CS232L – Database Management System

LAB 05

SQL AGGREGATE FUNCTIONS, SUB-QUERIES

Prepared By: Amna Arooj FCSE Computer Engineer

Ghulam Ishaq Khan Institute of Engineering Sciences

Faculty of Computer Science & Engineering

CS232L – Database Management system Lab

Lab 5 - SQL AGGREGATE FUNCTIONS, SUB-QUERIES

Objective

The objective of this session is to learn how to use the PostgreSQL aggregate functions such as AVG(), COUNT(), MIN(), MAX(), and SUM().deeper insight of different clauses used along with going over examples on how to put them to use with select DQL command.

Instructions

- Open the handout/lab manual in front of a computer in the lab during the session.
- Practice each new command by completing the examples and exercise.
- Turn-in the answers for all the exercise problems as your lab report.
- When answering problems, indicate the commands you entered, and the output displayed.
- Try to practice and revise all the concepts covered in all previous session before coming to the lab to avoid un-necessary ambiguities.

5.1 PostgreSQL Aggregate Functions

Aggregate functions perform a calculation on a set of rows and return a single row. PostgreSQL provides all standard SQL's aggregate functions as follows:

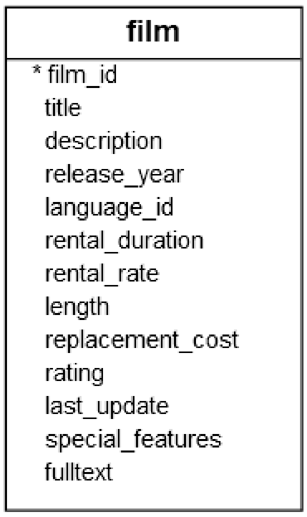
- AVG() – return the average value.
- COUNT() – return the number of values.
- MAX() – return the maximum value.
- MIN() – return the minimum value.
- SUM() – return the sum of all or distinct values.

We often use the aggregate functions with the GROUP BY clause in the SELECT statement. In these cases, the GROUP BY clause divides the result set into groups of rows and the aggregate functions perform a calculation on each group e.g., maximum, minimum, average, etc.

You can use aggregate functions as expressions only in the following clauses:

- SELECT clause.
- HAVING clause.

PostgreSQL aggregate functions examples
Let's use the film table in the sample database for the demonstration.

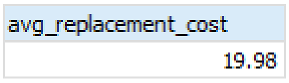


5.1.1 AVG() function examples

The following statement uses the AVG() function to calculate the average replacement cost of all films:

```
SELECT
        ROUND( AVG( replacement_cost ), 2 ) avg_replacement_cost
FROM
        film;
```

The following is the result:



```
avg_replacement_cost
               19.98
```

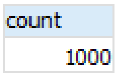
Noted that the ROUND() function was used to round the result to 2 decimal places.

5.1.2 COUNT() function examples

To get the number of films, you use the COUNT(*) function as follows:

```
SELECT
    COUNT(*)
FROM
    film;
```

Here is the output:

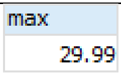


```
count
 1000
```

5.1.3 MAX() function examples

The following statement returns the maximum replacement cost of films.

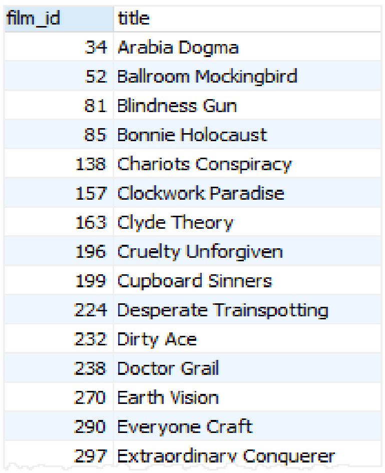
```
SELECT
    MAX(replacement_cost)
FROM
    film;
```



```
max
 29.99
```

To get the films that have the maximum replacement cost, you use the following query:

```
SELECT
        film_id,
        title
FROM
        film
WHERE
        replacement_cost =(
                SELECT
                        MAX( replacement_cost )
                FROM
                        film
        )
ORDER BY
        title;
```



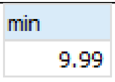
film_id	title
34	Arabia Dogma
52	Ballroom Mockingbird
81	Blindness Gun
85	Bonnie Holocaust
138	Chariots Conspiracy
157	Clockwork Paradise
163	Clyde Theory
196	Cruelty Unforgiven
199	Cupboard Sinners
224	Desperate Trainspotting
232	Dirty Ace
238	Doctor Grail
270	Earth Vision
290	Everyone Craft
297	Extraordinary Conquerer

The subquery returned the maximum replacement cost which then was used by the outer query for retrieving the film's information.

