Introduction to Databases

Yang Liu and Keith Jackson

April 26, 2017

Why Database

Better management of data

- Performance
- Scalability
- Data integrity
- Concurrence
- Fault-tolerant
- And more ...

Databases: SQL vs NoSQL

- SQL (Relational) databae: MySQL, Oracle, SQLite, etc.
  - Data stored in one or more tables (relations)
  - Each table has rows (records) and columns (attributes)

Example table: user_profile

row (record)

id	name	department_id	creation_time
1	James Smith	1	2017-04-26 15:00:00
4	Robert Johnson	1	2017-01-01

Column (attribute)

- NoSQL (Not Only SQL): Hadoop, Cassandra, MonetDB, etc. (http://nosql-database.org/)
  - Non-relational, distributed, and horizontally scalable (using multiple servers instead of one).

SQLite

Zero-configuration, transactional database (https://www.sqlite.org/about.html)

- Stored in a single file
- No need to configure
- Transactional: atomic, consistent, isolated, and durable (ACID)
- Most widely used database: https://www.sqlite.org/mostdeployed.html

Appropriate Uses For SQLite

Not good for

- Concurrent writes
- Big data
- Database file is remote (over network), not local.

More details: https://sqlite.org/whentouse.html

Access SQLite on Ada

Recall: **ssh your_net_id@ada.tamu.edu** to login to Ada cluster

$module spider SQLite # show modules related to SQLite

Versions:

....

SQLite/3.15.2-GCCcore-6.3.0

$module load SQLite/3.15.2-GCCcore-6.3.0 # load a SQLite module

$which sqlite3 # show which sqlite3 to be used

/software/easybuild/software/SQLite/3.15.2-GCCcore-6.3.0/bin/sqlite3

Create a Database

```
$ sqlite3 test.db     # (1) open database if it exists, otherwise create a new database
                      #     and open it
                      # (2) enter sqlite interactive environment
SQLite version 3.15.2 2016-11-28 19:13:37
Enter ".help" for usage hints.
sqlite> .database     # show opened database
seq  name             file
---  ---------------  ----------------------------------------------------------
0    main             /scratch/training/Databases/2017-Spring/test.db
sqlite> .quit         # exit interactive sqlite environment
$
```

Relational Database: Tables

A relational database logically consists of tables

- Table column (atttribute): all values in one column have the same data type (e.g., integer)
- Table row (record): a data item of several columns
- Table primary key: a subset of columns such that no two records have identical values for those columns. Null value is not allowed

Example table: user_profile

id	name	department_id	creation_time
1	James Smith	1	2017-04-26 15:00:00
4	Robert Johnson	1	2017-01-01

row (record)

primary key

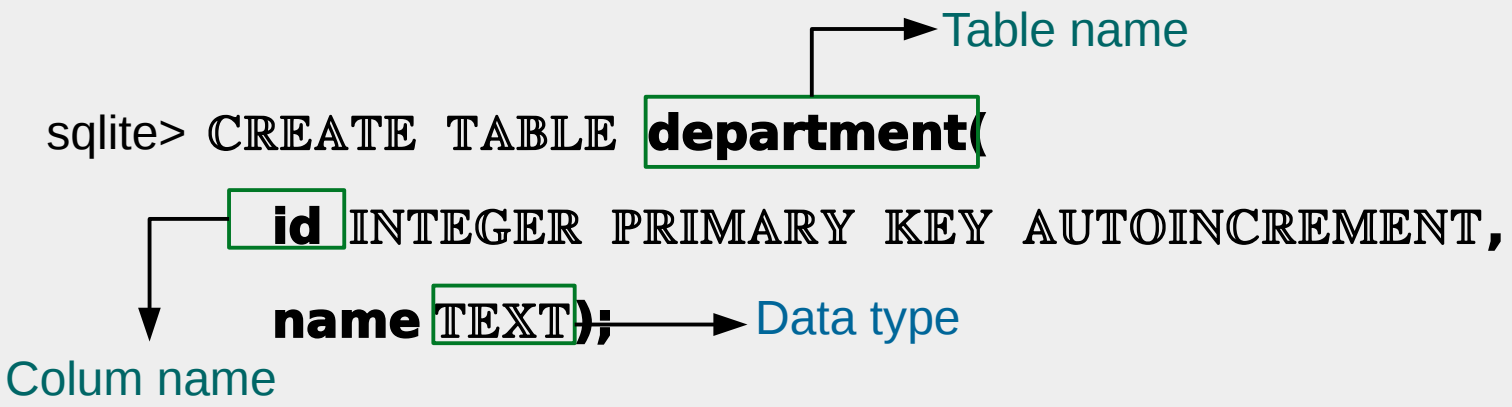
column

Data Types

Specify how data is stored in sqlite database. Sqlite uses dynamic typing: a data describe its own type, while a column does not specify the type of data.

- **Null**
  - A null value
- **Integer**
  - A signed integer (e.g. -5), stored in 1-8 bytes, e.g. smallint
- **Real**
  - A floating point value (e.g. 9.371), stored as an 8-byte number
- **Text**
  - A string of characters (e.g. “hello”), encoded as UTF-8, UTF-16BE, or UTF-16LE.
- **Blob**
  - A binary large object (e.g., an image), stored as it was input.

Create a Table: department

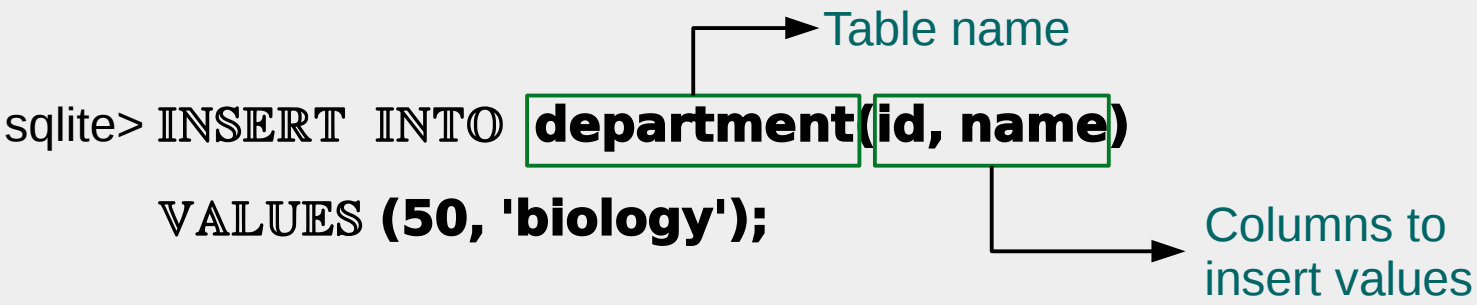


AUTOINCREMENT—value increased by 1 when no value is provided for a new data (record) to be inserted (further discussions: http://www.sqlitetutorial.net/sqlite-autoincrement/)

