ECE4721 SU 24 Lab3: SQLite

> Contributor: ECE4721J 24SU Teaching Group

- ECE4721 SU 24 Lab3: SQLite
  - Target of the Lab Session
  - Basic knowledge on Sql and sqlite.
    - Intro to sql
      - Relational Database
      - Keys, linking
        - sql in ece472:
    - Loading a table
    - Basic Sql Syntax
      - SELECT ... FROM ...
      - ORDER BY
    - Operators
      - Join
      - Union
      - Intersect
  - Workflow of Query using sql
  - Lab Submissions
  - Reference

Target of the Lab Session

In this lab, we will:

- Get some basic knowledge on Sql and sqlite;
- See how to integrate Sql with other common programming languages such as Java and Python.
- Practice through analyzing a famous dataset.

Basic knowledge on Sql and sqlite.

Intro to sql

From Wikipedia:

SQL is designed for a specific purpose: to query data contained in a **relational database**.

Relational Database

What is a relational database?

what is a relational database?

A relational database is a type of database that organizes data into tables which can be linked—or related—based on data common to each. This method of linking data is the basis of the "relational" in relational database. Each table, which is similar to a spreadsheet, consists of rows and columns. Rows represent individual records, and columns represent attributes or fields of the data.

Takeaway:

relational database organizes data into tables that can be **linked**.

Keys, linking

What does linking mean?

what does linking data mean? can you explain relational a little bit?

In the context of relational databases, "linking data" refers to the process of establishing relationships between separate data entities (often represented as tables) using keys. This concept is foundational to relational database design, allowing for efficient data retrieval and management. Here's a more detailed explanation:

Primary Keys and Foreign Keys

1. **Primary Key**: Each table in a relational database typically has a primary key. This is a column (or set of columns) that uniquely identifies each row within the table. For instance, in a table of users, the primary key might be a "user_id" that is unique to each user.

2. **Foreign Key**: A foreign key in one table is a field that matches the primary key column of another table. It creates a link between the two tables, indicating a relationship. For example, if there's another table called "orders" which includes a column "user_id" as a foreign key, it would point back to the "user_id" in the users table. This setup illustrates which user made each order.

Takeaways:

Key is something you use to get a value. (We often say key-value pair).

For example, if you want to get grades of students, you access `grade` column and now `grade` is the key.

Two kinds of keys: **primary key** and **foreign key**.

Each table typically has one primary key. The value associated with this key is different for each row in this table.

For example: in a grade file, the student_ID can be the primary key.

Note. One key does not have to be one attribute (name of one column), it can be combinations of attributes.

Composite Keys

A composite key is used when no single column uniquely identifies each row in a table, but a combination of columns does. For example, consider a table recording the enrollment of students in courses:

- **Student ID**: Identifies a student.
- **Course ID**: Identifies a course.

Neither column alone can uniquely identify a row in this table because a student can enroll in multiple courses and a course can have multiple students. However, the combination of **Student ID** and **Course ID** can uniquely identify each row, since a specific student can enroll in a specific course only once. Thus, a composite key comprising both **Student ID** and **Course ID** would be necessary and sufficient to uniquely identify each record in the enrollment table.

sql in ece472:

lab3, drill, project

Loading a table

The data we get are often `.csv` files, to use sqlite we need to turn it to tables, the data format that sql can recognize.

To create a table from a .csv file,

```
mkdir var
sqlite3 var/imdb.sqlite3
.quit
```

this will initialize the database and then quit; you can further preprocess the data e.g. remove the header.

then, run `sqlite3 var/imdb.sqlite3` and:

Create a table which defines the framework you want:

```
create table name
(
    nconst varchar(10) not null,
    primaryName text not null,
    birthYear varchar(4) not null,
    deathYear varchar(4) not null,
    primaryProfession text not null,
    knownForTitles text not null,
    primary key(nconst)
);
```

Then, import the data file into it:

```
.separator "\t"
.import name.basics.tsv name
```

This imports the data `name.basics.tsv` as a table `name`.

Basic Sql Syntax

SELECT ... FROM ...

```
SELECT [DISTINCT]
attr-list
FROM relation-list
WHERE qualification
```

OK it seems to be a little hard to understand, here is a simpler way to understand:

```
SELECT [I want it to be unique]
what I want
FROM the target dataset
WHERE I need them to satisfy these requirements
```

Still a little bit weird? Let's take an example.

