Fr. Conceicao Rodrigues College of Engineering, Bandra (West)

CERTIFICATE

Certified that the term work in the subject Advanced Data Management Technology is completed.

By: **Mareena Mark Fernandes**

Roll no.: **8669**

Semester: **V**

Academic Year: **2020-2021**

Exam Seat No.:

____________________________ _______________________

Teacher in Charge Principal

INDEX

Subject: OLAP Lab

Lab Outcome

Upon successful completion of this module students should be able to:

LO1 -Implement simple query optimizers and design alternate efficient paths for query execution.
LO2-Simulate the working of concurrency protocols, recovery mechanisms in a database
LO3-Design applications using advanced models like mobile, spatial databases.
LO4-Implement query processing and transaction processing mechanisms.
LO5- Design Star schema, Snowflake schema and Fact constellation Schema.
LO6- Analyze data using OLAP operations so as to take strategic decisions

List of Experiment

SR. NO.	EXPERIMENT NAME	LO MAPPING
1.	To execute complex SQL queries in PostgreSQL	
2.	To study query evaluation plan.	LO1
3.	To implement query cost optimization	LO1
4.	To implement concurrency control algorithm	LO2
5.	To implement ARIES recovery algorithm	LO2
6.	To implement Data Fragmentation	LO4
7.	To implement Query Processing for distributed Databases	LO4
8.	Case study on Data warehouse construction (schema design)	LO5
9.	Implementation of OLAP queries	LO6
10	Case study on Mobile, Temporal and Spatial databases	LO3

Experiment No.: 1

1. SELECT SUM(capacity) FROM classroom;

2. SELECT course.course_id course.title FROM course, prereq WHERE course.course_id != prereq.course_id;

3. SELECT course_id id, title FROM course WHERE NOT EXISTS (SELECT DISTINCT * FROM prereq WHERE prereq.course_id = course.course_id);

Post labs:

1. SELECT DISTINCT name FROM student WHERE id IN
   (SELECT DISTINCT id FROM takes WHERE course_id IN
   (SELECT course_id FROM course WHERE dept_name = 'Comp. Sci.'));

2.
   A) SELECT d.dept_name, CASE WHEN MAX(i.salary) IS NULL THEN 0 ELSE MAX(i.salary) END AS max_salary FROM instructor i RIGHT OUTER JOIN department d ON i.dept_name=d.dept_name GROUP BY d.dept_name;

   B) CREATE TEMP VIEW dept_max_salary AS SELECT d.dept_name, CASE WHEN MAX(i.salary) IS NULL THEN 0 ELSE MAX(i.salary) END AS max_salary
   FROM instructor i RIGHT OUTER JOIN department d ON i.dept_name=d.dept_name GROUP BY d.dept_name;

   SELECT dept_name, max_salary AS min_dept_salary FROM dept_max_salary WHERE max_salary IN
   (SELECT MIN(max_salary) max_salary FROM dept_max_salary WHERE max_salary>0);

Experiment No.: 2

1. **EXPLAIN SELECT * FROM takes NATURAL JOIN student;**

   OUTPUT:

   i. "Hash Join (cost=30.19..380.90 rows=10340 width=272)"
   ii. " Hash Cond: ((takes.id)::text = (student.id)::text)"
   iii. " -> Seq Scan on takes (cost=0.00..323.40 rows=10340 width=144)"
   iv. " -> Hash (cost=21.75..21.75 rows=675 width=152)"
   v. "   -> Seq Scan on student (cost=0.00..21.75 rows=675 width=152)"

2. **EXPLAIN SELECT * from takes NATURAL JOIN student WHERE id = '36052';**

   OUTPUT:

   i. "Nested Loop (cost=4.96..136.98 rows=52 width=272)"
   ii. " -> Index Scan using student_pkey on student (cost=0.28..8.29 rows=1 width=152)"
   iii. "   Index Cond: ((id)::text = '36052'::text)"
   iv. " -> Bitmap Heap Scan on takes (cost=4.69..128.17 rows=52 width=144)"
   v. "   Recheck Cond: ((id)::text = '36052'::text)"
   vi. "   -> Bitmap Index Scan on takes_pkey (cost=0.00..4.67 rows=52 width=0)"
   vii. "   Index Cond: ((id)::text = '36052'::text)"