5.1.4 MIN() function examples

The following example uses the MIN() function to return the minimum replacement cost of films:

```sql
SELECT
    MIN(replacement_cost)
FROM
    film;
```

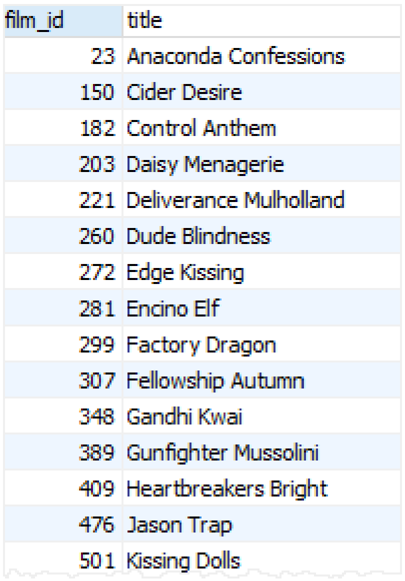


min
9.99

To get the films which have the minimum replacement cost, you use the following query:

```sql
SELECT
        film_id,
        title
FROM
        film
WHERE
        replacement_cost =(
                SELECT
                        MIN( replacement_cost )
                FROM
                        film
        )
ORDER BY
        title;
```

Code language: SQL (Structured Query Language) (sql)



film_id	title
23	Anaconda Confessions
150	Cider Desire
182	Control Anthem
203	Daisy Menagerie
221	Deliverance Mulholland
260	Dude Blindness
272	Edge Kissing
281	Encino Elf
299	Factory Dragon
307	Fellowship Autumn
348	Gandhi Kwai
389	Gunfighter Mussolini
409	Heartbreakers Bright
476	Jason Trap
501	Kissing Dolls

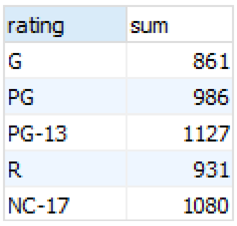
5.1.5 SUM() function examples

The following statement uses the SUM() function to calculate the total length of films grouped by film's rating:

```
SELECT
        rating,
        SUM( rental_duration )
FROM
        film
GROUP BY
        rating
ORDER BY
        rating;
```

Code language: SQL (Structured Query Language) (sql)

The following picture illustrates the result:



rating	sum
G	861
PG	986
PG-13	1127
R	931
NC-17	1080

5.1.6 PostgreSQL STRING_AGG() function

This is used to concatenate a list of strings and adds a place for a delimiter symbol or a separator between all of the strings. The output string won't have a separator or a delimiter symbol at the end of it. The PostgreSQL 9.0 version supports STRING_AGG() function. To concatenate the strings, we can employ a variety of separators or delimiter symbols.
Syntax
STRING_AGG (expression, separator|delimiter [order_by])

Expression: This character string can be any legitimate expression.
Delimiter/separator: This specifies the delimiter/separator used when concatenating strings.

The ORDER BY clause: specifies the order of the concatenated string results and is optional. Example:

```
CREATE TABLE players ( player_name TEXT , team_name TEXT , player_positon TEXT
) ;
```

With the above syntax, a table called “players” will be created, with the columns as player_name, team_name, and player_position.

***Note**: Run the following SELECT query to verify that the table is created with the desired columns. SELECT * FROM “players” ;*

Insert Values into the Table:

Let’s use the INSERT INTO command to add some values to the “players” table now:

INSERT INTO "players" VALUES ('Virat', 'India', 'Batsman'), ('Rohit', 'India', 'Batsman') , ('Jasprit', 'India', 'Bowler');

INSERT INTO "players" VALUES ('Chris', 'West Indies', 'Batsman'), ('Shannon', 'West Indies', 'Bowler'), ('Bravo', 'West Indies', 'Batsman');

INSERT INTO "players" VALUES ('James', 'New Zealand', 'All rounder');

SELECT * FROM "players" ;

```
—OUTPUT—
player_name | team_name | player_positon
-------------+-------------+----------------
 Virat | India | Batsman
 Rohit | India | Batsman
 Jasprit | India | Bowler
 Chris | West Indies | Batsman
 Shannon | West Indies | Bowler
 Bravo | West Indies | Batsman
 James | New Zealand | All rounder
(7 rows)
```

