Part A. 2.5 a. $\Pi_{person\_name}(\sigma_{company\_name = \text{"First Bank Corporation"}}(works))$

b. $\Pi_{person\_name, city}(\sigma_{company\_name = \text{"First Bank Corporation"}}(works) \bowtie employee)$

c. $\Pi_{person\_name, street, city}(\sigma_{company\_name = \text{"First Bank Corporation"} \wedge salary > 10,000}(works \bowtie employee))$

d. $\Pi_{person\_name}(employee \bowtie works \bowtie company)$

e. $\Pi_{company\_name}(company) \div \Pi_{city}(\sigma_{company\_name = \text{"Small Bank Corporation"}}(company))$

2.6 $\Pi_{customer\_name, customer\_city, loan\_number, amount}(borrower \bowtie loan \bowtie customer)$

a. Jackson does not appear in the result because there is no tuple in the customer relation with Jackson's name and his city and street. The natural join operator uses the Cartesian product and select operations, where borrower × customer but only the tuples where borrower.customer_name = customer.customer name are selected. Since there is no tuple in customer with Jackson's name, there will be no tuple in borrower × customer where borrower.customer_name = Jackson and customer.customer_name = Jackson. Thus, Jackson will not appear in the result.

b. We could modify the customer relation by inserting a tuple with Customer_name = Jackson. If Jackson's city and street are unknown, we can use a null value for city and street, or another special value (eg. N/A) that is not a name for an actual city / street.

c. $\Pi_{customer\_name, customer\_city, loan\_number, amount}((borrower \bowtie loan) \bowtie customer)$

2.7 a.

$works \leftarrow \Pi_{person\_name, company\_name, (salary * 1.1)}(\sigma_{works.company\_name = \text{"First Bank Corporation"}}(works)) \cup \sigma_{company\_name \neq \text{"First Bank Corporation"}}(works)$

b. $temp \leftarrow \Pi_{works.person\_name \text{ as } person\_name, company\_name, salary}(\sigma_{manager\_name = works.person\_name}(works \times managers))$

$updated \leftarrow \Pi_{person\_name, company\_name, salary * 1.1}(\sigma_{temp.salary * 1.1 \leq 100000}(temp)) \cup \Pi_{person\_name, company\_name, salary * 1.03}(\sigma_{temp.salary * 1.1 > 100000}(temp))$

$works \leftarrow (works - temp) \cup updated$

c. $works \leftarrow works - \sigma_{company\_name = \text{"Small Bank Corporation"}}(works)$

2.8 a. counts $\leftarrow {}_{account\_number}G_{count(customer\_name)}(depositor)$

$\Pi_{account\_number}(\sigma_{customer\_count > 2}(\rho_{counts(account\_number,\ customer\_count)}(counts)))$

b. product $\leftarrow \rho_{t1}(depositor) \times \rho_{t2}(depositor) \times \rho_{t3}(depositor)$

accounts $\leftarrow \sigma_{t1.account\_number = t2.account\_number \wedge t2.account\_number = t3.account\_number}(product)$

$\Pi_{t1.account\_number}(\sigma_{t1.customer\_name \neq t2.customer\_name \wedge t2.customer\_name \neq t3.customer\_name \wedge t1.customer\_name \neq t3.customer\_name}(accounts))$

2.9 a. counts $\leftarrow {}_{company\_name}G_{count\text{-}distinct(person\_name)\ as\ employee\_count}(works)$

max_count $\leftarrow G_{max(employee\_count)\ as\ max}(counts)$

$\Pi_{company\_name}(\sigma_{employee\_count = max}(counts \times max))$

b. payrolls $\leftarrow {}_{company\_name}G_{sum(salary)\ as\ payroll}(works)$

min_payroll $\leftarrow min_{(payroll)\ as\ min}(payrolls)$

$\Pi_{company\_name}(\sigma_{payroll = min\_payroll}(payrolls \times min\_payroll))$

c. avg_pays $\leftarrow {}_{company\_name}G_{avg(salary)\ as\ pay}(works)$

first_bank $\leftarrow \sigma_{company\_name = \text{"First Bank Corporation"}}(avg\_pays)$

product $\leftarrow$ avg_pays $\times$ first_bank

$\Pi_{avg\_pays.company\_name}(\sigma_{avg\_pays.pay > first\_bank.pay}(product))$

Part B

a) R = customer_likes, S = customer_drinks

R-S = name, S = drink

$\Pi_{R-S}(r)$ = all 4 names, $\Pi_{R-S}(r) \times S$ = Cartesian product of all names and drinks excluding tea since tea is not in customer_drinks (↑ in customer_drinks)

$\Pi_{R-S,S}(r)$ = all tuples in customer_likes

$\Pi_{R-S}(r) \times S - \Pi_{R-S,S}(r)$ = difference of cartesian product and actual tuples, thus all tuples of the original customer_likes relation would not be in this result. This result will be the names of the customers and the drinks those they do not like. Archie and Pascal do not like all of espresso, coffee, and mocha, so the result will still contain (Archie, coffee) and (Pascal, espresso). Eli and Oppa will have tuples corresponding to all drinks in customer_drinks in both $\Pi_{R-S}(r) \times S$ and $\Pi_{R-S,S}(r)$. They will also be in $\Pi_{R-S}(r) \times S$, but the − operation will exclude (Oppa, tea) thus, Eli and Oppa are not in the result.