Empty table created: department

id	name

Insert Data (Record): department




```
sqlite> INSERT INTO department(name)
        VALUES ('mathematics'), ('physics');
```

id (AUTOINCREMENT) can be omitted

Table department after Insertions

id	name
50	biology
51	mathematics
52	physics

Create a Table: user_profile

```
sqlite> CREATE TABLE user_profile(
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        department_id INT NOT NULL,
        creation_time DATETIME NOT NULL DEFAULT
                current_timestamp);
```

DEFAULT—value for a column if no value is given for that column when a record is inserted.

Empty table created: user_profile

id	name	department_id	creation_time

Insert Data (Record): user_profile

```
sqlite> INSERT INTO user_profile(name, department_id)
        VALUES ('James Smith', 51), ('Michael Smith', 52), ('Maria  Rodriguez', 53);

sqlite> INSERT INTO user_profile(name, department_id, creation_time)
        VALUES('Robert Johnson', 1, '2017-01-01'),
              ('Maria Garcia', 2, '2017-01-02'),
              ('Williams Hernandez', 3, '2017-01-02');
```

creation_time (DEFAULT current_timestamp) can be ommitted.

Table user_profile (partila data) after Insertions

id	name	department_id	creation_time
1	James Smith	51	2017-04-26 15:00:00
4	Robert Johnson	51	2017-01-01

One More Table: job_usage

```
sqlite> CREATE TABLE job_usage(
            id INT PRIMARY KEY NOT NULL,
            submit_time DATETIME NOT NULL,
            used_SUs REAL,
            user_profile_id INT NOT NULL);
```

Table job_usage (partial data) after Insertions

id	submit_time	used_SUs	user_profile_id
100	2017-01-01 19:30:05	20	1
200	2017-01-01 21:05:42	10	2

```
sqlite> INSERT INTO job_usage(id, submit_time, used_SUs, user_profile_id)
VALUES (100, '2017-01-01 19:30:05', 20, 1), (200, '2017-01-01 21:05:42', 10, 2),
       (300, '2017-01-05 03:52:38', 30, 3), (120, '2017-01-21 19:30:05', 200, 1),
       (260, '2017-01-21 21:05:42', 100, 2), (310, '2017-01-25 03:52:38', 300, 3);
```

Show Tables and Schemas

sqlite> **.tables** # list all tables

Department job_usage user_profile

sqlite> **.schema** # show complete database schema including all tables and indices

CREATE TABLE department(id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT);

…

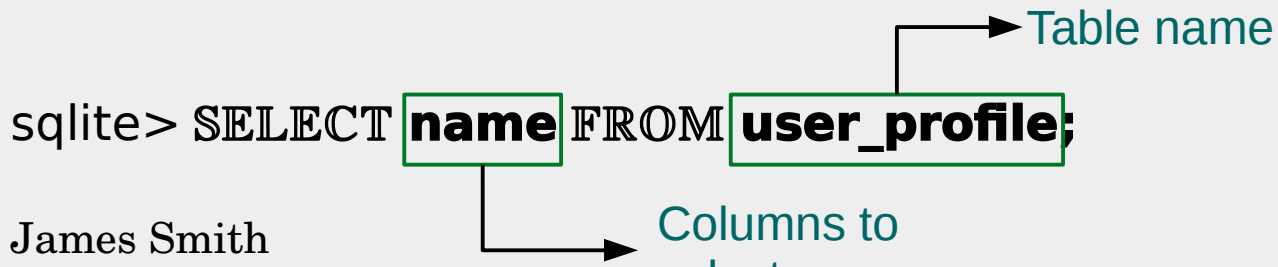
CREATE TABLE job_usage(id INT PRIMARY KEY NOT NULL, submit_time

DATETIME NOT NULL, used_SUs REAL, user_profile_id INT NOT NULL);

sqlite> **.schema job_usage** # show schema for table job_usage

CREATE TABLE job_usage(id INT PRIMARY KEY NOT NULL, submit_time DATETIME NOT NULL, used_SUs REAL, user_profile_id INT NOT NULL);

Select Data (Records): Select



James Smith

Micahel Smith

...

Maria Garcia

Williams Hernandez

```
sqlite> SELECT * FROM user_profile;     # * refers to all columns.
```

1|James Smith|51|2017-04-05 19:30:04

...

