Report on Lab Work 2

Lei Liu, 9669373

1. Three Pairs of SQL Queries

1.1 JOIN VS CROSS JOIN

```
A. select o.name as Organization,
o.established, c.name as country,
 c.capital as capital, cit.name as city,
cit.population from country as c,
      organization as o,
      ismember as m,
      city as cit
where c.code = o.country
       and o.country = m.country
       and o.abbreviation = m.organization
       and cit.name = o.city;

B. select o.name as Organization,
o.established, c.name as country,
c.capital as capital, cit.name as city,
cit.population from ismember as m
       cross join country as c
       cross join organization as o
       cross join city as cit
where c.code = o.country
       and o.country = m.country
       and o.abbreviation = m.organization
       and cit.name = o.city;
```

Here is a comparison between JOIN query and CROSS JOIN query. In Sqlite3, joins are implemented as nested loops. JOIN query will be optimized by reorder the tables in FROM clause which analyse those tables and choose the best efficient strategy. This basically means join tables from the smallest to the largest one, so that the outer loop is able to be run as less time as possible. However, CROSS JOIN will never reorder tables and join them from the left-most table to the right-most table, even join tables from the largest one which means the times of running outer loop could be very high. Thus, in general, JOIN query (query A) will be faster than CORSS JOIN (query B). Table 1 is the explanations of query plans.

Table 1: Details of query plans

selectid	order	from	detail
Query A: JOIN			
0	0	1	SCAN TABLE organization AS o (~168 rows)
0	1	2	SEARCH TABLE ismember AS m USING COVERIN
0	2	0	SEARCH TABLE country AS c USING INDEX sq
0	3	3	SEARCH TABLE city AS cit USING INDEX sql
Query B: CROSS JOIN			
0	0	0	SCAN TABLE ismember AS m USING COVERING INDEX sqlite_autoindex_isMember_1 (~9968 rows)
0	1	1	SEARCH TABLE country AS c USING INDEX sqlite_autoindex_Country_2 (Code=?) (~1 rows)
0	2	2	SEARCH TABLE organization AS o USING INDEX sqlite_autoindex_Organization_1 (Abbreviatiion)
0	3	3	SEARCH TABLE city AS cit USING INDEX sqlite_autoindex_City_1 (Name=?) (~2 rows)

Form the table, it is clear that JOIN query reorder the tables and begin join from table Organization, whereas CROSS JOIN still begin at Ismember table. Table 2 is the User Executing time for each query running ten times. This shows that JOIN outperformed than CROSS JOIN by 95% on average.

Table 2: User Time for Running Queries Ten Times

JOIN	CROSS JOIN
0.000958	0.013554
0.000887	0.014051
0	0.0142
0.0009	0.01301
0	0.014029
0.000796	0.01374
0.000837	0.013098
0.000973	0.014191
0.000129	0.013971
0.000818	0.014302

1.2 OR VS IN

```
A. select *
from airport
where city='Beijing'
        or city='Shanghai'
        or city='Guangzhou';
```

```
B. select *
from airport
where city in('Beijing', 'Shanghai',
'Guangzhou');
```

From SQLite Documents, there are two conditions that a query can apply OR optimizations. One is that all OR-connected terms are sharing a common column name. The optimizer will reconstruct the query as an IN query which could go on to constrain an index rather than scan the whole table. Another condition is that terms in a query are not containing the same column. This query will be optimized by applying different index for each OR clause term.

Table 3: User Time for Running Queries Ten Times

OR	IN
0.000946	0
0.000565	0.000138
0.000498	0.000555
0.000461	0.000731
0.000617	0
0.000593	0.000583
0.00078	0.000584
0.000759	0.000463
0.000687	0.000465
0.000476	0

Table 3 is the time for executing each queries ten times. The IN query seems perform better than the OR query by approximately 15% on average. The reason is that there was not index for the queried column.

1.3 LIKE VS AND

```
A. select * from city
where name like 'Man%';;
```

```
B. select * from city
where name>='Man'
        and name<'Mao';
```

Query B outperformed than Query A as there is an index for constraints. LIKE query will not use an index even if there is an index which accelerates searching. Table 4 is logging the time of for this pair of queries. It is obvious that an AND query with index executes much faster than LIKE query, and the percentage of improvement is about 76% on average.

