2.5

(a)

$$\Pi_{person\_name}(\sigma_{company\_name=\text{"First Bank Corporation"}}(works))$$

(b)

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

(c)

$$\Pi_{person\_name,city,street}(\sigma_{company\_name=\text{"First Bank Corporation"} \wedge salary>10000}(employee \bowtie works))$$

(d)

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

(e)

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

2.6

The rewritten query would be

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

(a) Because Jackson is not in the customer relation (under *customer_name*)

(b) I would either add Jackson to the customer relation or add a *customer_city* attribute to the borrower scheme.

(c)

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

2.7

(a)

$$works \leftarrow \Pi_{person\_name,company\_name,(salary*1.10)}(\sigma_{company\_name=\text{"First Bank Corporation"}}(works)) \cup$$
$$\sigma_{company\_name\neq\text{"First Bank Corporation"}}(works)$$

(b)

$$temp1 \leftarrow \Pi_{person\_name \text{ as } manager\_name,company\_name,salary}(works)$$
$$temp2 \leftarrow temp1 \bowtie manages$$
$$temp3 \leftarrow \Pi_{manager\_name \text{ as } person\_name,company\_name,salary}(temp2)$$
$$works \leftarrow \Pi_{person\_name,company\_name,(salary*1.10)}(\sigma_{(salary*1.10)\leq 100000}(temp3)) \cup$$
$$\Pi_{person\_name,company\_name,(salary*1.03)}(\sigma_{(salary*1.10)>100000}(temp3)) \cup (works - temp3)$$

(c)

$$works \leftarrow \sigma_{company\_name\neq\text{"Small Bank Corporation"}}(works)$$

2.8

(a)

$$\Pi_{account\_number}(\sigma_{customer\_count>2}(_{account\_number}\mathcal{G}_{count(customer\_name) \text{ as } customer\_count}(depositor)))$$

(b)

$$\Pi_{account\_number}(\sigma_{a.account\_number=b.account\_number \wedge b.account\_number=c.account\_number}$$
$$\wedge a.customer\_name\neq b.customer\_name \wedge b.customer\_name\neq c.customer\_name$$
$$\wedge a.customer\_name\neq c.customer\_name(\rho_a(depositor) \times \rho_b(depositor) \times \rho_c depositor))$$

2.9

(a)

$$temp1 \leftarrow_{company\_name} \mathcal{G}_{count(person\_name) \text{ as } person\_count}(works)$$

$$max = \mathcal{G}_{max(person\_count) \text{ as } max\_count}(temp1)$$

$$temp2 = temp1 \times max$$

$$\Pi_{company\_name}(\sigma_{person\_count=max\_count}(temp2))$$

(b)

$$temp1 \leftarrow_{company\_name} \mathcal{G}_{sum(salary) \text{ as } payroll}(works)$$

$$min = \mathcal{G}_{min(payroll) \text{ as } min\_payroll}(temp1)$$

$$temp2 = temp1 \times min$$

$$\Pi_{company\_name}(\sigma_{payroll=min\_payroll}(temp1))$$

(c)

$$first\_avg \leftarrow \mathcal{G}_{avg(salary) \text{ as } first\_bank\_avg\_salary}(\sigma_{company\_name=\text{"First Bank Corporation"}}(works))$$

$$avg\_salaries \leftarrow_{company\_name} \mathcal{G}_{avg(salary) \text{ as } avg\_salary}(works)$$

$$temp1 \leftarrow avg\_salaries \times first\_avg$$

$$\Pi_{company\_name}(\sigma_{avg\_salary>first\_bank\_avg\_salary}(avg\_salaries))$$

Relational Division Operations

(a) Let us label the following relations in the definition of relational division as follows

- $a = \Pi_{R-S}(r) \times s$
- $b = \Pi_{R-S,S}(r)$
- $c = \Pi_{R-S}(a - b)$
- $d = \Pi_{R-S}(r)$

Now for the explanation. a is all possible combinations of names and foods, and b is just $monkey\_likes$, but with the attributes ordered to be identical to that produced by the expression $(\Pi_{R-S}(r) \times s)$. Given this, we have that c contains all the names that do not have an entry for every food in $monkey\_foods$. This is because if a name does have an entry for every food in $monkey\_foods$, then all those tuples will cancel with the tuples from a (which contains all the combinations of names and foods). Then we just subtract these names from d, which just projects the names of the monkeys from r. So clearly, we are left only with the names of the monkeys that *do* have an entry for every food in s.