3. **EXPLAIN SELECT id, COUNT(*) FROM takes GROUP BY id;**

   OUTPUT:

   I. "HashAggregate (cost=375.10..377.10 rows=200 width=32)"
   II. " Group Key: id"
   III. " -> Seq Scan on takes (cost=0.00..323.40 rows=10340 width=24)"

4. **EXPLAIN SELECT * FROM student , instructor WHERE student.id = instructor.id AND student.id = '24746';**

   OUTPUT:

   i. "Nested Loop (cost=0.42..16.47 rows=1 width=306)"
   ii. " -> Index Scan using student_pkey on student (cost=0.28..8.29 rows=1 width=152)"
   iii. "   Index Cond: ((id)::text = '24746'::text)"
   iv. " -> Index Scan using instructor_pkey on instructor (cost=0.15..8.17 rows=1 width=154)"
   v. "   Index Cond: ((id)::text = '24746'::text)"

5. **EXPLAIN SELECT * FROM student, instructor WHERE UPPER (student.id) = UPPER (instructor.id) AND student.id = '72014';**

   OUTPUT:

   i. "Hash Join (cost=8.30..24.00 rows=2 width=306)"
   ii. " Hash Cond: (upper((instructor.id)::text) = upper((student.id)::text))"
   iii. " -> Seq Scan on instructor (cost=0.00..14.40 rows=440 width=154)"
   iv. " -> Hash (cost=8.29..8.29 rows=1 width=152)"
   v. "   -> Index Scan using student_pkey on student (cost=0.28..8.29 rows=1 width=152)"
   vi. "   Index Cond: ((id)::text = '72014'::text)"

Post labs:

1. **Translate the following queries to relational algebra**
   i. Select movieTitle from StarsIn, MovieStar where starName=name and birthdate=1960

Ans:

$$\Pi_{movieTitle}(\sigma_{starName=name \wedge birthdate=1960}(StarsIn \bowtie MovieStar))$$

   ii. Select name from MovieStar where birthdate=1960 and name=starName

Ans:

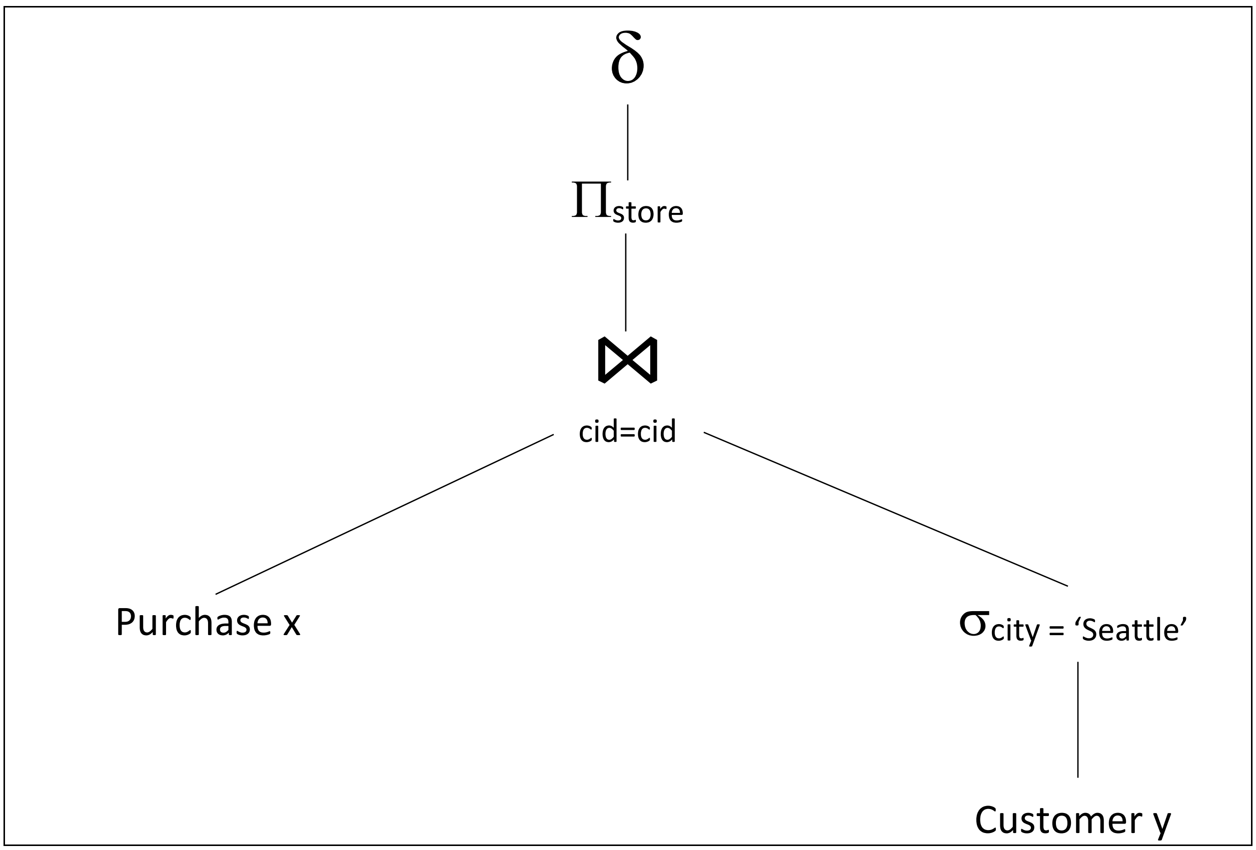
$$\Pi_{name}(\sigma_{birthdate=1960 \wedge name=starName}(MovieStar))$$

2. Given this database schema, draw a logical query plan for the queries
   Schema:
   Product (<u>pid</u>, name, price)
   Purchase (<u>pid</u>, <u>cid</u>, store)
   Customer (<u>cid</u>, name, city)

