Solutions to Practice Exercises

14.1 a. $E_1 \nabla_\theta (E_2 - E_3) = (E_1 \nabla_\theta E_2 - E_1 \nabla_\theta E_3)$.

Let us rename $(E_1 \nabla_\theta (E_2 - E_3))$ as $R_1$, $(E_1 \nabla_\theta E_2)$ as $R_2$ and $(E_1 \nabla_\theta E_3)$ as $R_3$. It is clear that if a tuple $t$ belongs to $R_1$, it will also belong to $R_2$. If a tuple $t$ belongs to $R_3$, $t[E_3$’s attributes] will belong to $E_3$, hence $t$ cannot belong to $R_1$. From these two we can say that

$$\forall t, t \in R_1 \implies t \in (R_2 - R_3)$$

It is clear that if a tuple $t$ belongs to $R_2 - R_3$, then $t[R_3$’s attributes] $\in E_2$ and $t[R_2$’s attributes] $\notin E_3$. Therefore:

$$\forall t, t \in (R_2 - R_3) \implies t \in R_1$$

The above two equations imply the given equivalence.

This equivalence is helpful because evaluation of the right hand side join will produce many tuples which will finally be removed from the result. The left hand side expression can be evaluated more efficiently.

b. $\sigma_\theta (_{A \bar{G}_F} E) = _A \bar{G}_F (\sigma_\theta (E))$, where $\theta$ uses only attributes from $A$.

$\theta$ uses only attributes from $A$. Therefore if any tuple $t$ in the output of $_{A \bar{G}_F} E$ is filtered out by the selection of the left hand side, all the tuples in $E$ whose value in $A$ is equal to $t[A]$ are filtered out by the selection of the right hand side. Therefore:

$$\forall t, t \notin \sigma_\theta (_{A \bar{G}_F} E) \implies t \notin _A \bar{G}_F (\sigma_\theta (E))$$

Using similar reasoning, we can also conclude that

$$\forall t, t \notin _A \bar{G}_F (\sigma_\theta (E)) \implies t \notin \sigma_\theta (_{A \bar{G}_F} E)$$

The above two equations imply the given equivalence.
This equivalence is helpful because evaluation of the right hand side avoids performing the aggregation on groups which are anyway going to be removed from the result. Thus the right hand side expression can be evaluated more efficiently than the left hand side expression.

c. \( \sigma_\theta(E_1 \bowtie E_2) = \sigma_\theta(E_1) \bowtie E_2 \) where \( \theta \) uses only attributes from \( E_1 \).

\( \theta \) uses only attributes from \( E_1 \). Therefore if any tuple \( t \) in the output of \( (E_1 \bowtie E_2) \) is filtered out by the selection of the left hand side, all the tuples in \( E_1 \) whose value is equal to \( t[E_1] \) are filtered out by the selection of the right hand side. Therefore:

\[ \forall t, t \notin \sigma_\theta(E_1 \bowtie E_2) \Rightarrow t \notin \sigma_\theta(E_1) \bowtie E_2 \]

Using similar reasoning, we can also conclude that

\[ \forall t, t \notin \sigma_\theta(E_1) \bowtie E_2 \Rightarrow t \notin \sigma_\theta(E_1 \bowtie E_2) \]

The above two equations imply the given equivalence.

This equivalence is helpful because evaluation of the right hand side avoids producing many output tuples which are anyway going to be removed from the result. Thus the right hand side expression can be evaluated more efficiently than the left hand side expression.

14.2  

a. \( R = \{(1,2)\}, S = \{(1,3)\} \)

The result of the left hand side expression is \( \{(1)\} \), whereas the result of the right hand side expression is empty.

b. \( R = \{(1,2),(1,5)\} \)

The left hand side expression has an empty result, whereas the right hand side one has the result \( \{(1,2)\} \).

c. Yes, on replacing the \textit{max} by the \textit{min}, the expressions will become equivalent. Any tuple that the selection in the rhs eliminates would not pass the selection on the lhs if it were the minimum value, and would be eliminated anyway if it were not the minimum value.

d. \( R = \{(1,2)\}, S = \{(2,3)\}, T = \{(1,4)\} \)

The left hand side expression gives \( \{(1,2,\text{null},4)\} \) whereas the the right hand side expression gives \( \{(1,2,3,\text{null})\} \).

e. Let \( R \) be of the schema \((A,B)\) and \( S \) of \((A,C)\). Let \( R = \{(1,2)\}, S = \{(2,3)\} \) and let \( \theta \) be the expression \( C = 1 \). The left side expression’s result is empty, whereas the right side expression results in \( \{(1,2,\text{null})\} \).