Now to explain more specifically why Guenter would appear in the result. Guenter has an entry for every food in $monkey\_foods$. In other words, all possible combinations of Guenter and food are present in $monkey\_likes$. So when we do $(a - b)$, no tuples will have the name Guenter, because all the rows with his name will subtract out. Note that the tuple (Guenter, tofu) doesn't matter here, because tofu does not appear in the $food$ column of $monkey\_foods$. All that does matter is that Guenter has an entry for every food in $monkey\_foods$, so that the subtraction $a - b$ kills his name. So then, when we project the names in c, then subtract c from d, we won't subtract the name Guenter, and Guenter will appear in the final result.

(b)

$$r \div_E s = r \div s - \Pi_{R-S}(\Pi_{R-S,S}(r) - (\Pi_{R-S}(r) \times s))$$

We can see that this works for the following reason. The second subtraction subtracts the Cartesian product from the reordered first relation r. This leaves only the tuples from r that do not appear in the Cartesian product of $\Pi_{R-S}(r)$ and s. In other words, this leaves the tuples that do not

match the contents of s. So then we can just subtract them from the regular division and get our exact division.

In terms of the example, the right side of the second subtraction gives all combinations of names and foods. So when we subtract this from the left side of the second subtraction, we are left only with the names and foods that are not valid, i.e. those names that are paired with foods not in $monkey\_foods$.

(c)

$$\Pi_{R-S}({}_{R-S}\mathcal{G}_{count(S) \text{ as } num}(r \bowtie s) \bowtie \mathcal{G}_{count(S) \text{ as } num}(s))$$

Query Optimization and Equivalence Rules

(a) They are equivalent.

This is because when we select before we group, we select a set of tuples based on a predicate θ using only attributes from A. So when we select using θ, we select sets of tuples that would be groups. That is, if we select a tuple t that would be in a group g, we also select all other tuples in g. This is because all tuples in g have the same grouping attributes, and our predicate θ only uses attributes from A, the grouping attributes.

The other way around, we group, then select. So this way, we select sets of tuples that are groups.

So overall, we select the same rows either way, since in both ways we select the same groups of tuples.

(b) They are not equivalent. Here is a counterexample.

Relation r

Person	**Rating**
Sid	0
Sid	1
Connor	2

Relation s

Person	**Rating**
Sid	0
Matt	3

We have that $\Pi_A(r - s)$, if A projects on **Person**, would project Sid, Connor, and Matt, while $\Pi_A(r) - \Pi_A(s)$ would project only Connor and Matt.

(c) They are not equivalent. Here is a counterexample.

Relation r

a	**b1**
1	2

Relation s

a	**b2**
0	3

Relation t

a	**b3**
1	4

We have that $(r \rtimes s) \rtimes t$ gives us a relation with a tuple $(1, 2, null, 4)$, while $r \rtimes (s \rtimes t)$ gives us a relation with a tuple $1, 2, null, null$.

(d) They are equivalent. This is because θ is a predicate that uses attributes only from r, and the left outer join keeps all the tuples from r, which means that we will select the same tuples if we select from r and then left outer join with s as if we left outer join r with s and then select tuples. In other words, $r \rtimes s$ includes every tuple from r in one of its tuples. So selecting on θ, which only uses attributes from r, selects the same tuples whether it is applied before or after the join.

(e) They are not equivalent. Here is a counterexample.

Relation r

Store
Walmart
Costco

Relation s

Store	**Product**
Walmart	Boxes
Costco	Bags

We have that $\sigma_\theta(r \rtimes s)$, if θ is the predicate $Product =$ "Boxes", is just the tuple Walmart, Boxes. On the other hand, we have that $r \rtimes \sigma_\theta(s)$ has the tuple Walmart, Boxes and the tuple Costco, null.