6|Williams Hernandez|53|2017-01-02

Select Data with Conditions : Where

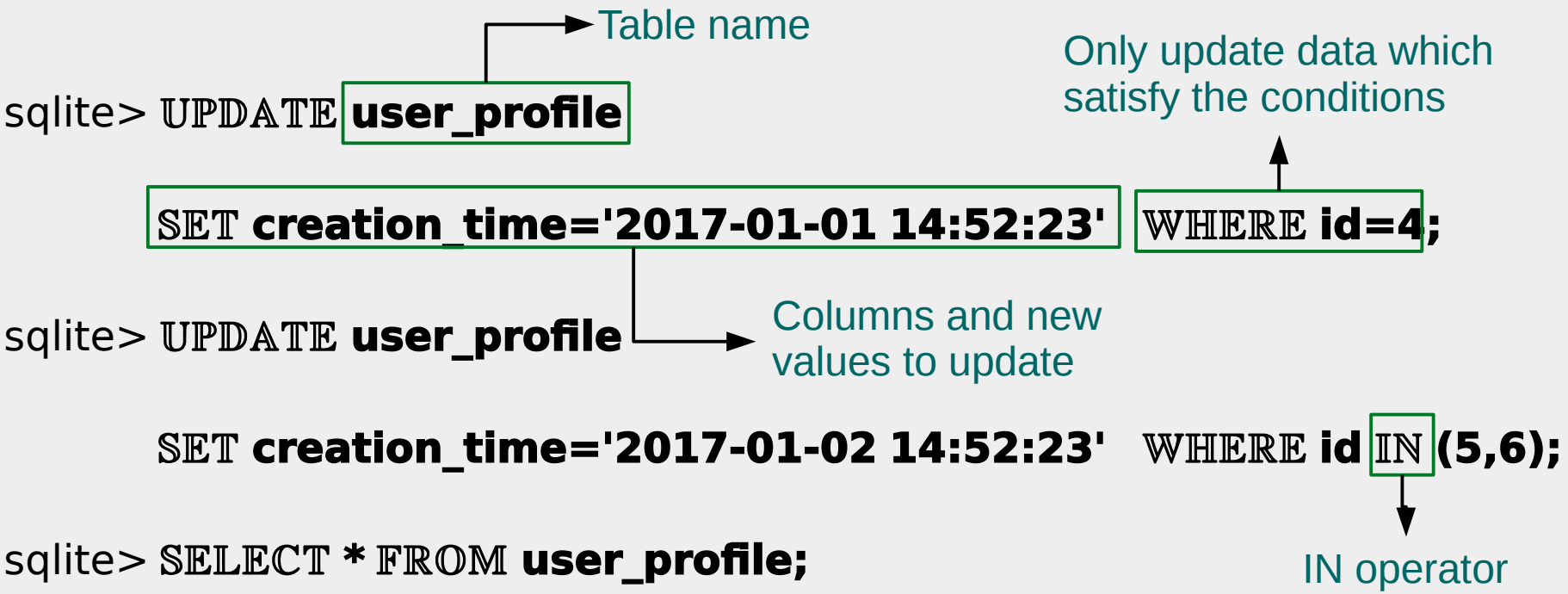
```
sqlite> SELECT *
        FROM user_profile
        WHERE department_id = 51;
1|James Smith|51|2017-04-05 19:30:04
4|Robert Johnson|51|2017-01-01 14:52:23
```

Only select data which satisfy the 'WHERE' conditions

```
sqlite> SELECT *
        FROM user_profile
        WHERE department_id = 1 AND creation_time > '2017-03-01';
1|James Smith|51|2017-04-05 19:30:04
```

Logical operations can be used in WHERE condiiitions

Update Data (Records)



Aggregate Functions

```
sqlite> SELECT min(used_SUs) mininum_usage,
        max(used_SUs) maximum_usage,
        avg(used_SUs) avarage_usage,
        count(*) number_of_jobs
        FROM job_usage;
10.0|300.0|110.0|6
```

Average of all used_SUs

Number of rows selected

Table job_usage

Id	submit_time	used_SUs	user_profile_id
100	2017-01-01 19:30:05	20.0	1
200	2017-01-01 21:05:42	10.0	2
300	2017-01-05 03:52:38	30.0	3
120	2017-01-21 19:30:05	200.0	1
260	2017-01-21 21:05:42	100.0	2
310	2017-01-25 03:52:38	300.0	3

Aggregate Functions with Group : Group by

```
sqlite> SELECT user_profile_id, sum(used_SUs)
        FROM job_usage
        GROUP BY user_profile_id;
1|220.0
2|110.0
3|330.0
```

Group data by colums listed here

Table job_usage (with group by user_profile_id)

Id	submit_time	used_SUs	user_profile_id
100	2017-01-01 19:30:05	20.0	1
120	2017-01-21 19:30:05	200.0	1
200	2017-01-01 21:05:42	10.0	2
260	2017-01-21 21:05:42	100.0	2
300	2017-01-05 03:52:38	30.0	3
310	2017-01-25 03:52:38	300.0	3

Group 1 (user_profile_id=1)

Group 2 (user_profile_id=2)

Group 3 (user_profile_id=3)

Order Selected Data : Order by

```
sqlite> SELECT user_profile_id, sum(used_SUs) total_used_SUs
        FROM job_usage
        GROUP BY user_profile_id
        ORDER BY total_used_SUs;
```

Order **selected data** by the columns listed here

```
2|110.0
1|220.0
3|330.0
```

Non decreasing order by default

```
sqlite> SELECT user_profile_id, sum(used_SUs) total_used_SUs
        FROM job_usage
        GROUP BY user_profile_id
        ORDER BY total_used_SUs DESC;
```

Change the order to be non-incresaing

```
3|330.0
1|220.0
2|110.0
```

Non increasing order

Select Data from Multiple Tables : Join

```
sqlite> SELECT u.name, sum(j.used_SUs) total_used_SUs
        FROM job_usage j   JOIN user_profile u   ON j.user_profile_id= u.id
        GROUP BY user_profile_id   ORDER BY total_used_SUs DESC ;
```

id	name	department_id	creation_time
3	Maria Rodriguez	53	2017-04-05 19:30:04
4	Robert Johnson	51	2017-01-01

user_profile — join — job_usage

id	submit_time	used_SUs	user_profile_id
300	2017-01-05 03:52:38	30	3
310	2017-01-25 03:52:38	300	3

New/Temporary table after join

on (user_profile.id) — on (job_usage.user_profile_id)

user_profile.id	user_profile.name	user_profile.department_id	user_profile.creation_time	job_usage.id	job_usage.submit_time	job_usage.used_SUs	job_usage.user_profile_id
3	Maria Rodriguez	53	2017-04-05 19:30:04	300	2017-01-05 03:52:38	30	3
3	Maria Rodriguez	53	2017-04-05 19:30:04	310	2017-01-25 03:52:38	300	3

Select Data from Multiple Tables : Join (Continue)

```
sqlite> SELECT u.name, sum(j.used_SUs) total_used_SUs
        FROM job_usage j
        JOIN user_profile u  ON j.user_profile_id= u.id
        GROUP BY user_profile_id
        ORDER BY total_used_SUs DESC ;
```

Maria Rodriguez | 330.0
James Smith | 220.0
Michael Smith | 110.0

New/Temporary table after join

user_profile.id	user_profile.name	user_profile.department_id	user_profile.creation_time	job_usage.id	job_usage.submit_time	job_usage.used_SUs	job_usage.user_profile_id
3	Maria Rodriguez	53	2017-04-05 19:30:04	300	2017-01-05 03:52:38	30	3
3	Maria Rodriguez	53	2017-04-05 19:30:04	310	2017-01-25 03:52:38	300	3

Delete Data: Delete

sqlite> SELECT * FROM **department**;

51 | biology

52 | mathematics

...

sqlite> DELETE FROM **department** WHERE **id =** 1;

sqlite> SELECT * FROM **department**; # record/row (51, 'biology') no longer in the table

52 | mathematics

..