$\rightarrow \Pi_{R-S}((\Pi_{R-S}(r) \times S) - \Pi_{R-S,S}(r)) =$ | Archie / Pascal |

$\Pi_{R-S}(r) =$ | Archie / Pascal / Eli / Oppa |

So $\Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times S) - \Pi_{R-S,S}(r))$

$=$ | Eli / Oppa |

b) $r \mid_E s = r \mid s - \Pi_{R-S}(\Pi_{R-S,S}(r) - (\Pi_{R-S}(r) \times S))$

We know that $\Pi_{R-S,S}(r)$ is all the tuples in r with their attributes reordered. $\Pi_{R-S}(r) \times S$ is the cartesian product of $\Pi_{R-S}(r)$ and S, which is the cartesian product of rows in r with attributes only in R and not S with all the rows in S. Thus, $\Pi_{R-S,S}(r) - (\Pi_{R-S}(r) \times S)$ will only have the tuples that are not in $\Pi_{R-S}(r) \times S$. In our case, (Oppa, tea) is not in $\Pi_{R-S}(r) \times S$. Then, we subtract these tuples with attributes of R-S from our original division, which means that $r \mid_E s$ will exactly match the cartesian of S. In our case, $r \mid s$ gives {(Eli), (Oppa)} but $\Pi_{R-S}(\Pi_{R-S,S}(r) - (\Pi_{R-S}(r) \times S))$ will give {(Oppa)} since {(Oppa, tea)} does not match customer_drinks. Thus, we will end up with {(Eli)}.

Part B c.

σ customer-likes ⋈ customer-drinks

name $G_{count(drinks)}$(likes ⋈ drinks) =

Archie	2
Eli	3
Pascal	2
Oppa	3

$G_{count(drinks)}$(drinks) = 3

Oppa	3
Eli	3

$\Pi_{name}(_{name}G_{count(drinks)}(likes \bowtie drinks) \bowtie G_{count(drinks)}(drinks))$

$$\Pi_{R-S}(\sigma_{count1=count2}(_{R-S}G_{count(S)\ as\ count1}(r \bowtie s) \times G_{count(S)\ as\ count2}(s)))$$

Part C

a) Yes. $\sigma_\theta(_AG_F(r))$: we select tuples from $_AG_F(r)$ where θ only uses attributes from A. Since we have $_AG_F(r)$, the aggregate functions will not act on attributes from A because we group on A. Thus, the tuples will have the same values for the attributes in A. This is same as $_AG_F(\sigma_\theta(r))$ because when we select tuples with θ, those attributes are the grouping attributes. This will not modify the grouping attributes, so the grouping will still be the same as $\sigma_\theta(_AG_F(r))$.

b) No. Consider the example:

r =

name	drink
Eli	coffee
Eli	mocha
Oppa	coffee

s =

name	drink
Eli	coffee
Pascal	coffee
Pascal	espresso

, A = name

$\Pi_A(r-s) = \Pi_{name}($

name	drink
Eli	mocha
Oppa	coffee

$) =$

name
Eli
Pascal

$\Pi_A(r) - \Pi_A(s) =$

Eli
Oppa

$-$

Eli
Pascal

$=$

name
Oppa

Part C c) No. Consider

r =

name	drink
Eli	coffee

s =

name	state
Oppa	CA

t =

name	city
Eli	LA

$(r \bowtie s) \bowtie t =$

name	drink	state
Eli	coffee	null

$\bowtie$

name	city
Eli	LA

$=$

name	drink	state	city
Eli	coffee	null	LA

$r \bowtie (s \bowtie t) =$

name	drink
Eli	coffee

$\bowtie$

name	state	city
Oppa	CA	null

$=$

name	drink	state	city
Eli	coffee	null	null

d) Yes. We only select tuples based on attributes from r. The left outer join will not exclude any tuples from r in $r \bowtie s$, thus the tuples we select from this will have the same values for attributes from r as $\sigma_\theta(r)$. If we select then left join, the same tuples from s will be excluded as in $\sigma_\theta(r \bowtie s)$ since θ is only based on attributes from r.

e) No.

r =

name	b_1
Eli	1
Oppa	2

s =

name	b_2
Eli	3
Oppa	4

$\theta : b_2 = 3$

$\sigma_\theta(r \bowtie s) =$

name	b_1	b_2
Eli	1	3

$r \bowtie \sigma_\theta(s) =$

name	b_1
Eli	1
Oppa	2

$\bowtie$

name	b_2
Eli	3

$=$

name	b_1	b_2
Eli	1	3
Oppa	2	null

Feedback Survey submitted :)