14.3  

a. We define the multiset versions of the relational-algebra operators here. Given multiset relations \( r_1 \) and \( r_2 \),

i. \( \sigma \)

Let there be \( c_1 \) copies of tuple \( t_1 \) in \( r_1 \). If \( t_1 \) satisfies the selection \( \sigma_\theta \), then there are \( c_1 \) copies of \( t_1 \) in \( \sigma_\theta(r_1) \), otherwise there are none.

ii. \( \Pi \)

For each copy of tuple \( t_1 \) in \( r_1 \), there is a copy of tuple \( \Pi_A(t_1) \) in \( \Pi_A(r_1) \), where \( \Pi_A(t_1) \) denotes the projection of the single tuple \( t_1 \).

iii. \( \times \)

If there are \( c_1 \) copies of tuple \( t_1 \) in \( r_1 \) and \( c_2 \) copies of tuple \( t_2 \) in \( r_2 \), then there are \( c_1 \times c_2 \) copies of the tuple \( t_1,t_2 \) in \( r_1 \times r_2 \).
iv. \( \times \)

The output will be the same as a cross product followed by a selection.

v. \( - \)

If there are \( c_1 \) copies of tuple \( t \) in \( r_1 \) and \( c_2 \) copies of \( t \) in \( r_2 \), then there will be \( c_1 - c_2 \) copies of \( t \) in \( r_1 - r_2 \), provided that \( c_1 - c_2 \) is positive.

vi. \( \cup \)

If there are \( c_1 \) copies of tuple \( t \) in \( r_1 \) and \( c_2 \) copies of \( t \) in \( r_2 \), then there will be \( c_1 + c_2 \) copies of \( t \) in \( r_1 \cup r_2 \).

vii. \( \cap \)

If there are \( c_1 \) copies of tuple \( t \) in \( r_1 \) and \( c_2 \) copies of \( t \) in \( r_2 \), then there will be \( \min(c_1, c_2) \) copies of \( t \) in \( r_1 \cap r_2 \).

b. All the equivalence rules 1 through 7.b of section 14.2.1 hold for the multiset version of the relational-algebra defined in the first part.

There exist equivalence rules which hold for the ordinary relational-algebra, but do not hold for the multiset version. For example consider the rule:

\[
A \cap B = A \cup B - (A - B) - (B - A)
\]

This is clearly valid in plain relational-algebra. Consider a multiset instance in which a tuple \( t \) occurs 4 times in \( A \) and 3 times in \( B \). \( t \) will occur 3 times in the output of the left hand side expression, but 6 times in the output of the right hand side expression. The reason for this rule to not hold in the multiset version is the asymmetry in the semantics of multiset union and intersection.

14.4 The relation resulting from the join of \( r_1, r_2, \) and \( r_3 \) will be the same no matter which way we join them, due to the associative and commutative properties of joins. So we will consider the size based on the strategy of \(((r_1 \bowtie r_2) \bowtie r_3)\). Joining \( r_1 \) with \( r_2 \) will yield a relation of at most 1000 tuples, since \( C \) is a key for \( r_2 \). Likewise, joining that result with \( r_3 \) will yield a relation of at most 1000 tuples because \( E \) is a key for \( r_3 \). Therefore the final relation will have at most 1000 tuples.

An efficient strategy for computing this join would be to create an index on attribute \( C \) for relation \( r_2 \) and on \( E \) for \( r_3 \). Then for each tuple in \( r_1 \), we do the following:

a. Use the index for \( r_2 \) to look up at most one tuple which matches the \( C \) value of \( r_1 \).

b. Use the created index on \( E \) to look up in \( r_3 \) at most one tuple which matches the unique value for \( E \) in \( r_2 \).