- Schema:
  - Sailors (sid, sname, rating, age)
  - Boats (bid, bname, color)
  - Reserves (sid, bid, rday)
- **Query:** Find the names of sailors who have reserved boat #103

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid = 103;
```

[credit: EECS 484 24WN, University of Michigan]

What I want? (SELECT) Sailor's name!

What source(datasets) I will look at?(FROM) Sailors S, Reserves R

What requirements should be satisfied? (WHERE) S.sid = R.sid AND R.bid = 103

S,R are called range variables.

ORDER BY

It is common that we want to order the results we quiried:

```
SELECT S.sname, S.age
FROM Sailors S
ORDER BY S.age DESC
```

Attribute(s) in ORDER BY clause (must be) in SELECT list

Operators

After learning about the basic sql common commands, we may want to see some powerful operators.

Join

Consider the code

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid = 103;
```

It can be regarded as an **Inner Join**:

```
SELECT S.sname
FROM Sailors S INNER JOIN Reserves R ON S.sid = R.sid
WHERE R.bid = 103
```

Inner join: compares the values in the columns being joined (the inner part of Venn diagram)

There are also other kinds of join. e.g. full outer join.

Union

Consider the following problem:

Find names of sailors who have reserved a red **or** a green boat.

```
SELECT  DISTINCT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
  AND (B.color = 'red' OR B.color = 'green');
```

Understanding the code:

why we need

```
S.sid = R.sid AND R.bid = B.bid
```

It is like 2 steps of retrieval: which sailors reserved boat? which boat do the reservation reserves?

that may give you some ideas on what 'relative'/'linking' means.

Rewriting it using `UNION`:

```
SELECT  S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red' UNION
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid and R.bid = B.bid AND B.color = 'green';
```

Intersect

Let's modifty the above problem a little bit:

Find names of sailors who have reserved a red **and** a green boat.

It seems that **intersect** is the dual of **union**, so one may want to write:

```
SELECT  S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'

INTERSECT

SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid and R.bid = B.bid AND B.color = 'green';
```

- Schema:
  - Sailors (sid, sname, rating, age)
  - Boats (bid, bname, color)
  - Reserves (sid, bid, rday)
- **Query:** Find the names of sailors who have reserved boat #103

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid = 103;
```

looking back on this schema, do you see any problem?

What if two sailors coincidently have the same name? and one of them reserves red boat, and the other reserves green one?

This name will be selected! That is not what we want.

What is the underlying reason for this problem? **sname** might be **NOT unique**!

Takeaway : **Intersection on non-unique key is dangerous.**

How to solve the problem? Intersection on **unique** key!

What is the **key** that must be unique? **Primary Key**.

So the fixed code is as follows:

```
CREATE VIEW RedGreenSailors AS
SELECT  S.sid
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
INTERSECT
SELECT S.sid
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid and R.bid = B.bid AND B.color = 'green';

SELECT S.sname
FROM Sailors S, RedGreenSailors R
WHERE S.sid = R.sid;
DROP VIEW RedGreenSailors;
```

Or, without `VIEW`:

```
SELECT S.sname
FROM Sailors S,
(SELECT  S.sid
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
INTERSECT
SELECT S.sid
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid and R.bid = B.bid AND B.color = 'green')
RedGreenSailors
WHERE S.sid = RedGreenSailors.sid;
```

One lab's time is not enough to cover all the common sql commands e.g. `COUNT` , `GROUP BY`, `EXCEPT`, but it should provide a good start so that you are more prepared to learn these by yourselves.

As you may also see, the sql code becomes more and more complicated. To finish more complex tasks, we may want to integrate sql with other languages, such as Java and Python.

Workflow of Query using sql

Below are some *personal advice*:

1. Before you interact with a dataset, it is recommanded to look first at its documentation:

for IMDb: https://developer.imdb.com/non-commercial-datasets/

2. After you do the query. Think: Is the returning result valid? Validity not only relies on how it aligns with your program/expecation, but also **common sense**.

Lab Submissions

Please finish the ex3 and ex5 as depicted in the manual. Please hand in a pdf report (e.g. exported by markdown), together with your codes onto canvas. For the advanced queries, your implementation can either be in python or in java. **Lab 3 is an individual lab**.

Reference

[1] Atul Prakash and Lin Ma, EECS 484 WN2024, University of Michigan - Ann Arbor

[2] Kaiwen Zhang etc. VE482 Lab4 22FA.