BUSA8090 - Assignment 1, Task 3

Samuel Gardiner (44952619)

16 April 2020

Note to the marker:

Each SQL answer is available as a `.sql` file which contains only the SQL query, and as a `.sh` file which (when executed) will print the query to the terminal, connect to the `compbiol` database on your AWS instance, and then execute the query.

All of the answer SQL queries and `bash` scripts are contained within a GitHub repository at `https://github.com/gardiners/A1T3`

To download all of the files and get them ready to mark, run the following on your Ubuntu AWS instance:

```
source <(curl -s https://raw.githubusercontent.com/gardiners/A1T3/master/install.sh)
```

Which will clone the GitHub repository into a new directory called `gardiners-A1T3`, make all of the scripts executable, and then put you into that directory ready to mark.

Question 1

a.

Since we already have the `expression` table in our MySQL `compbiol` database, we can export it to a file. From `man mysql` (at line 60 of the man page):

```
   ·   --batch, -B

           Print results using tab as the column separator, with each row on a
           ↪  new line. With this option, mysql does not use the history
           ↪  file.
```

So, with the `--batch` argument and the file redirect operator `>`, we can write a tab-separated file from the command-line:

```
# Create a directory to store the TSV file:
mkdir -p ~/busa/A1T3
# Write the TSV file:
mysql --batch -u awkologist -p compbiol -e \
"SELECT * FROM expression \
ORDER BY gene;" > ~/busa/A1T3/expression.tsv
```

We have ordered the output by `gene` to make life easier when using the `join` command from `bash` in part **c.**, below.

The tab-separated file is available at `https://raw.githubusercontent.com/gardiners/A1T3/master/expression.tsv` for marking.

b.

Using the same logic as in **a.**, above, we query for every row in the `annotation` table and then save it to a file `annotation.tsv` using the `--batch` argument to `mysql`.

```
mysql --batch -u awkologist -p compbiol -e \
"SELECT * FROM annotation \
ORDER BY gene;" > ~/busa/A1T3/annotation.tsv
```

Again, we have ordered by `gene` to make part **c.**, below, easier. The tab-separated file is available at `https://raw.githubusercontent.com/gardiners/A1T3/master/annotation.tsv` for marking.

c.

Since `join` requires its input files to be sorted on the key field, both TSV files exported above were sorted on `gene` at the time of export, ensuring that the list of genes is in the same order in both files. We could also use `sort` inside a subshell for each file when calling `join`, but that would make the `join` call harder to interpret (and harder to mark).

So, assuming we are in a directory with our files named `annotation.tsv` and `expression.tsv`, we can implement the given relational algebra statement with the following `bash` command:

```
join -t $'\t' --header annotation.tsv expression.tsv | cut -f 1,2,4 > join.tsv
```

Examining each element of the line, we have:

- the `join` command. Since the default is to join on the first field in each input, and our files already have the key field `gene` as the first field, we do not need to explicity specify the key.
  - the `-t $'\t'` argument, which specifies which character should be used to separate values in the input files. `$'\t'` indicates that a literal tab character should be used as the separator (the `$'...'` construct is ANSI C quoting of literals). By default `join` uses a space *or* tab character as its delimiter, but returns space-delimited output. Since we want to pass the output to `cut`, we force `join` to use tab as its input and output delimiter.
  - `--header` indicates that the first line of the input files should not be matched upon, and should simply be combined and returned as the first line of the output.
  - `annotation.tsv expression.tsv` specifies the input files.
- the pipe `|` to send the output of `join` to the input of `cut`
- the `cut` command, to select the fields that we wish to return. By default, `cut` uses the tab character as its field delimeter.

  - the `-f 1,2,4` argument to return the fields `gene`, `function` and `expr_value` (the first, second and fourth fields in the joined dataset).
- the file redirect operator `>` to save the output to the file `join.tsv`

The answer command has been saved with the filename `join.sh` in the GitHub repository for A1T3.

- The command for marking is at `https://raw.githubusercontent.com/gardiners/A1T3/master/join.sh`
- Example output is at `https://raw.githubusercontent.com/gardiners/A1T3/master/join.tsv`

d.