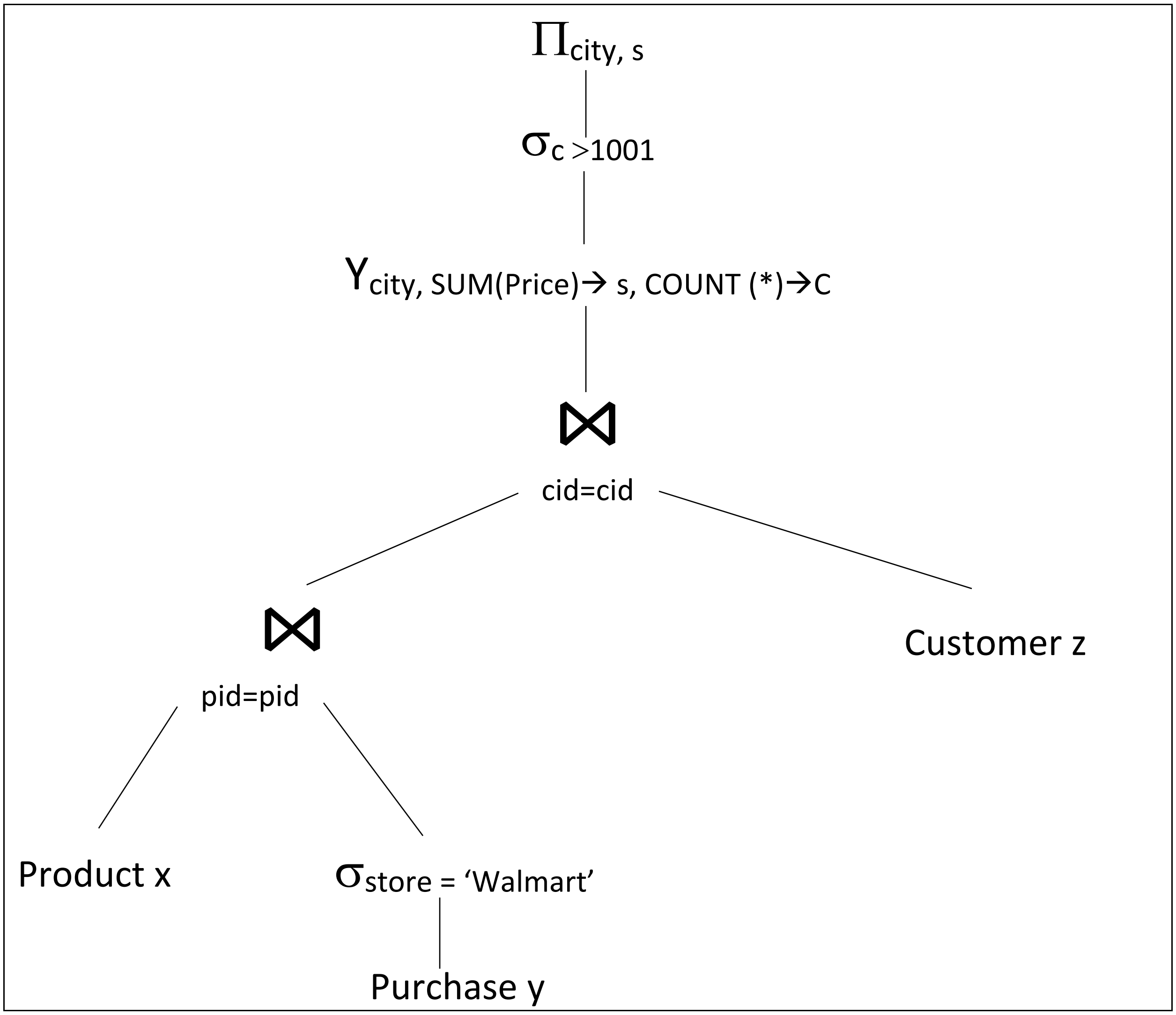
   i. Select distinct x.store from purchase x, customer y where x.cid=y.cid and y.city= 'Seattle'
   ii. Select z.city, sum(x.price) from product x, purchase y, customer z where x.pid=y.pid and y.cid=z.cid and y.store= 'Walmart' groupby z.city having count(*) >100

Ans:

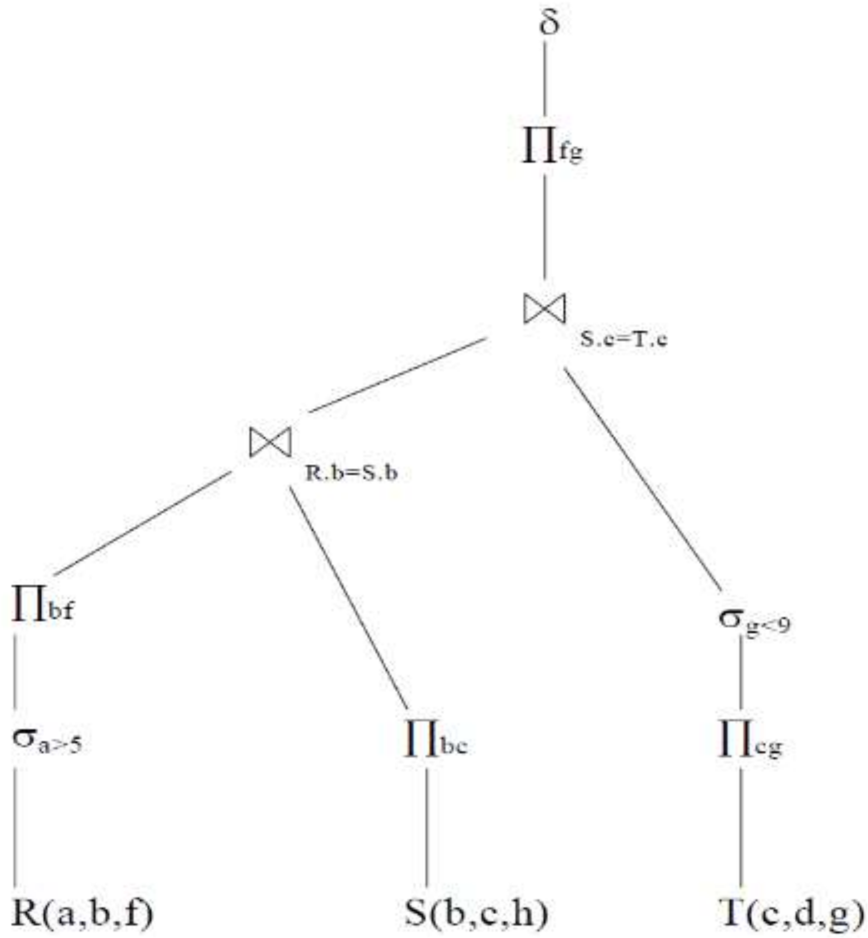
i.



ii.


$\Pi_{city,\ s}$
$\sigma_{c > 1001}$
$\gamma_{city,\ SUM(Price) \rightarrow s,\ COUNT(*) \rightarrow C}$
$\bowtie$
cid=cid
$\bowtie$
pid=pid
Customer z
Product x
$\sigma_{store\ =\ 'Walmart'}$
Purchase y

3. Write a SQL query that is equivalent to the logical plan below:



Ans:

Select distant R.f, T.g from R,S,T where R.a > 5 and T.g < 9 and R.b > S.b and S.c =T.c;

Experiment No. 3

1.

Query1] **EXPLAIN SELECT name FROM instructor WHERE EXISTS (SELECT * FROM teaches WHERE instructor.ID = teaches.ID AND teaches.year = 2007);**

OUTPUT:

i. "Hash Semi Join (cost=16.41..32.00 rows=3 width=58)"
ii. " Hash Cond: ((instructor.id)::text = (teaches.id)::text)"
iii. " -> Seq Scan on instructor (cost=0.00..14.40 rows=440 width=82)"
iv. " -> Hash (cost=16.38..16.38 rows=3 width=24)"
v. "   -> Seq Scan on teaches (cost=0.00..16.38 rows=3 width=24)"
vi. "   Filter: (year = '2007'::numeric)"

Query2] **EXPLAIN SELECT DISTINCT name FROM instructor, teaches WHERE instructor.ID = teaches.ID AND teaches.year = 2007;**

OUTPUT:

i. "Unique (cost=32.52..32.53 rows=3 width=58)"
ii. " -> Sort (cost=32.52..32.52 rows=3 width=58)"
iii. "   Sort Key: instructor.name"
iv. "   -> Hash Join (cost=16.41..32.49 rows=3 width=58)"
v. "   Hash Cond: ((instructor.id)::text = (teaches.id)::text)"
vi. "   -> Seq Scan on instructor (cost=0.00..14.40 rows=440 width=82)"
vii. "   -> Hash (cost=16.38..16.38 rows=3 width=24)"
viii. "   -> Seq Scan on teaches (cost=0.00..16.38 rows=3 width=24)"
ix. "   Filter: (year = '2007'::numeric)"

Since cost of Query1<Query2, Query1 is the optimized query.