Structured Query Language (SQL)

SQL Include both DDL and DML. SQL is declarative: define what to do, not how to do.

- DDL (Data Definition Language): create table, etc
- DML (Data Manipulation Language): select, delete, etc.

Execute SQL Commands in a File

$ **sqlite3 test.db < user_job_usage.sql**

Maria Rodriguez|330.0
James Smith|220.0
Michael Smith|110.0

$ **cat user_job_usage.sql**

```
SELECT u.name, sum(j.used_SUs) total_used_SUs
FROM job_usage j
JOIN user_profile u
ON j.user_profile_id= u.id
GROUP BY user_profile_id
ORDER BY total_used_SUs DESC;
```

Store Results into a File

$ **sqlite3 test.db <user_job_usage.sql**

> user_job_usage.txt

$ **cat user_job_usage.txt**

Maria Rodriguez | 330.0
James Smith | 220.0
Micahel Smith | 110.0

Store Results into a CSV File

$ **sqlite3 test.db <user_job_usage_csv.sql**

$ **cat user_job_usage.csv**

name,total_used_SUs
“Maria Rodriguez",330.0
“James Smith",220.0
"Michael Smith",110.0

$ **cat user_job_usage_csv.sql**

.headers on
.mode csv
.output user_job_usage.csv
SELECT u.name, sum(j.used_SUs) total_used_SUs FROM job_usage j JOIN user_profile u ON j.user_profile_id= u.id GROUP BY user_profile_id ORDER BY total_used_SUs DESC ;

Import Data from a CSV File

sqlite> **create table user_job_usage(name** TEXT NOT NULL**,**

total_used_SUs REAL**);** # create an empty table

sqlite> **.separator ,** # specify the separator used in the csv file to import

sqlite> **.import user_job_usage.csv user_job_usage** #import the csv file into the

table. Do NOT have ";" at the end of the ".import" command.

sqlite> SELECT * FROM **user_job_usage**;

"Maria Rodriguez",330.0
"James Smith",220.0
"Michael Smith",110.0

Dump Database

$ **sqlite3 test.db .dump > test.db.backup** # save database (both schema and data)

$ **cat test.db.backup**

PRAGMA foreign_keys=OFF;

BEGIN TRANSACTION;

CREATE TABLE job_usage(id INT PRIMARY KEY NOT NULL, submit_time DATETIME NOT NULL, used_SUs REAL, user_profile_id INT NOT NULL);

INSERT INTO "job_usage" VALUES(100,'2017-01-01 19:30:05',20.0,1);

...

COMMIT;

Restore Databases

$ **rm test.db**

$ **sqlite3 test.db**

sqlite> .schema

sqlite> # empty database

$ **sqlite3 test.db < test.db.backup**

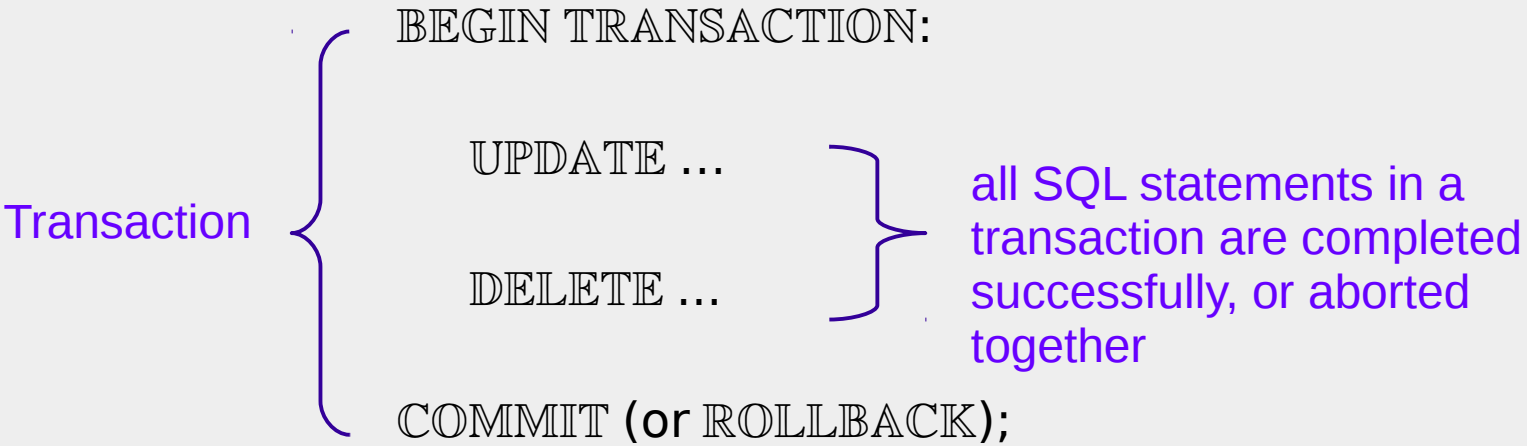
$ **sqlite3 test.db**

sqlite> .schema # show schema for database from backup

CREATE TABLE job_usage(id INT PRIMARY KEY NOT NULL, submit_time DATETIME NOT NULL, used_SUs REAL, user_profile_id INT NOT NULL);

...

