The Australian National University, School of Computing
COMP2400/6240 (Relational Databases)
Semester 2, 2022

Lab 3

Advanced SQL (solutions)

The purpose of this lab is to extend your knowledge of SQL. SQL is powerful and the results that can be expressed as a single query is often quite impressive. Using a single query is a good goal, as a single query can be optimized by the database system, whereas alternative solutions that involve multiple queries often require expensive storage or transmission of data over the network.

In the following exercises, try to find a single query that returns the desired result. All queries are run against our example database (created by running employeeCreate.sql). The relation schemas of our example database are described below,

- DEPARTMENT(dname, dnumber, mgrssn, mgrstartdate)
  with the primary key {dnumber};
- EMPLOYEE(fname, minit, lname, ssn, bdate, address, salary, superssn, dno)
  with the primary key {ssn};
- PROJECT(pname, pnumber, plocation, dnum)
  with the primary key {pnumber};
- WORKS_ON(ssn, pno, hours)
  with the primary key {ssn, pno};
- DEPT_LOCATION(dnumber, dlocation)
  with the primary key {dnumber, dlocation};

where the foreign keys are:

EMPLOYEE[dno]$\subseteq$DEPARTMENT[dnumber]
PROJECT[dnum]$\subseteq$DEPARTMENT[dnumber]
WORKS_ON[ssn]$\subseteq$EMPLOYEE[ssn]
WORKS_ON[pno]$\subseteq$PROJECT[pnumber]
DEPT_LOCATION[dnumber]$\subseteq$DEPARTMENT[dnumber]

The above example database state is shown in the following figure

EMPLOYEE

fname	minit	lname	ssn	bdate	address	salary	superssn	dno
Michio		Morishima	20118	1973-07-18	79 Macpherson St, Turner	52107.00	21286	1000
John		Backus	20766	1984-12-03	25 Burns St, Yarralumla	46789.00	21287	1007
Gramsci		Antonio	20876	1991-01-22	27 Garibaldi St, Ashfield, NSW	71569.00	20915	1001
Ada		Lovelace	21286	1985-12-10	17 Ainslie Ave, Reid, ACT	62107.00	21286	1000
Milton		Friedman	29057	1972-07-31	75 Wakefield Ave, Ainslie	37764.00	21287	1007
Edsger	W	Dijkstra	20765	1980-05-11	192 Wattle St, O'Connor ACT	73567.00	20766	1000
Grace	M	Hopper	20864	1976-12-09	45 Cobol St, Parramatta, NSW	78563.00	21286	1000
Frederick	W	Taylor	20915	1986-03-20	14 Blackett St, Downer, ACT	56098.00	20915	1001
John	M	Keynes	21287	1983-06-05	94 Earle St, Lyneham, ACT	73567.00	21287	1007

(9 rows)

DEPARTMENT

dname	dnumber	mgrssn	mgrstartdate
Information Technology	1000	20765	2007-01-01
Administration	1001	20915	2004-02-29
Finance	1007	21287	2005-06-07

(3 rows)

DEPT_LOCATION

dnumber	dlocation
1000	Canberra
1000	Sydney
1001	Canberra
1007	Canberra
1007	Sydney

(5 rows)

PROJECT

pname	pnumber	plocation	dnum
Difference Engine	9000	Canberra	1000
Red tape is Fun	9001	Canberra	1001
Object Oriented COBOL	9002	Sydney	1000

(3 rows)

WORKS_ON

ssn	pno	hours
20765	9000	100
20765	9001	500
20864	9002	50
20915	9000	250

(4 rows)

1 Warm-Up Exercises

(1) *Find the downloaded file* `employeeCreate.sql` *(e.g., on* `Desktop`*) and start* `psql` *from* `shell` *and then type*
(Ignore this step for Option 1 as the `empplyee` *database has been already loaded to PostgreSQL in docker)*

```
\i ~/Desktop/employeeCreate.sql
```

Now we have created our example database for the following questions and queries.