2. A

Query1] **EXPLAIN SELECT * FROM student, instructor WHERE student.id = instructor.id AND student.id = '3335';**

OUTPUT:

i. "Nested Loop (cost=0.42..16.47 rows=1 width=306)"
ii. " -> Index Scan using student_pkey on student (cost=0.28..8.29 rows=1 width=152)"
iii. "   Index Cond: ((id)::text = ‘3335’::text)"
iv. " -> Index Scan using instructor_pkey on instructor (cost=0.15..8.17 rows=1 width=154)"
v. "   Index Cond: ((id)::text = ‘3335’::text)"

Query2] **EXPLAIN SELECT * FROM student, instructor WHERE instructor.id IN (SELECT instructor.id FROM instructor WHERE instructor.id = student.id AND student.id = '3335');**

OUTPUT:

i. "Nested Loop (cost=0.00..1239269.75 rows=148500 width=306)"
ii. " Join Filter: (SubPlan 1)"
iii. " -> Seq Scan on student (cost=0.00..21.75 rows=675 width=152)"
iv. " -> Materialize (cost=0.00..16.60 rows=440 width=154)"
v. "   -> Seq Scan on instructor (cost=0.00..14.40 rows=440 width=154)"
vi. " SubPlan 1"
vii. "   -> Result (cost=0.15..8.17 rows=1 width=24)"
viii. "   One-Time Filter: ((student.id)::text = '3335'::text)"
ix. "   -> Index Only Scan using instructor_pkey on instructor instructor_1 (cost=0.15..8.17 rows=1 width=24)"
x. "   Index Cond: (id = (student.id)::text)"

<u>Since cost of Query1<Query2, Query1 is the optimized query.</u>

2. B

Query1] **EXPLAIN SELECT * FROM student, instructor WHERE UPPER(student.id) = UPPER(instructor.id) AND student.id = '3335';**

OUTPUT:

i. "Hash Join (cost=8.30..24.00 rows=2 width=306)"
ii. " Hash Cond: (upper((instructor.id)::text) = upper((student.id)::text))"
iii. " -> Seq Scan on instructor (cost=0.00..14.40 rows=440 width=154)"
iv. " -> Hash (cost=8.29..8.29 rows=1 width=152)"
v. "   -> Index Scan using student_pkey on student (cost=0.28..8.29 rows=1 width=152)"
vi. "   Index Cond: ((id)::text = '3335'::text)"

***Query2]* EXPLAIN SELECT * FROM student, instructor WHERE instructor.id IN (SELECT instructor.id FROM instructor WHERE UPPER(student.id) = UPPER(instructor.id) AND student.id = '3335');**

OUTPUT:

i. "Nested Loop (cost=0.00..2634427.25 rows=148500 width=306)"
ii. " Join Filter: (SubPlan 1)"
iii. " -> Seq Scan on student (cost=0.00..21.75 rows=675 width=152)"
iv. " -> Materialize (cost=0.00..16.60 rows=440 width=154)"
v. "   -> Seq Scan on instructor (cost=0.00..14.40 rows=440 width=154)"
vi. " SubPlan 1"
vii. "   -> Result (cost=0.00..17.70 rows=2 width=24)"
viii. "   One-Time Filter: ((student.id)::text = '3335'::text)"
ix. "   -> Seq Scan on instructor instructor_1 (cost=0.00..17.70 rows=2 width=24)"
x. "   Filter: (upper((student.id)::text) = upper((id)::text))"

