CS3223 Database Systems Implementation

Project Write-up

Christian Drake Martin, A0200700H

Justin Tzuriel Krisnahadi, A0200794H

Pang Yong Jia, A0167896A

Features Implemented

- Block Nested Loops join
- Sort Merge join (with external sort-merge algorithm)
- DISTINCT
- ORDERBY including ASC (in ascending order) and DESC (in descending order)
- Aggregate functions (MIN, MAX, COUNT)

Bugs Found

- RandomDB is unable to generate data when the attributes are of type "REAL".

  When we want to create a table with the "REAL" attribute type, the .det file must contain the "REAL" keyword for that attribute. However, in the RandomDB.java file, to generate attributes of the "REAL" data type, it will check the .det file and see if the data type specified is equal to "FLOAT" instead of "REAL". This will prevent "REAL" attributes from being generated. A simple bug fix is to change the "FLOAT" to "REAL", and attributes of type "REAL" will then be able to be generated properly.

Experiment Details

For both experiments, we decided to use a buffer size of **600 bytes** and **10 buffer pages.**

The following are the details of the tables that we have created for our experiment, populated with random data.

Table	**# Records**	**Tuple Size (Bytes)**	**<attribute name> <data type> <range> <key type> <column size>**
Flights	15000	36	flno INTEGER 30000 PK 4 from STRING 3 NK 6 to STRING 3 NK 6 distance INTEGER 13000 NK 4 departs INTEGER 2400 NK 8 arrives INTEGER 2400 NK 8

Aircraft	15000	108	aid INTEGER 30000 PK 4 aname STRING 50 NK 100 cruisingrange INTEGER 4000 NK 4
Schedule	15000	8	flno INTEGER 30000 PK 4 aid INTEGER 30000 PK 4
Certified	15000	8	eid INTEGER 30000 PK 4 aid INTEGER 30000 PK 4
Employees	15000	108	eid INTEGER 30000 PK 4 ename STRING 50 NK 100 salary INTEGER 5000 NK 4

Experiment 1

1) Join of Employees and Certified (via eid)

SQL Query:

```
SELECT EMPLOYEES.eid,EMPLOYEES.ename, CERTIFIED.eid, CERTIFIED.aid, EMPLOYEES.salary
FROM EMPLOYEES, CERTIFIED
WHERE EMPLOYEES.eid=CERTIFIED.eid
```

Join Algorithm	Time (s)
Block Nested Loops	285.947
Sort Merge	11.432

2) Join of Flights and Schedule (via flno)

SQL Query:

```
SELECT *
FROM FLIGHTS, SCHEDULE
WHERE FLIGHTS.flno=SCHEDULE.flno
```

Join Algorithm	Time (s)
Block Nested Loops	41.846

Sort Merge	13.952

3) Join of Schedule and Aircrafts (via aid)

SQL Query:

```
SELECT SCHEDULE.flno, SCHEDULE.aid, AIRCRAFT.aid
FROM SCHEDULE, AIRCRAFT
WHERE SCHEDULE.aid=AIRCRAFT.aid
```

Join Algorithm	Time (s)
Block Nested Loops	30.229
Sort Merge	11.700

Observations of results:

For all three joins, Sort Merge executes faster than Block Nested Loops. This is perhaps due to the fact that when the join attribute is a primary key of one of the tables, in the merging phase of Sort Merge, both tables are read at most once, so the bulk of the cost comes from sorting the tables. Meanwhile, in Block Nested Loops, one table still has to be read repeatedly. Additionally, both Sort Merge and Block Nested Loops depend on the table size and the number of buffers. The difference is, since the cost model of Sort Merge uses a logarithmic function, the change of the table size or the number of buffers affects its performance less than it affects Block Nested Loops. In other words, for a large table with not a lot of buffers, Sort Merge would be expected to perform better.

Experiment 2

List of Pilots scheduled for a flight

SQL Query:

```
SELECT EMPLOYEES.eid, EMPLOYEES.ename, CERTIFIED.aid
FROM EMPLOYEES, CERTIFIED, SCHEDULE
WHERE EMPLOYEES.eid=CERTIFIED.eid, CERTIFIED.aid=SCHEDULE.aid
```

Execution Plan	Time (s)
(EMPLOYEES $\bowtie_{BNL}$ CERTIFIED) $\bowtie_{BNL}$ SCHEDULE	1158.976
EMPLOYEES$\bowtie_{BNL}$ (CERTIFIED $\bowtie_{BNL}$ SCHEDULE)	574.729
SCHEDULE $\bowtie_{BNL}$ (EMPLOYEES $\bowtie_{BNL}$ CERTIFIED)	779.155
(EMPLOYEES $\bowtie_{SM}$ CERTIFIED) $\bowtie_{SM}$ SCHEDULE	31.143
EMPLOYEES $\bowtie_{SM}$ (CERTIFIED $\bowtie_{SM}$ SCHEDULE)	32.270
SCHEDULE $\bowtie_{SM}$ (EMPLOYEES $\bowtie_{SM}$ CERTIFIED)	28.617

Observations of results:

Similar to Experiment 1, Sort Merge executes faster than Block Nested Loops. Aside from the algorithm used, the time taken depends on the execution plan (especially for Block Nested Loops). This is usually due to the size of the intermediate tables (where different plans would create different intermediate tables). Interestingly, the worst plans are different for Block Nested Loops and Sort Merge. Perhaps this is because the size of the intermediate table (EMPLOYEES $\bowtie_{BNL}$ CERTIFIED) is very large, and this affect Block Nested Loops much more than Sort Merge, as the number of scans of the right table depends on the size of the left table. With the same line of reasoning we can also see why the swapping of left/right tables (e.g. (EMPLOYEES $\bowtie$ CERTIFIED) $\bowtie$ SCHEDULE and SCHEDULE $\bowtie$ (EMPLOYEES $\bowtie$ CERTIFIED) affects Block Nested Loops greatly.

In a real-world system with larger data sets, the size of the intermediate tables could be very huge. Depending on the ordering of different query execution plans, we can have many different possibilities of intermediate tables. The difference in the size of intermediate tables may lead to very significant differences in the time taken to execute the queries. Query execution plan orderings with very small intermediate results will lead to shorter times in executing the queries, as all the join operations deal with a small table. On the contrary, plan orderings with very big intermediate results will lead to much longer times in executing the queries.