(2) *The result of the following query contains duplicate records. Look up the* `DISTINCT` *in the PostgreSQL manual (or in the lecture slides), and use it to improve your query.*

```
SELECT supervisor.lname, supervisor.ssn
FROM employee, employee AS supervisor
```

```
WHERE employee.superssn = supervisor.ssn;
```

Solution:

- ```
  SELECT DISTINCT supervisor.lname, supervisor.ssn
  FROM employee, employee AS supervisor
  WHERE employee.superssn = supervisor.ssn;
  ```

(3) *Now look up* `ORDER BY`*, and use it to modify the following query and order the result in the ascending order of their last names.*

```
SELECT lname, ssn
FROM employee;
```

Solution:

- ```
  SELECT lname, ssn
  FROM employee
  ORDER BY lname ASC;
  ```

2 Aggregation - Grouping and Having

In SQL, the `GROUP BY` clause is used in conjunction with aggregate functions to group a table based on one or more columns. If the columns $A_1, \ldots, A_n$ are listed the `GROUP BY` clause of a query, then the database returns one group per unique value of the combination of these columns and aggregate functions can be used in the query to each of these groups. For example,

```
SELECT dno, SUM(salary)
FROM employee
GROUP BY dno;
```

will form a group of employees for each unique department number. Then the SUM function is applied to the salaries of employees in each group. You can refer to the PostgreSQL manual for all possible aggregate functions provided by PostgreSQL.

The `HAVING` clause can also be used to apply conditions on a grouping operation in a query. For example,

```
SELECT dno, COUNT(*)
FROM employee
GROUP BY dno HAVING COUNT(*)>2;
```

returns the department number and the number of employees of those departments which have more than 2 employees.

Use aggregation functions to write down the following SQL queries.

(4) *Write a single query which shows the average salary of employees for each department.*

Solution:

- ```
  SELECT dno, AVG(salary)
  FROM employee
  GROUP BY dno;
  ```

(5) *Show the project numbers and total hours for the projects if their total hours are larger than 200 hours.*

Solution:

- ```
  SELECT pno, SUM(hours)
  FROM works_on
  GROUP BY pno
  HAVING SUM(hours)>200;
  ```

(6) *Show the project numbers, names and total hours for the projects if their total hours are larger than 200 hours. Compare your query with the query written in the previous exercise.*

Solution:

- ```
  SELECT pno, pname, SUM(hours)
  FROM works_on, project
  WHERE works_on.pno = project.pnumber
  GROUP BY pno, pname
  HAVING SUM(hours)>200;
  ```

(7) *Show the department number, department name and average salary of all employees who work in the department if the average salary is greater than $60,000*
  ```

Solution:

- ```
  SELECT dnumber, dname, AVG(salary)
  FROM employee, department
  WHERE employee.dno = department.dnumber
  GROUP BY dnumber, dname
  HAVING AVG(salary)>60000;
  ```

3 Inner Join and Outer Join

In SQL, the most frequently used join is `INNER JOIN`, which produces only the set of records that match in both tables. Write down the following SQL queries using `INNER JOIN`.

(8) *List the employees who work on at least one project.*

Solution:

- ```
  SELECT DISTINCT employee.*
  FROM employee INNER JOIN works_on ON employee.ssn=works_on.ssn;
  ```

(9) *List the projects which at least one employee works on.*

Solution:

- ```
  SELECT DISTINCT project.*
  FROM project INNER JOIN works_on ON project.pnumber=works_on.pno;
  ```

In SQL, `OUTER JOIN` includes `LEFT JOIN` and `RIGHT JOIN`. `LEFT JOIN` produces a complete set of records from the "left" table, with the matching records (where available) in the "right" table. If there is no match, the records from the "right" table will contain NULL. `RIGHT JOIN` produces a complete set of records from the "right" table, with the matching records (where available) in the "left" table. If there is no match, the records from the "left" table will contain NULL. Write down the following SQL queries using OUTER JOIN.

(10) *List all the employees, and the project numbers of the projects they work on if any.*

Solution:

- ```
  SELECT employee.*, works_on.pno
  FROM employee LEFT JOIN works_on ON employee.ssn=works_on.ssn;
  ```

4 Subqueries

Part of the power of SQL comes from the ability to compose queries using subqueries.

4.1 Table subqueries

SQL allows a subquery to be used in the `FROM` clause where a table would be allowed. For example,