Transaction



Trasaction Atomicity: Changes Invisible Before Commit

```
sqlite> .schema user_job_usage
CREATE TABLE user_job_usage(name TEXT NOT NULL, total_used_SUs REAL);
sqlite> SELECT * FROM user_job_usage;
name|total_used_SUs
Maria Rodriguez|330.0
James Smith|220.0
Michael Smith|110.0
```

```
sqlite> BEGIN TRANSACTION;
sqlite> INSERT INTO user_job_usage(name,
total_used_SUs) VALUES("Yang Liu", 100);
sqlite> UPDATE user_job_usage SET
total_used_SUs=50 WHERE name ="Maria
Rodriguez";
```

Transaction T1 not committed yet

```
sqlite> select * from user_job_usage;
name|total_used_SUs
Maria Rodriguez|330.0
James Smith|220.0
Michael Smith|110.0
```

Another user does not see any changes from T1

Trasaction Atomicity: Changes Visible After Commit

```
sqlite> .schema user_job_usage
CREATE TABLE user_job_usage(name TEXT NOT NULL, total_used_SUs REAL);
sqlite> SELECT * FROM user_job_usage;
name|total_used_SUs
Maria Rodriguez|330.0
James Smith|220.0
Michael Smith|110.0
```

```
sqlite> BEGIN TRANSACTION;
sqlite> INSERT INTO user_job_usage(name,
total_used_SUs) VALUES("Yang Liu", 100);
sqlite> UPDATE user_job_usage SET
total_used_SUs=50 WHERE name ="Maria
Rodriguez";
sqlite> COMMIT;
```

Transaction T1 committed

```
sqlite> select * from user_job_usage;
name|total_used_SUs
Maria Rodriguez|50.0
James Smith|220.0
Michael Smith|110.0
Yang Liu|100
```

Another user sees all changes from T1

Trasaction Atomicity: Partial Changes Visible After Failure

```
sqlite> .schema user_job_usage
CREATE TABLE user_job_usage(name TEXT NOT NULL, total_used_SUs REAL);
sqlite> SELECT * FROM user_job_usage;
name|total_used_SUs
Maria Rodriguez|330.0
James Smith|220.0
Michael Smith|110.0
```

```
sqlite> BEGIN TRANSACTION;
sqlite> INSERT INTO user_job_usage(name,
total_used_SUs) VALUES("Yang Liu", 100);
sqlite> UPDATE user_job_usage SET
non_existing_column=50 WHERE name
="Maria Rodriguez";
sqlite> COMMIT;
```

Transaction T1 committed

```
sqlite> select * from user_job_usage;
name|total_used_SUs
Maria Rodriguez|330.0
James Smith|220.0
Michael Smith|110.0
Yang Liu|100
```

Another user sees partial changes from T1

Trasaction Atomicity: Rollback

```
sqlite> .schema user_job_usage
CREATE TABLE user_job_usage(name TEXT NOT NULL, total_used_SUs REAL);
sqlite> SELECT * FROM user_job_usage;
name|total_used_SUs
Maria Rodriguez|330.0
James Smith|220.0
Michael Smith|110.0
```

```
sqlite> BEGIN TRANSACTION;
sqlite> INSERT INTO user_job_usage(name,
total_used_SUs) VALUES("Yang Liu", 100);
sqlite> UPDATE user_job_usage SET
non_existing_column=50 WHERE name
="Maria Rodriguez";
sqlite> ROLLBACK;
```

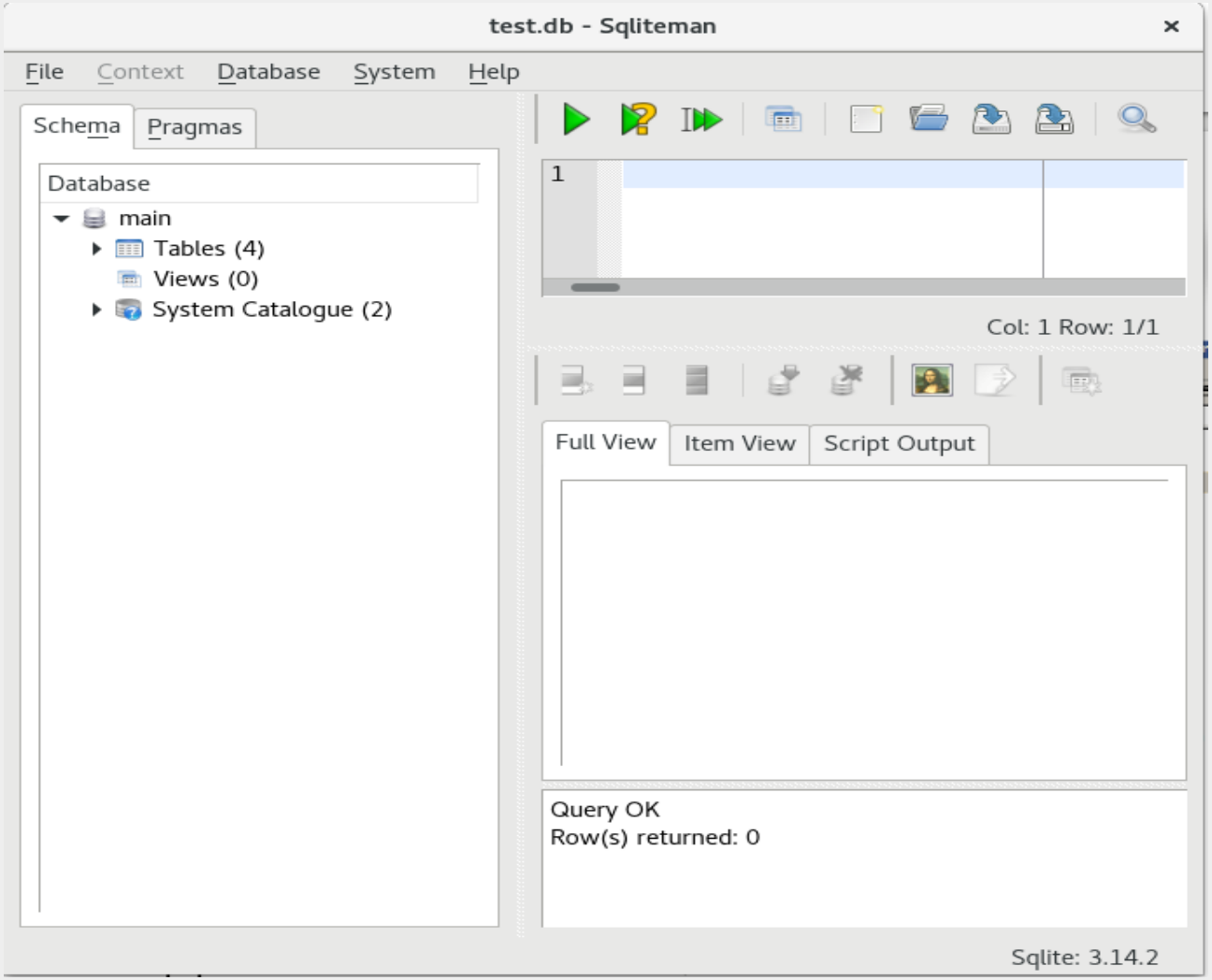
Transaction T1 rolled back

```
sqlite> select * from user_job_usage;
name|total_used_SUs
Maria Rodriguez|330.0
James Smith|220.0
Michael Smith|110.0
```