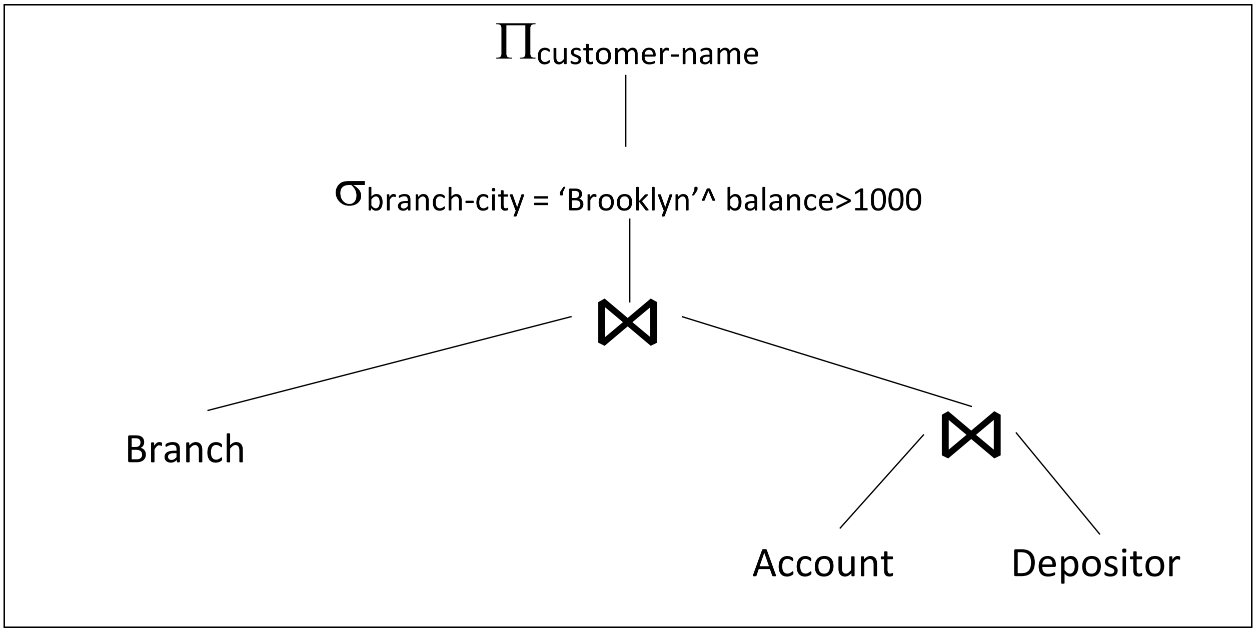
<u>Since cost of Query1<Query2, Query1 is the optimized query.</u>

Post labs:

1. Convert the following query into logically equivalent queries using equivalence rules.

$$\Pi_{customer-name}\ (\sigma_{branch-city = \text{"Brooklyn"} \ \wedge \ balance > 1000}\ (branch \bowtie (account \bowtie depositor)))$$

Ans:

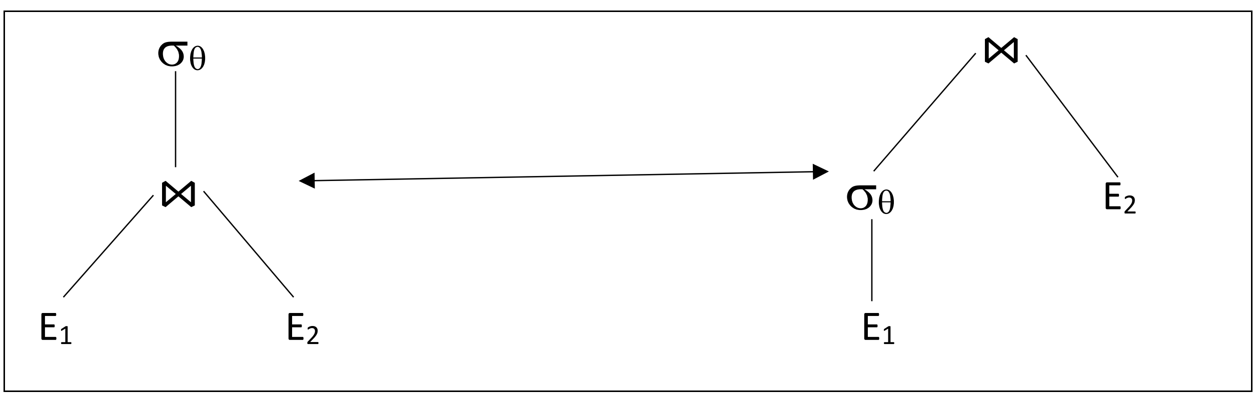


Query transformation using join associativity:

Natural joins are associative:

$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

Example:

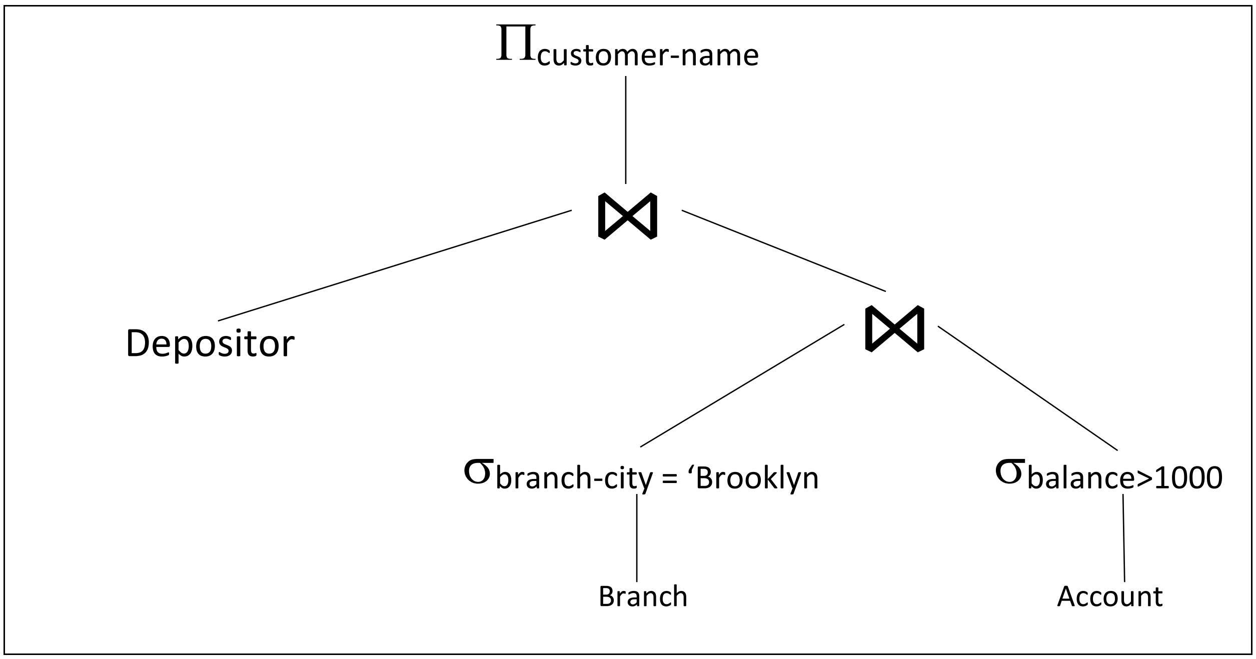


Second form provides an opportunity to apply the "perform selections early" rule resulting in the sub expression.

$$\sigma_{\text{branch-city = 'Brooklyn}}(\text{branch}) \bowtie \sigma_{\text{balance>1000}}(\text{account})$$

∴ Logically equivalent query is,

$$\Pi_{\text{customer-name}} ((\sigma_{\text{branch-city = 'Brooklyn}}(\text{branch}) \bowtie \sigma_{\text{balance>1000}}(\text{account})) \bowtie \text{depositor})$$



2. **Consider the following database**
   a. **Write a nested query on the relation account to find, for each branch with name starting with B, all accounts with the maximum balance at the branch.**
   b. **Rewrite the preceding query, without using a nested subquery; in other words, decorrelate the query**

Ans:

a. Select A.account_number from account A where A.branch_name like 'B%' and A.balance = (Select max(B.balance) from account B where B.branch_name = A.branch_name);
b. Create view V1 as (Select branch_name, max(balance) from account group by branch)name);
   Select account_number from account, V1 where account.branch_name like 'B%' and account.branch_name = V1.branch_name and account.balance = V1.balance;