```
SELECT MAX(dept_salary)
FROM (SELECT dnumber, SUM(salary) AS dept_salary
      FROM department, employee
      WHERE dno = dnumber
      GROUP BY dnumber) AS by_dept;
```

In a table subquery, the table alias (e.g., `AS by_dept`) is mandatory in PostgreSQL.

Write down the following SQL queries using *table subqueries*.

(11) *How many hours have been spent working on the most time-consuming project?*

Solution:

- ```
  SELECT MAX(project_hours)
  FROM (SELECT SUM(hours) AS project_hours
        FROM works_on
        GROUP BY pno) AS h;
  ```

(12) *Find the highest paid employee of each department, and show their first and last names, department numbers and salaries.*

Solution:
  ```

- ```
  SELECT fname, lname, h.dno, max_salary
  FROM employee, (SELECT dno, MAX(salary) AS max_salary
                  FROM employee
                  GROUP BY dno) AS h
  WHERE employee.salary = h.max_salary
  AND employee.dno = h.dno;
  ```

(13) *List the first and last names of employees who work in departments with more than one location.*

Solution:

- ```
  SELECT fname, lname
  FROM employee AS e, (SELECT dnumber, COUNT(*) AS loc_count
                       FROM dept_location
                       GROUP BY dnumber) AS a
                       WHERE e.dno = a.dnumber
                       AND a.loc_count >1;
  ```

4.2 Correlated subqueries

A correlated subquery involves an SQL expression that is (logically at least) executed once per row of the outer query, and use a value from the row of the outer query to calculate a result. Correlated subqueries always appear in the `WHERE` clause of an SQL query. For example, to list the first and last names of employees who work for departments which have a Canberra office:

- ```
  SELECT fname, lname
  FROM employee AS e
  WHERE EXISTS (SELECT * FROM dept_location AS l
                WHERE l.dnumber = e.dno
                AND l.dlocation = 'Canberra');
  ```

Note that the `WHERE` clause in the inner query compares the `dnumber` from `dept_location` (in the inner query) with the `dno` from `employee` (in the outer query). The inner query is evaluated once for each row in the outer query.

Write down the following SQL queries using *correlated subqueries*.
  ```

(14) *List the names of all departments that have at least one employee whose salary is less than 50000.*

Solution:

- ```
  SELECT dname
  FROM department AS d
  WHERE EXISTS (SELECT * FROM employee AS e
                WHERE d.dnumber = e.dno AND e.salary < 50000);
  ```

(15) *List the first and last names of employees who work in departments with more than one location.*

Solution:

- ```
  SELECT fname, lname
  FROM employee AS e
  WHERE (SELECT COUNT(*) FROM dept_location AS l
         WHERE l.dnumber = e.dno) >1;
  ```

(16) *List the first and last names of employees who have a higher salary than their supervisor.*

Solution:

- ```
  SELECT e.fname, e.lname
  FROM employee AS e
  WHERE salary>(SELECT s.salary FROM employee AS s
                WHERE s.ssn = e.superssn);
  ```

(17) *Which employee(s) has/have contributed the most hours to projects run by departments they do not belong to? List the first and last name(s) of the employee(s).*

Solution:
  ```

- ```
  SELECT fname, lname
  FROM employee, works_on, project
  WHERE dno != dnum
        AND works_on.ssn = employee.ssn
        AND works_on.pno = project.pnumber
  GROUP BY employee.ssn
  HAVING SUM(hours) = (SELECT MAX(total_hours)
                       FROM(SELECT SUM(hours) AS total_hours
                            FROM employee,works_on,project
                            WHERE dno != dnum
                            AND works_on.ssn = employee.ssn
                            AND works_on.pno = project.pnumber
                            GROUP BY employee.ssn) AS m);
  ```

5 SQL practice on the MovieDB database

The Assignment 1 on SQL will be questions on the MovieDB database that you should answer using SQL queries.

A copy of the MovieDB database is available on the docker (Option 1) and the CECS Ubuntu Virtual Desktop or lab machines on campus (Option 2). You should first open a command shell and then connect to the MovieDB database by entering

```
psql moviedb
```

Example question:

- Find all the movies produced in Australia. List the titles and production years of these movies.

Example answer:

- ```
  SELECT title, production_year
  FROM movie
  WHERE lower(country) = 'australia';
  ```

Try to check the query result of the above query against the MovieDB database.