Simple Data Storage: SQLite

Dr. Bo (Beth) Sun

How to store the data?
What’s the easiest way?

SQLite

Most popular embedded database in the world

- Well-known users: http://www.sqlite.org/famous.html
  iPhone (iOS), Android, Chrome (browsers), Mac, etc.

Self-contained: one file contains data + schema

Serverless: database right on your computer

Zero-configuration: no need to set up!

http://www.sqlite.org http://www.sqlite.org/different.html

SQL Refresher

SQL Refresher: create table

```
>sqlite3 database.db

sqlite> create table student(id integer, name text);
sqlite> .schema
CREATE TABLE student(id integer, name text);
```

Id	name

SQL Refresher: insert rows

```
insert into student values(111, "Smith");
insert into student values(222, "Johnson");
insert into student values(333, "Lee");
select * from student;
```

id	name
111	Smith
222	Johnson
333	Lee

SQL Refresher: create another table

```
create table takes
(id integer, course_id integer, grade integer);
```

```
sqlite>.schema
CREATE TABLE student(id integer, name text);
CREATE TABLE takes (id integer, course_id integer,
grade integer);
```

id	course_id	grade

SQL Refresher: joining 2 tables

More than one tables - **joins**

E.g., create roster for this course (6242)

id	name
111	Smith
222	Johnson
333	Lee

id	course_id	grade
111	6242	100
222	6242	90
222	4000	80

SQL Refresher: joining 2 tables + filtering

```
select name from student, takes
where
    student.id = takes.id and
    takes.course_id = 6242;
```

id	name
111	**Smith**
222	**Johnson**
333	Lee

id	course_id	grade
111	6242	100
222	6242	90
222	4000	80

Summarizing data:
Find **id** and **GPA** (a summary) for each student

```
select id, avg(grade)
from takes
group by id;
```

Id	course_id	grade
111	6242	100
222	6242	90
222	4000	80

id	avg(grade)
111	100
222	85

Filtering Summarized Results

```
select id, avg(grade)
from takes
group by id
having avg(grade) > 90;
```

id	course_id	grade
111	6242	100
222	6242	90
222	4000	80

id	avg(grade)
111	100
~~222~~	~~85~~

SQL General Form

```
select a1, a2, ... an
from t1, t2, ... tm
where predicate
[order by ....]
[group by ...]
[having ...]
```

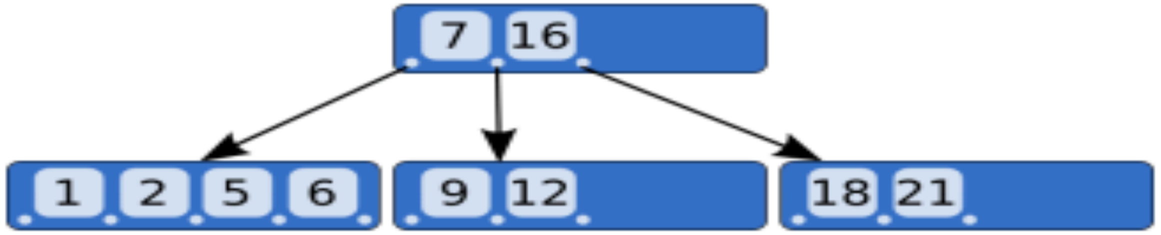
SQLite easily scales to multiple GBs.

What if slow?

Important sanity check:
Have you (or someone) created appropriate **indexes**?

SQLite's indices use **B-tree** data structure.
O(log n) speed for adding/finding/deleting an item

```
create index student_id_index on
student(id);
```



https://en.wikipedia.org/wiki/B-tree

Create Index course_index on takes (course_id)

id	course_id	grade
111	6242	100
222	6242	90
222	4000	80

course_id	course_index
6242	1
6242	1
4000	2

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

```
CREATE VIEW [average_grade]  AS
SELECT id, avg(grade)
FROM takes
Group by id;
```

Why view?

1. views provide an abstraction layer over tables. You can add and remove the columns in the view without touching the schema of the underlying tables.
2. you can use views to complex queries with joins to simplify the data access

SQLite: Full Text Search (FTS) for fast text-based querying

- FTS3 and FTS4 are SQLite virtual table modules that allows users to perform full-text searches on a set of documents.

```
CREATE VIRTUAL TABLE student1 USING fts3(id integer, name text); /*
FTS3 table */
CREATE TABLE student2 (id integer, name text); /* Ordinary table */
```

```
SELECT count(*) FROM student1 WHERE name MATCH ’alek'; /* 0.03 seconds */
SELECT count(*) FROM student2 WHERE name LIKE '%alek%'; /* 22.5 seconds */
```

https://www.sqlite.org/fts3.html

Import data

- Insert into student1(id, name) values (111, “alek”);
- Insert into *FTS_table_name* select * from *ordinary_table*

Simple FTS Queries

```
-- The examples in this block assume the following FTS table:
CREATE VIRTUAL TABLE mail USING fts3(subject, body);
SELECT * FROM mail WHERE rowid = 15; -- Fast. Rowid lookup.
SELECT * FROM mail WHERE body MATCH 'sqlite'; -- Fast. Full-text query.
SELECT * FROM mail WHERE mail MATCH 'search'; -- Fast. Full-text query.
SELECT * FROM mail WHERE rowid BETWEEN 15 AND 20; -- Fast. Rowid lookup.
SELECT * FROM mail WHERE subject = 'database'; -- Slow. Linear scan.
SELECT * FROM mail WHERE subject MATCH 'database'; -- Fast. Full-text query.
```

- *-- Virtual table declaration*

CREATE VIRTUAL TABLE docs USING fts3(title, body);

- *-- Query for all documents containing the term "linux":*

SELECT * FROM docs WHERE docs MATCH 'linux';

- *-- Query for all documents containing a term with the prefix "lin". This will match -- all documents that contain "linux", but also those that contain terms "linear", --"linker", "linguistic" and so on.*

SELECT * FROM docs WHERE docs MATCH 'lin*';

-- All documents for which "linux" is the first token of at least one -- column.

SELECT * FROM docs WHERE docs MATCH '^linux’;

NEAR Queries

- *-- Search for a document that contains the terms "sqlite" and "database" with -- not more than 10 intervening terms.*

SELECT * FROM docs WHERE docs MATCH 'sqlite NEAR database';

- *-- Search for a document that contains the phrase "ACID compliant" and the term -- "database" with not more than 2 terms separating the two.*

SELECT * FROM docs WHERE docs MATCH 'database NEAR/2 "ACID compliant"';