14.5 The estimated size of the relation can be determined by calculating the average number of tuples which would be joined with each tuple of the second relation. In this case, for each tuple in \( r_3 \), \( \frac{1500}{V(C, r_2) = 15/11} \) tuples (on the average) of \( r_2 \) would join with it. The intermediate relation would have \( 15000/11 \) tuples. This relation is joined with \( r_3 \) to yield a result of approximately 10,227 tuples \((15000/11 \times 750/100 = 10227)\). A good strategy should join \( r_1 \) and \( r_2 \) first, since
the intermediate relation is about the same size as \( r_1 \) or \( r_2 \). Then \( r_3 \) is joined to this result.

14.6 a. Use the index to locate the first tuple whose \texttt{branch\_city} field has value “Brooklyn”. From this tuple, follow the pointer chains till the end, retrieving all the tuples.

b. For this query, the index serves no purpose. We can scan the file sequentially and select all tuples whose \texttt{branch\_city} field is anything other than “Brooklyn”.

c. This query is equivalent to the query:

\[
\sigma_{\texttt{branch\_city} \geq 'Brooklyn' \land \texttt{assets} < 5000} (\texttt{branch}).
\]

Using the \texttt{branch\_city} index, we can retrieve all tuples with \texttt{branch\_city} value greater than or equal to “Brooklyn” by following the pointer chains from the first “Brooklyn” tuple. We also apply the additional criteria of \texttt{assets} < 5000 on every tuple.

14.7 Each join order is a complete binary tree (every non-leaf node has exactly two children) with the relations as the leaves. The number of different complete binary trees with \( n \) leaf nodes is \( \frac{1}{n} \binom{2(n-1)}{n-1} \). This is because there is a bijection between the number of complete binary trees with \( n \) leaves and number of binary trees with \( n-1 \) nodes. Any complete binary tree with \( n \) leaves has \( n-1 \) internal nodes. Removing all the leaf nodes, we get a binary tree with \( n-1 \) nodes. Conversely, given any binary tree with \( n-1 \) nodes, it can be converted to a complete binary tree by adding \( n \) leaves in a unique way. The number of binary trees with \( n-1 \) nodes is given by \( \frac{1}{n} \binom{2(n-1)}{n-1} \), known as the Catalan number. Multiplying this by \( n! \) for the number of permutations of the \( n \) leaves, we get the desired result.

14.8 Consider the dynamic programming algorithm given in Section 14.4.2. For each subset having \( k+1 \) relations, the optimal join order can be computed in time \( 2^{k+1} \). That is because for one particular pair of subsets \( A \) and \( B \), we need constant time and there are at most \( 2^{k+1} - 2 \) different subsets that \( A \) can denote. Thus, over all the \( \binom{n}{k+1} \) subsets of size \( k+1 \), this cost is \( \binom{n}{k+1} 2^{k+1} \). Summing over all \( k \) from 1 to \( n-1 \) gives the binomial expansion of \( (1 + x)^n - x \) with \( x = 2 \). Thus the total cost is less than \( 3^n \).

14.9 The derivation of time taken is similar to the general case, except that instead of considering \( 2^{k+1} - 2 \) subsets of size less than or equal to \( k \) for \( A \), we only need to consider \( k+1 \) subsets of size exactly equal to \( k \). That is because the right hand operand of the topmost join has to be a single relation. Therefore the total cost for finding the best join order for all subsets of size \( k+1 \) is \( \binom{n}{k} (k+1) \), which is equal to \( n \binom{n-1}{k} \). Summing over all \( k \) from 1 to \( n-1 \) using the binomial expansion of \( (1 + x)^{n-1} \) with \( x = 1 \), gives a total cost of less than \( n2^{n-1} \).

14.10 a. The nested query is as follows:
Exercises

select  
from account S
where S.branch_name like 'B%' and
S.balance =
(select max(T.balance)
from account T
where T.branch_name = S.branch_name)

b. The decorrelated query is as follows:

c. In general, consider the queries of the form:

where, \( f \) is some aggregate function on attributes \( A_2 \), and \( op \) is some boolean binary operator. It can be rewritten as

where \( P_2^1 \) contains predicates in \( P_2 \) without selections involving correlation variables, and \( P_2^2 \) introduces the selections involving the correlation variables. \( V \) contains all the attributes that are used in the selections involving correlation variables in the nested query.