Another user sees none of those changes from T1

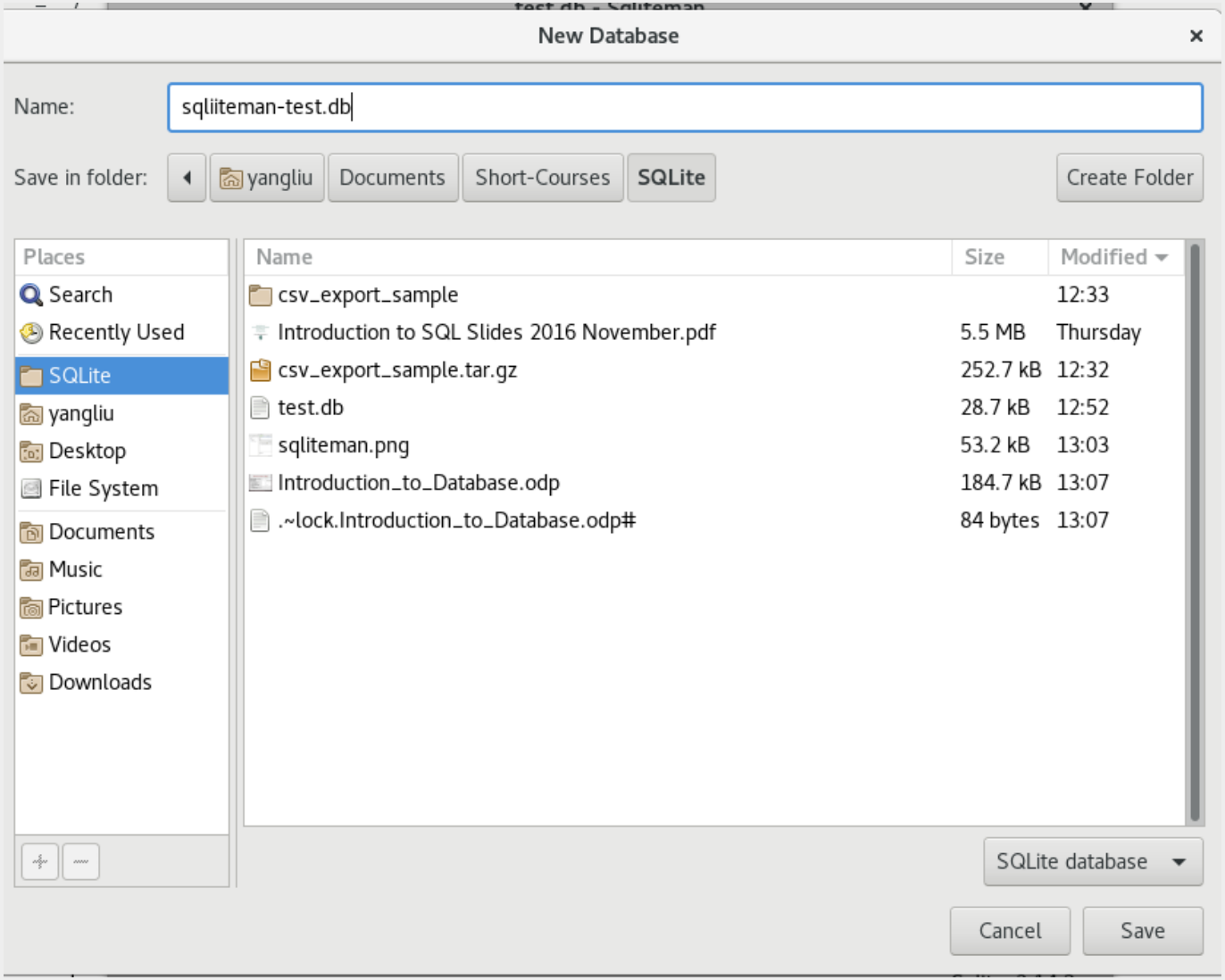
Sqliteman: Managment Tool for Sqlite

- Rpm available on my computer (Fedora Linux)
- Just run ‘sqliteman’ and a graphic interface will be dsiplayed:
- More management tools: https://www.sqlite.org/cvstrac/wiki?p=ManagementTools