(i)

To find the duplicated metabolisms in the `annotation` table, our algebraic operations are

- find the inner join of `annotation` with itself, with the joining conditions
  - both 'copies' of `metabolism` in the joint tuple are the same, AND
  - each 'copy' of `gene` in in the joint tuple are different
- project the resulting list of metabolisms.

In relational algebra, our statement is therefore:

$$\pi_{\textbf{metabolism}} (\textbf{annotation} \bowtie_{\textbf{metabolism}=\textbf{metabolism} \text{ AND } \textbf{gene}\neq\textbf{gene}} \textbf{annotation})$$

(ii)

Since the relational algebra statement above is guaranteed to return distinct tuples (by its axioms), we use `SELECT DISTINCT` in our SQL query to ensure that we obtain only distinct relations as its result. That is, we wish to know only the names of the duplicated metabolisms; we don't want a repeated row for each time a metabolism is duplicated.

Therefore, our SQL query is:

```
SELECT DISTINCT a1.metabolism
FROM annotation a1
        INNER JOIN annotation a2
        ON (a1.metabolism = a2.metabolism
                AND a1.gene <> a2.gene);
```

which returns the result

```
+-------------------------+
| metabolism              |
+-------------------------+
| Pyrimidine biosynthesis |
+-------------------------+
```

- an SQL command for marking is available at `https://raw.githubusercontent.com/gardiners/A1T3/master/q1dii.sql`
- an executable `bash` script that runs the SQL command at the terminal is available at `https://raw.githubusercontent.com/gardiners/A1T3/master/q1dii.sh`

Question 2

a.

(i)

Assuming our dialect of relational algebra has a definition for NULL,

$$\pi_{\textbf{LastName},\textbf{FirstName}}\left(\sigma_{\textbf{Coach}\ \text{IS NULL}}\left(\textbf{Member}\right)\right)$$

(ii)

Assuming our dialect of the relational calculus has a definition for NULL,

$$\{\ \texttt{m.LastName, m.FirstName} \mid \texttt{Member(m) and m.Coach IS NULL}\ \}$$

(iii)

Our query is

```
SELECT LastName, Firstname
  FROM Member
 WHERE Coach IS NULL;
```

which returns

```
+----------+-----------+
| LastName | FirstName |
+----------+-----------+
| Stone    | Michael   |
| Nolan    | Brenda    |
| Branch   | Helen     |
| Beck     | Sarah     |
| Spence   | Thomas    |
| Olson    | Barbara   |
| Wilcox   | Daniel    |
| Young    | Betty     |
| Willis   | Carolyn   |
| Kent     | Susan     |
+----------+-----------+
```

- The SQL query for marking is available at `https://raw.githubusercontent.com/gardiners/A1T3/master/q2a.sql`

- An executable `bash` script that will run the query against the `compbiol` database is available at `https://raw.githubusercontent.com/gardiners/A1T3/master/q2a.sh`

b.

(i)

We select from Member all of the players who meet the condition that their join date is between the two dates specified, and then project the atributes LastName and FirstName:

$$\pi_{\textbf{LastName},\textbf{FirstName}}\left(\sigma_{\textbf{JoinDate}\geq\text{`2010-01-01' AND }\textbf{JoinDate}\leq\text{`2010-12-31'}}\left(\textbf{Member}\right)\right)$$

(ii)

```
{ m.LastName, m.FirstName | Member(m)
                            and m.JoinDate ≥ '2010-01-01'
                            and m.JoinDate ≤ '2010-12-31' }
```

(iii)

We can use `BETWEEN` as a syntactic sweetener instead of a pair of comparisons. A query that implements the above expressions (when executed against `compbiol`) is

```
SELECT  LastName, FirstName
  FROM  Member
 WHERE  JoinDate BETWEEN '2010-01-01' AND '2010-12-31';
```

which returns:

```
+----------+-----------+
| LastName | Firstname |
+----------+-----------+
| Beck     | Sarah     |
| Kent     | Susan     |
+----------+-----------+
```

- The SQL for marking is at `https://raw.githubusercontent.com/gardiners/A1T3/master/q2b.sql`
- An executable `bash` script that runs the SQL against `compbiol` is available at `https://raw.githubusercontent.com/gardiners/A1T3/master/q2b.sh`

c.