Table 4: User Time for Running Queries Ten Times

like	and (index)
0.001195	0.000611
0.000371	0
0.001108	0.00047
0.001042	0.000223
0.000708	0
0.001013	0.000124
0.001016	0
0.001327	0
0.001193	0.000364
0.00109	0.000617

2. Summary

Firstly, as mentioned above, the sqlite3 optimizer will reorder tables in join clause and execute in a better strategy. This is done automatically, and the optimization results is efficient as shown in figure 1. This benefit allows anyone produce great performed queries but do not have to consider too much about joining tables.

Secondly, as shown in figure 2, IN query is generally faster than OR queries. In sqlite3, the two kinds of OR optimizer can apply separate indices or rewrite query as IN clause that accelerate searching. This makes the query more feasible and faster than without optimizer that scan the whole table and compare value several times. But, it basically needs the assistant of indices.

Besides, figure 3 illustrate again that a query applying index is much faster than without using any index. Fortunately, sqlite3 will automatically generate an index for each table according to their primary ked. This also optimize some queries, and most importantly, empower its optimizer to be used and work well.

In conclusion, the sqlite3 optimizer can produce different strategies for a query and choose the best one according to indices and its statistic of the whole database.

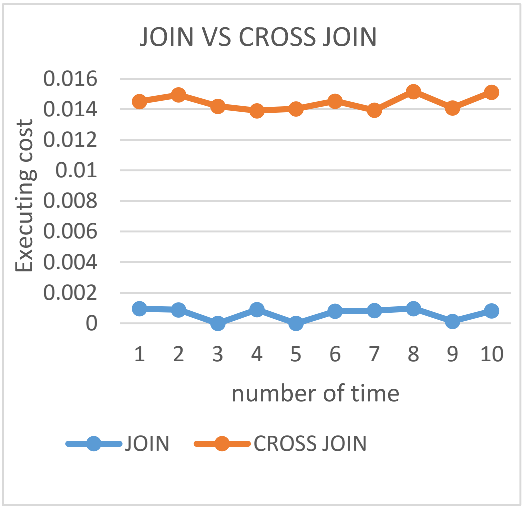

Figure 1: Comparison between JOIN and CROSS JOIN

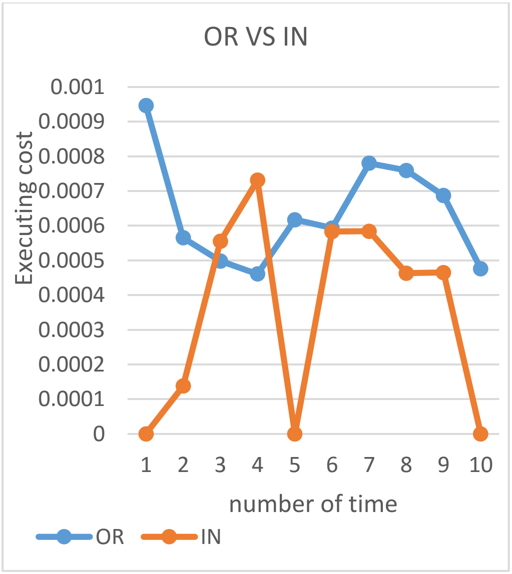

Figure 2: Comparison Between OR and IN

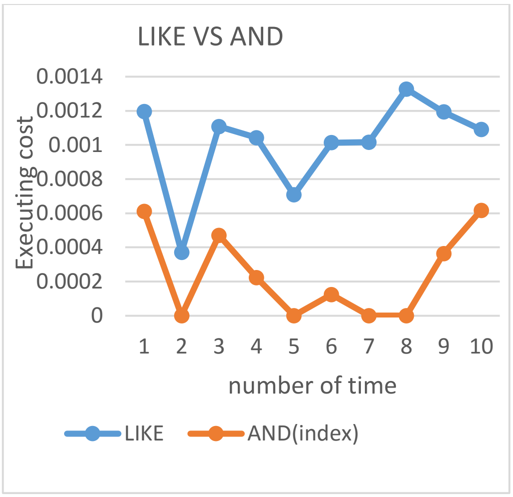

Figure 3: Comparison Between LIKE and AND (index)