The Australian National University, School of Computing
COMP2400/6240 (Relational Databases)
Semester 1, 2025

# Lab 3

# Advanced SQL

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 are 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 require storage or data transmission.

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]⊆DEPARTMENT[dnumber]
DEPARMENT[mgrssn]⊆EMPLOYEE[ssn]
PROJECT[dnum]⊆DEPARTMENT[dnumber]
WORKS_ON[ssn]⊆EMPLOYEE[ssn]
WORKS_ON[pno]⊆PROJECT[pnumber]
DEPT_LOCATION[dnumber]⊆DEPARTMENT[dnumber]

The 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)** *Create (or reset) the employee database.*

Find the file `employeeCreate.sql` that you downloaded in Lab 2 (or download it again), start `psql` in a Terminal and use the command

```
u1234567=> \i ~/path-to-file/employeeCreate.sql
```

(where "`~/path-to-file/`" is the path to the location in your home directory where you saved the file) to create the example employee database tables. Running this script again will delete ("drop") these tables if they exist, then (re-)create them and add content. So, if you have modified any of the tables and want to reset the database to the initial example state, just run the script again.

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

```
select supervisor.lname,
       supervisor.ssn
from   employee,
       employee as supervisor
where  employee.superssn = supervisor.ssn;
```

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

```
select lname,
       ssn
from   employee;
```

## 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 in 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 across other columns in 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 depart-*

*ment.*

**(5)** *Show the project numbers and total hours for all projects whose total hours are more than 200 hours.*

**(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.*

**(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.*

# 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. **`inner join`** can be abbreviated as just **`join`**. Write down the following SQL queries using **`inner join`**.

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

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

In SQL, **`outer join`** includes the three forms **`left outer join`**, **`right outer join`** and **`full outer join`**. All three can be abbreviated by omitting the word "**`outer`**". **`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`**. Likewise, **`right join`** produces a complete set of records from the "right" table, with the matching records (where available) in the "left" table and **`null`** where there is no match. **`full join`** produces a complete set of records from both tables. Write the following SQL queries using the appropriate form of **`outer join`**.

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

**(11)** *List all the projects, and the SSNs of the employees who work on the projects if any.*

# 4 Subqueries

Much 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
               join employee on dno = dnumber
        group by dnumber
       ) as by_dept;
```

In a table subquery, the table alias ( `as by_dept` in the example above) is mandatory in PostgreSQL.

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

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

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

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

## 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 uses 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*.

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

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

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

**(18)** *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).*

# 5 SQL practice on the MovieDB database

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 lab system. To connect to it, start `psql` with the name of the database as argument:

```
u1234567@n11XltYZ:~$ 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 result of the above query against the MovieDB database.