(i)

In the algebraic paradigm, we perform two subequeries to assumble the filter for the select operator:

$$\textbf{m1} \leftarrow \pi_{\textbf{MemberID}}(\textbf{Entry})$$
$$\textbf{m2} \leftarrow \pi_{\textbf{MemberID}}\left(\sigma_{\textbf{Year}=2014}\left(\textbf{Entry}\right)\right)$$

The first of the subqueries (here **m1**) simply lists all of the MemberIDs that have competed in tournaments. The second subquery (**m2**) lists all of the MemberIDs who competed in 2014. We can use these subqueries together in our main algebraic statement to perform a set difference:

$$\pi_{\mathbf{MemberID,LastName,FirstName}}\left(\sigma_{\mathbf{MemberID\ \texttt{IN}\ a1\ \texttt{AND}\ MemberID\ \texttt{NOT IN}\ a2}}\left(\mathbf{Member}\right)\right)$$

(ii)

$$\{\ \texttt{m.MemberID, m.LastName, m.FirstName} \mid \texttt{Member(m), Entry(e)}$$
$$\texttt{and m.MemberID} = \texttt{e.MemberID}$$
$$\texttt{and NOT } \exists\texttt{(n) Entry(n) and n.Year} = 2014\ \}$$

(iii)

A query which implements the expressions above (in the relational calculus style) is

```
SELECT DISTINCT m.MemberID, m.LastName, m.FirstName
           FROM Member m
                INNER JOIN Entry e
                        ON m.MemberID = e.MemberID
          WHERE NOT EXISTS (SELECT * FROM Entry e
                              WHERE m.MemberID = e.MemberID
                                AND e.Year = 2014);
```

which yields the result:

```
+----------+----------+-----------+
| MemberID | LastName | FirstName |
+----------+----------+-----------+
|      228 | Burton   | Sandra    |
|      239 | Spence   | Thomas    |
+----------+----------+-----------+
```

- The SQL answer for marking is at `https://raw.githubusercontent.com/gardiners/A1T3/master/q2c.sql`
- A `bash` script which executes the SQL against `compbiol` is available at `https://raw.githubusercontent.com/gardiners/A1T3/master/q2b.sh`

d.

(i)

$$\{\ \texttt{m.MemberID, m.LastName, m.FirstName} \mid \texttt{Member(m) AND}$$
$$\forall\texttt{(y) Entry (y)}$$
$$(\exists\texttt{(e) Entry(e) AND y.Year = e.Year AND}$$
$$\texttt{m.MemberID = e.MemberID)}\}$$

We can also express this as a double-negative to aid translation to SQL: we want every member where there does not exist a year in which there does not exist an entry for that player.

(ii)

We can express this without the double-negative in SQL if we take the algebraic approach. We use `COUNT(DISTINCT Years)` for our member-entries and compare this to the total number of distinct years in which members have competed using the `HAVING` clause and a subquery:

```
  SELECT m.MemberId, m.LastName, m.FirstName
    FROM Member m
         INNER JOIN Entry e
         ON m.MemberID = e.MemberID
GROUP BY m.MemberID, LastName, FirstName
  HAVING COUNT(DISTINCT e.Year) = (SELECT COUNT(DISTINCT e.Year) FROM Entry e);
```

which returns only the players who played in 2013, 2014 and 2015:

```
+----------+----------+-----------+
| MemberId | LastName | FirstName |
+----------+----------+-----------+
|      235 | Cooper   | William   |
|      286 | Pollard  | Robert    |
|      415 | Taylor   | William   |
+----------+----------+-----------+
```

- The SQL for marking is at `https://raw.githubusercontent.com/gardiners/A1T3/master/q2d.sql`
- An executable `bash` script which prints the query text, runs the query against the `compbiol` database and then returns the results is available at `https://raw.githubusercontent.com/gardiners/A1T3/master/q2d.sh`