SOFTENG 351: Lab #10

June 4, 2020

Aiden Burgess
abur970 - 600280511

1.

No indices, so we can rule out hash join immediately. Can not use merge join as it requires ordering on the attributes.

Therefore, we must use block nested-loop join which will require:

$n_s + n_r \, IOs$

Where n_r is the number of blocks r is stored in, n_s is the number of blocks s is stored in.

The amount of memory needed is at least $min(n_s, n_r) + 1$ blocks in main memory as we need 1 block free for the output buffer.

2.

Simple translation: $\pi_{T.branch\_name}(\sigma_{T.assets>S.assets \wedge S.branch\_city="Brooklyn"}((\rho_T(branch) \times (\rho_S(branch)))$

Optimized: $\pi_{T.branch\_name}(\sigma_{T.assets>S.assets}(\rho_T(\pi_{branch\_name,\, assets}(branch)) \times \rho_S(\pi_{assets}(\sigma_{branch\_city="Brooklyn"}))$

In the optimized version, the selection and projection statements have been moved down as far as possible, so that operations higher in the hierarchy do not perform as much work.

3.

a)

SQL result

Host: Student mysql database
Database: stu_abur970_SOFTENG_351_C_S1_2020_A2_Q1
Generation Time: Jun 03, 2020 at 08:43 PM
Generated by: phpMyAdmin 4.0.10.7 / MySQL 5.1.73-log
SQL query: EXPLAIN SELECT * FROM DEPARTMENT WHERE Dname="Headquarters";
Rows: 1

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	DEPARTMENT	ALL	*NULL*	*NULL*	*NULL*	*NULL*	6	Using where

b)

SQL result

Host: Student mysql database
Database: stu_abur970_SOFTENG_351_C_S1_2020_A2_Q1
Generation Time: Jun 03, 2020 at 08:30 PM
Generated by: phpMyAdmin 4.0.10.7 / MySQL 5.1.73-log
SQL query: EXPLAIN SELECT * FROM DEPARTMENT WHERE Dname="Headquarters";
Rows: 1

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	DEPARTMENT	const	Dname	Dname	17	const	1	

c)

SQL result

Host: Student mysql database
Database: stu_abur970_SOFTENG_351_C_S1_2020_A2_Q1
Generation Time: Jun 03, 2020 at 09:13 PM
Generated by: phpMyAdmin 4.0.10.7 / MySQL 5.1.73-log
SQL query: EXPLAIN SELECT * FROM DEPARTMENT INNER JOIN EMPLOYEE ON Dnumber=Dno;
Rows: 2

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	DEPARTMENT	ALL	PRIMARY	*NULL*	*NULL*	*NULL*	6	
1	SIMPLE	EMPLOYEE	ref	Dno	Dno	4	stu_abur970_SOFTENG_351_C_S1_2020_A2_Q1.DEPARTMENT...	3	

d)

SQL result

Host: Student mysql database
Database: stu_abur970_SOFTENG_351_C_S1_2020_A2_Q1
Generation Time: Jun 03, 2020 at 09:16 PM
Generated by: phpMyAdmin 4.0.10.7 / MySQL 5.1.73-log
SQL query: EXPLAIN SELECT * FROM DEPENDENT INNER JOIN WORKS_ON WHERE Hours>8;
Rows: 2

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	DEPENDENT	ALL	*NULL*	*NULL*	*NULL*	*NULL*	11	
1	SIMPLE	WORKS_ON	ALL	*NULL*	*NULL*	*NULL*	*NULL*	48	Using where; Using join buffer

4.

The size of $r_1 \bowtie r_2 \bowtie r_3$ is estimated to be 1000 tuples, as C is a foreign key referencing r_2, and E is a foreign key referencing r_3.

The strategy for computing the join is to go through each tuple in $r_1$sequentially.

For each tuple (A, B, C):

- Lookup tuple for C in r_2. As C is the primary key of r_2, there can be at most one tuple. This can be done efficiently using an index.
- Lookup tuple for E in r_3. As E is the primary key of r_3, there can be at most one tuple. This can be done efficiently using an index.

5.

Consider the relations r1(A, B, C), r2(C, D, E), and r3(E, F) without primary keys, except the entire schema. LetV (C, r1) be 900,V (C, r2) be 1100,V (E, r2) be 50, and V (E, r3) be 100. Assume that r1 has 1000 tuples, r2 has 1500 tuples, and r3 has 750 tuples. Estimate the size of r1 ./ r2 ./ r3 and give an efficient strategy for computingthe join.

$estimate(|r_1 \bowtie r_2|) = min(n_{r_2} \times \frac{n_{r_1}}{V(C,r_1)}, n_{r_1} \times \frac{n_{r_2}}{V(C,r_2)}) = min(1667, 1364) = 1364\, tuples$

$estimate(|r_1 \bowtie r_2 \bowtie r_3|) = n_{r_1 \bowtie r_2} \times \frac{r_3}{V(E, r_3)} = 1364 \times \frac{750}{100} = 10230\, tuples$

6.

Atomicity

Transactions are either performed entirely or not at all. This prevents errors where transactions are stopped in the middle. These errors can violate integrity constraints on the database and be difficult to rollback.

Consistency Preservation

A transaction should take a database from one consistent state to another. Again, this prevents the database from reaching an invalid state which violates constraints. For applications like banking, if an invalid state occurs, many users could be impacted.

Isolation

A transaction should not be interfered by other transactions. If another transaction interfered with the current transaction, calculations could be wrong, leading to invalid database states. This maintains the integrity of the data and helps with concurrency control.

Durability/Permanency

A transaction's changes must be persistent in the database. Once a transaction has been completed, it can not be completely removed from the history of the database. It prevents physical issues like power outages or hardware failure to affect the database state.