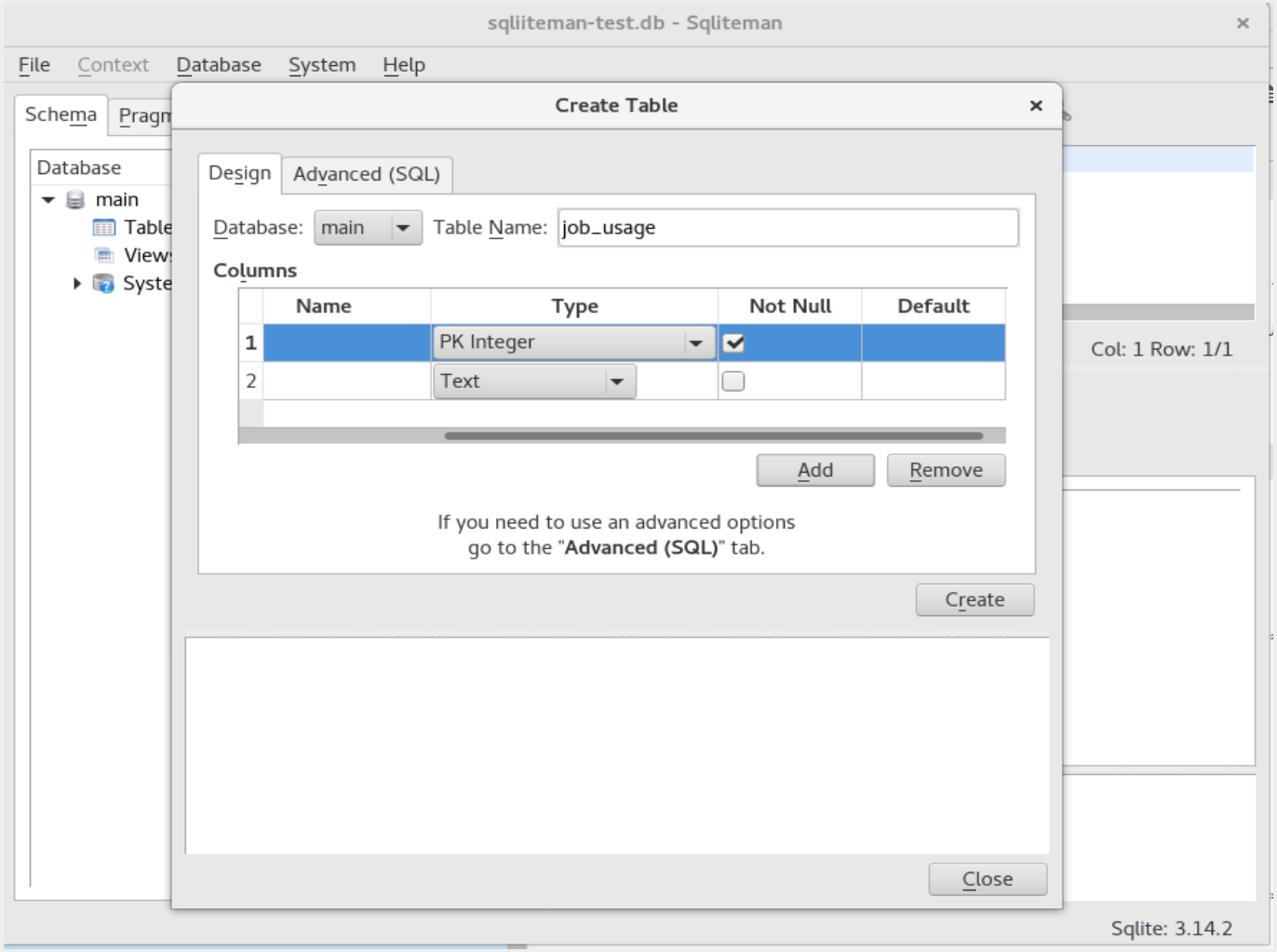
Sqliteman: Creat a Database

- Click ‘File’
- Click ‘New’
- Enter database name ‘sqliteman-test.db’
- Click ‘Save’



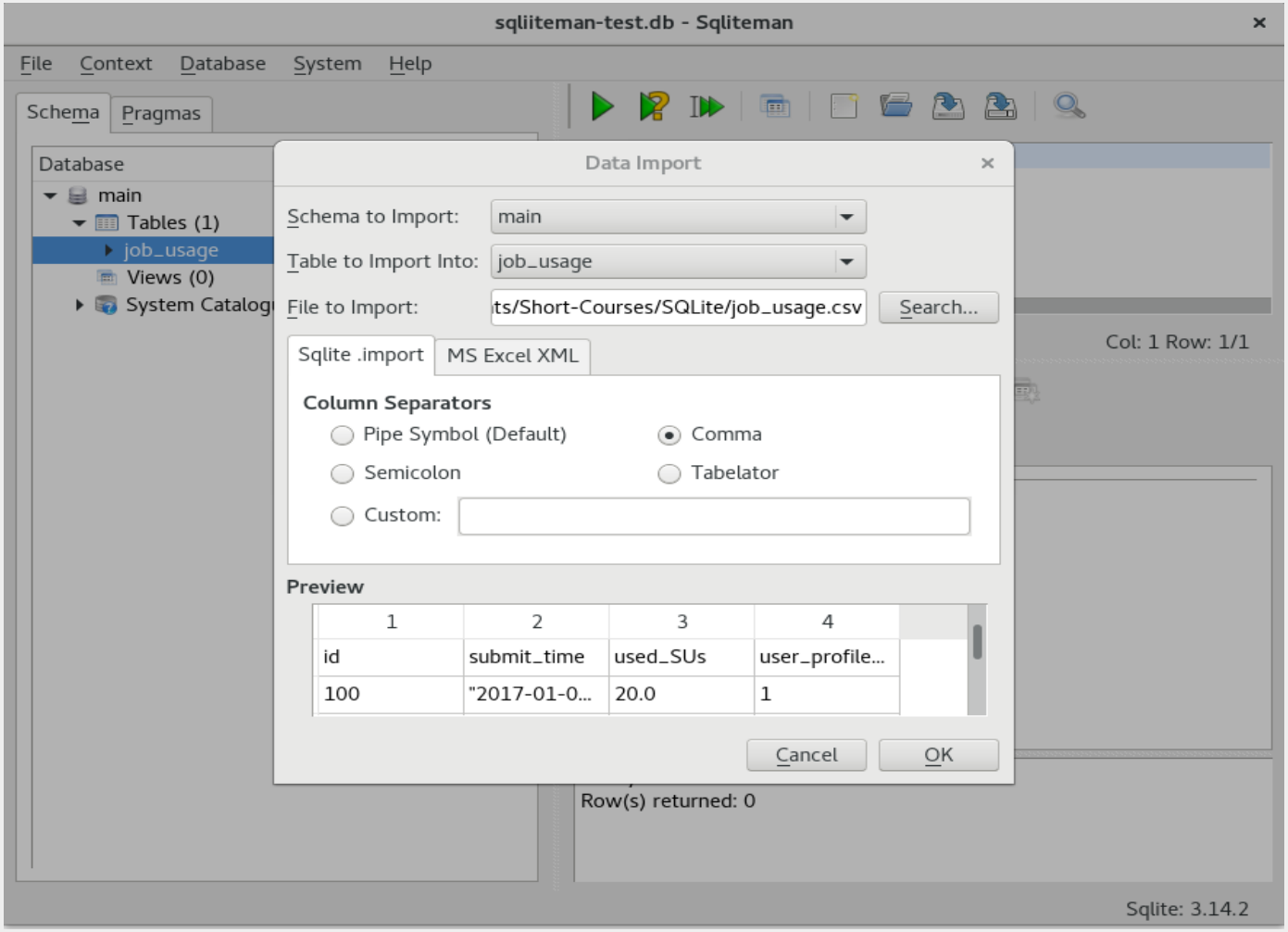
Sqliteman: Create a Table

- Click ‘Database’
- Click ‘Create Table’
- Fill in definitions for first columm
- Click ‘Add’ to add one more column and define it
- …
- Click ‘Create’
- Click ‘Close’



