For the above schema (the primary key for each relation is denoted by the underlined attribute), provide relational algebra expressions for the following queries:

**Note:** For notational convenience, I am using `pname` instead of `person-name`, `cname` instead of `company-name`, and `mname` instead of `manager-name`. In addition, instead of `lives[pname]` I am using the equivalent notation `lives,pname` in an SQL-like syntax in some queries – you can use either.

1. Find the tuples of employees (i.e., persons) who work for the City Bank company (which is a specific company in the database).
   
   `{w|w ∈ works ∧ (w[cname] = 'City Bank')} 

2. Find the name and city of all employees who work for City Bank. Similar to previous query, except we have to access the lives table to extract the city of the employee. Note the join condition in the query. The ‘type’ of `p` has two attributes - name and city- as referenced in the right hand side of the query.

   `{p|∃w ∈ works, ∃l ∈ lives 
    ((w[cname] = 'City Bank') ∧ (w[pname] = l[pname])) 
    ∧(l[pname] = p[pname] ∧ (w[city] = p[city])))}

3. Find all employees (names) who live in the same city and on the same street as their manager. This requires accessing lives table twice – once for finding city of employee and a second time for finding city of manager. Therefore we need two variables from lives with different names; one will refer to employee and the other will refer to the manager. Since `x` is only defined for `pname` on the right hand side of the query, it has a single attribute `pname`.

   `{x|∃y, z ∈ lives ∃m ∈ manages 
    ((z.city = y.city) ∧ (y.street = z.street) ∧ 
     (y.pname = m.pname) ∧ (z.pname = m.managername) ∧ (x.pname = y.pname))}

**SQL queries:**
4. Find all employees who earn more than every employee City Bank.

   Need to use nested queries. Find set of salaries of employees of City Bank. Next in outer
   query, compare salary of employee with every member of the set of salaries of City Bank (i.e.,
   output of nested query).

   \[
   \text{SELECT w1.pname} \\
   \text{FROM works w1} \\
   \text{WHERE w1.salary > ALL (} \\
   \text{  \hspace{1cm} SELECT w2.salary} \\
   \text{  \hspace{1cm} FROM works w2} \\
   \text{  \hspace{1cm} WHERE w2.cname = 'City Bank'} \hspace{1cm} ) ;
   \]

   An alternate solution is to simply find max salary in the nested query and check if w1.salary
   is greater than the max.

5. Find the company with the most employees.

   This query requires use of GROUPBY operator and nested queries. In subquery, for each
   company find the number of employees. Next, in the outer query, for each company check if
   the number of employees in the company is greater than or equal to all values output by the
   nested query (which is the number of employees at each company). For this last condition we
   need to use the HAVING clause since we are checking the condition for a group.

   \[
   \text{SELECT w1.cname} \\
   \text{FROM works w1} \\
   \text{GROUPBY w1.cname} \\
   \text{having count(w1.person-name) >= ALL (} \\
   \text{  \hspace{1cm} SELECT COUNT(w2.person-name)} \\
   \text{  \hspace{1cm} FROM works w2} \\
   \text{  \hspace{1cm} groupby w2.cname);} \\
   \]

6. Find the companies (company name) that pay more, on average, than the average salary
   paid by City Bank.

   We need aggregate operations, groupby, and nested queries.

   \[
   \text{SELECT w1.cname} \\
   \text{FROM works w1} \\
   \text{GROUPBY w1.cname} \\
   \text{HAVING AVG(w1.salary) > (} \\
   \text{  \hspace{1cm} SELECT AVG(w2.salary)} \\
   \text{  \hspace{1cm} FROM works w2} \\
   \text{  \hspace{1cm} WHERE w2.cname= 'City Bank'} ) ;
   \]