We will use the STRING_AGG() function to produce a list of values separated by commas. The syntax to create comma-separated values is as follows:

```
SELECT "team_name",string_agg("player_name", ',' )
FROM "players" GROUP BY "team_name" ;
—OUTPUT—
 team_name | string_agg
-------------+--------------------
 West Indies | Chris,Shannon,Bravo
 India | Virat,Rohit,Jasprit
 New Zealand | James
(3 rows)
```

The “player_name” column in the SELECT query is separated by commas and displayed alongside the “team_name” as seen in the output obtained. The rows are divided according to the field “team_name” using the GROUP BY command. The expression that needs to be separated is defined in the first parameter of the STRING_AGG() function, and the values are separated in the second parameter by the comma character “,”.

To obtain the output of the PostgreSQL STRING_AGG() function in an ordered manner(alphabetically), we can use the following command:

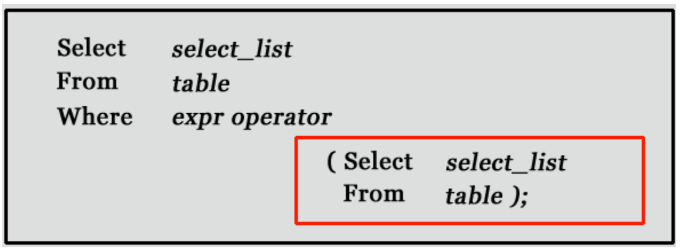
```
SELECT "team_name",
string_agg ("player_name", ',' ORDER BY "player_name" asc) AS
players_name,
string_agg ("player_positon", ',' ORDER BY "player_positon" asc) AS
players_positions
FROM "players" GROUP BY "team_name";

—OUTPUT—
team_name     | players_name         | players_positions
-------------+---------------------+-----------------------
India         | Jasprit,Rohit,Virat | Batsman,Batsman,Bowler
New Zealand   | James               | All rounder
West Indies   | Bravo,Chris,Shannon | Batsman,Batsman,Bowler
(3 rows)
```

5.2 SQL Subquery

A subquery is a SQL query nested inside a larger query.
In PostgreSQL subquery can be nested inside a SELECT, INSERT, UPDATE, DELETE, SET, or DO statement or inside another subquery. A subquery is usually added within the WHERE Clause of another SQL SELECT statement. You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, SOME, or ALL. A subquery can be treated as an inner query, which is a SQL query placed as a part of another query called as outer query. The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

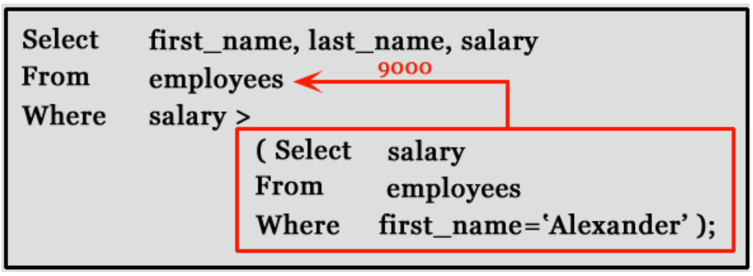
Subquery Syntax:



```
Select    select_list
From      table
Where     expr operator
                      ( Select   select_list
                        From     table );
```

The subquery (inner query) executes once before the main query (outer query) executes.
The main query (outer query) use the subquery result.

PostgreSQL Subquery Example:
Using a subquery, list the name of the employees, paid more than 'Alexander' from employees.


Select first_name, last_name, salary
From employees
Where salary >
9000
(Select salary
From employees
Where first_name='Alexander');

Code:
```
SELECT first_name,last_name, salary FROM employees
WHERE salary >
(SELECT max(salary) FROM employees
WHERE first_name='Alexander');
```

Sample Output:

first_name	last_name	salary
Steven	King	24000
Neena	Kochhar	17000
Lex	De Haan	17000
Nancy	Greenberg	12000
Den	Raphaely	11000
John	Russell	14000
Karen	Partners	13500
Michael	Hartstein	13000
Hermann	Baer	10000
Shelley	Higgins	12000

Subqueries: Guidelines
There are some guidelines to consider when using subqueries :

- A subquery must be enclosed in parentheses.
- Use single-row operators with single-row subqueries, and use multiple-row operators with multiple-row subqueries.
- If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

