How many unique users contributed to the dataset? (Special thanks to the SQL Sample Project for help with syntax)

A screenshot of a SQLite3 command prompt window. The title bar shows the path 'C:\Users\Bash\sqlite_windows\sqlite_windows\sqlite3.exe'. The window contains the following text:

```
sqlite> SELECT COUNT(DISTINCT(users.uid))
...> FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) users;
332
sqlite>
```

Given the special attention paid to dealing with missing values in the dataset, I wanted to know just how many rows from our 'nodes' and 'ways' tables were filled with the default values.

```
C:\Users\Bash\sqlite_windows\sqlite_windows\sqlite3.exe
sqlite> SELECT COUNT(*) FROM nodes
...> WHERE id = 0000
...> OR lat = 0.00
...> OR lon = 0.00
...> OR user = "Missing Value"
...> OR uid = 0000
...> OR version = "Missing Value"
...> OR changeset = 0000
...> OR timestamp = "Missing Value";
1
sqlite> SELECT COUNT(*) FROM ways
...> WHERE id = 0000
...> OR user = "Missing Value"
...> OR uid = 0000
...> OR version = "Missing Value"
...> OR changeset = 0000
...> OR timestamp = "Missing Value";
0
sqlite> SELECT * FROM nodes
...> WHERE id = 0000
...> OR lat = 0.00
...> OR lon = 0.00
...> OR user = "Missing Value"
...> OR uid = 0000
...> OR version = "Missing Value"
...> OR changeset = 0000
...> OR timestamp = "Missing Value";
32820705,40.6303271,-123.4687854,"Missing Value",0,1,167590,"2007-07-25T09:56:21"
sqlite>
```

Naturally, all of the extra trouble I went through to handle missing values was apparently caused by a single 'node' element that was missing its 'user' value. C'est la vie!

Finally, I noticed that the 'node_id' field in ways_nodes references the 'id' field in nodes. I'm curious which of the 'id' numbers are referenced the most.

```
C:\Users\Bash\sqlite_windows\sqlite_windows\sqlite3.exe
sqlite> SELECT nodes.id as Node, COUNT(ways_nodes.node_id) as NodeUse
...> FROM nodes JOIN ways_nodes
...> ON nodes.id = ways_nodes.node_id
...> GROUP BY Node
...> ORDER BY NodeUse Desc
...> Limit 10;
86122425!7
1025120607!7
1056491759!7
1064113270!7
85810999!6
86122433!6
565302686!6
565302689!6
1024647552!6
1053795951!6
sqlite>
```