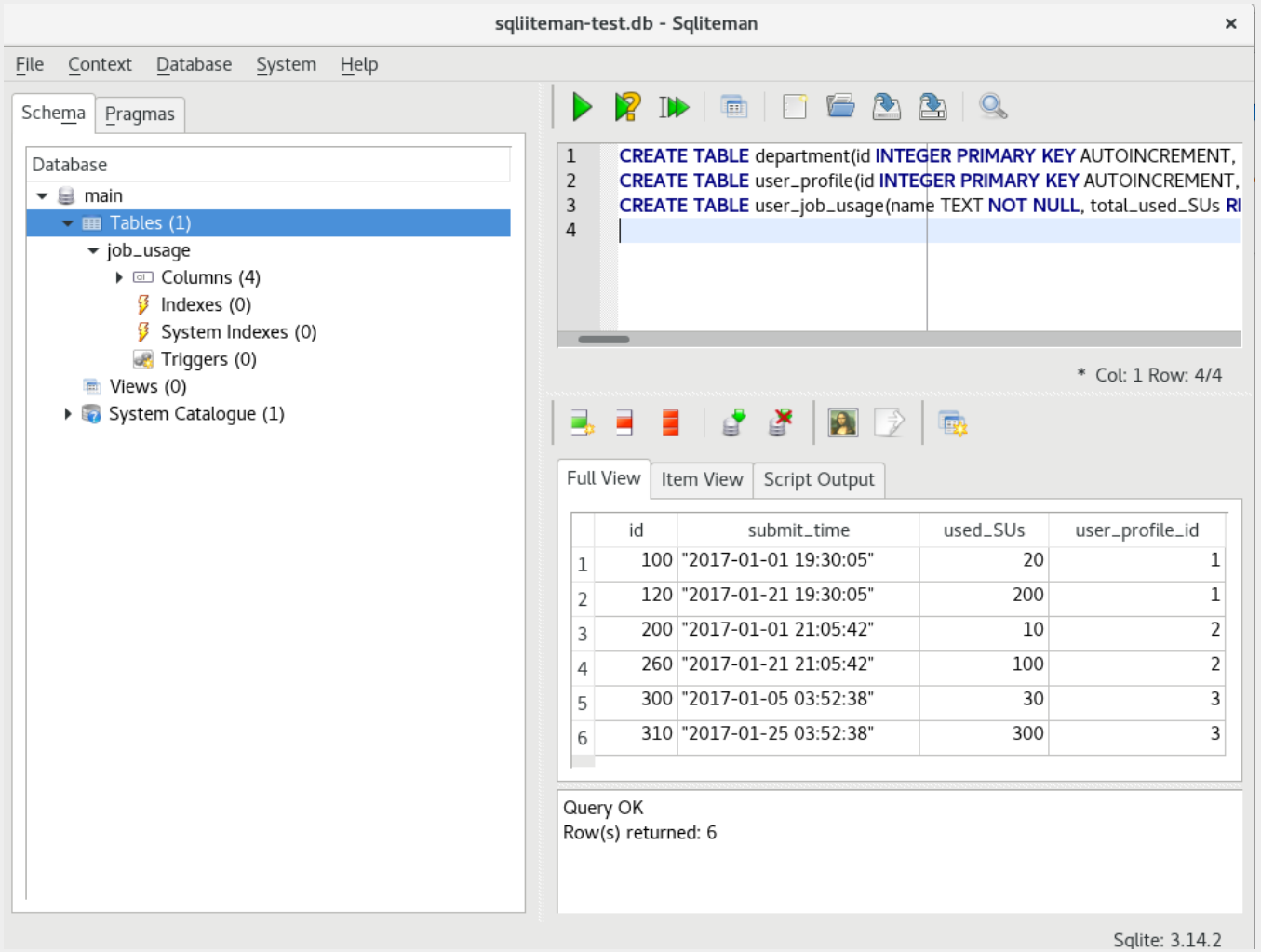
Sqliteman: Import CSV Data Into Table

- Expand the ‘Tables’
- Click ‘job_usage’ (table to expand) with right button of mouse.
- Select the file to import and ‘column ‘separators’ (comma for this example)
- Click ‘Ok’



Sqliteman: Run SQL Statements

- Enter multiple SQL statements in SQL editor
- Click 'Run multiple SQL ...'



Python Program Accessing SQLite

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import sqlite3 as lite
con = lite.connect('test.db')
with con:
    con.row_factory = lite.Row
    cur = con.cursor()
    query = """\
      SELECT u.name, sum(j.used_SUs) total_used_SUs       FROM job_usage j
     JOIN user_profile u      ON j.user_profile_id= u.id
     GROUP BY user_profile_id      ORDER BY total_used_SUs DESC;
"""
    cur.execute(query)
    rows = cur.fetchall()
    for row in rows:
        print "%24s : %5.2f" % (row["name"], row["total_used_SUs"])
```

Perl Program Accesing SQLite

```
#! /usr/bin/perl

use DBI qw(:sql_types);


my $file = "test.db";

my $dsn = "dbi:SQLite:dbname=$file";

my $dbh = DBI->connect($dsn, '', '', { RaiseError => 1,
AutoCommit => 0 }, )

   or die $DBI::errstr;


my $query =  <<"END_QUERY";

  SELECT u.name, sum(j.used_SUs) total_used_SUs

    FROM job_usage j

    JOIN user_profile u

    ON j.user_profile_id= u.id

    GROUP BY user_profile_id

    ORDER BY total_used_SUs DESC;

END_QUERY
```

Example program (part 1)

```
my $sth = $dbh->prepare($query)

   or die $dbh->errstr;

eval { $sth->execute(); };

if ($@) # error

{

   die "SQLite ($query): $DBI::errstr ";

}


while ($href = $sth->fetchrow_hashref())

{

   my $name = $href->{name};

   my $SUs  = $href->{total_used_SUs};

   printf "%24s : %5.2f\n", $name, $SUs;

}
```

Example program (part 2)

Upcoming Seminar

Toughness, Roughness, and Crack Path Engineering for Improved Fracture Resistance

Alan Needleman

University Distinguished Professor

Koldus Building 110

May 1, 2017 : 2:00-03:00pm

More details:

http://hprc.tamu.edu/events/seminars/FY2017/2017-05-01/index.php

HPRC Research Computing Week