Types of Subqueries

1. The Subquery as Scalar Operand
2. Comparisons using Subqueries
3. Subqueries with ALL, ANY, IN, or SOME
4. Row Subqueries

5. Subqueries with EXISTS or NOT EXISTS
6. Correlated Subqueries
7. Subqueries in the FROM Clause

5.2.1PostgreSQL Subquery as Scalar Operand

- A scalar subquery is a subquery that returns exactly one single value.
- It is normally executed in select query.
- It is an error to use a query that returns more than one row or more than one column as a scalar subquery.
- During a particular execution, if the subquery returns no rows, that is not an error; the scalar result is taken to be null.
- The subquery can refer to variables from the surrounding query, which will act as constants during any one evaluation of the subquery.

Example: PostgreSQL Subquery as Scalar Operand

Code:

```
SELECT employee_id, last_name,
(CASE WHEN department_id=(
SELECT department_id from departments WHERE location_id=2500)
THEN 'Canada' ELSE 'USA' END)
FROM employees;
```

Sample Output:

```
employee_id |  last_name   |  case
        100 | King         | USA
        101 | Kochhar      | USA
        102 | De Haan      | USA
        103 | Hunold       | USA
        104 | Ernst        | USA
        105 | Austin       | USA
        106 | Pataballa    | USA
        107 | Lorentz      | USA
        108 | Greenberg    | USA
        109 | Faviet       | USA
              ......................
107 rows in set (0.00 sec)
```

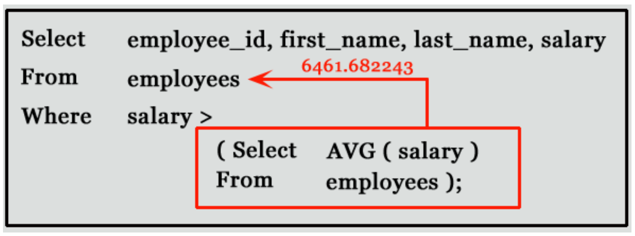
5.2.2 PostgreSQL Subqueries: Using Comparisons

A subquery can be used before or after any of the comparison operators. The subquery can return at most one value. The value can be the result of an arithmetic expression or a column function. SQL then compares the value that results from the subquery with the value on the other side of the comparison operator. You can use the following comparison operators :

Operator	Description
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

!=	Not equal to
<>	Not equal to
<=>	NULL-safe equal to operator

For example, suppose you want to find the employee id, first_name, last_name, and salaries for employees whose average salary is higher than the average salary throughout the company.



```
Select    employee_id, first_name, last_name, salary
From      employees
Where     salary >
              ( Select   AVG ( salary )
                From     employees );
```

Code:

```
SELECT employee_id,first_name,last_name,salary
FROM employees
WHERE salary >
(SELECT AVG(SALARY) FROM employees);
```

Sample Output:

employee_id	first_name	last_name	salary
100	Steven	King	24000
101	Neena	Kochhar	17000
102	Lex	De Haan	17000
103	Alexander	Hunold	9000
108	Nancy	Greenberg	12000
109	Daniel	Faviet	9000
110	John	Chen	8200
114	Den	Raphaely	11000
121	Adam	Fripp	8200
145	John	Russell	14000
146	Karen	Partners	13500
176	Jonathon	Taylor	8600
177	Jack	Livingston	8400
201	Michael	Hartstein	13000
204	Hermann	Baer	10000
205	Shelley	Higgins	12000
206	William	Gietz	8300

5.2.3 PostgreSQL Subqueries with ALL operator

Syntax:

```
expression operator ALL (subquery)
```

The ALL operator compares value to every value returned by the subquery. The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result.

1. The result of ALL is true if all rows yield true (including the case where the subquery returns no rows).
2. The result is false if any false result is found.
3. The result is NULL if the comparison does not return false for any row, and it returns NULL for at least one row.

Example: PostgreSQL Subquery, ALL operator

The following query selects the department with the highest average salary. The subquery finds the average salary for each department, and then the main query selects the department with the highest average salary.

Code:

```
SELECT department_id, AVG(SALARY)

FROM employees GROUP BY department_id

HAVING AVG(SALARY)>=ALL

(SELECT AVG(SALARY) FROM employees

GROUP BY department_id);
```

Sample Output:

department_id	avg
9	19333.33

Note: Here we have used ALL keyword for this subquery as the department selected by the query must have an average salary greater than or equal to all the average salaries of the other departments.

5.2.4 PostgreSQL Subqueries with ANY/SOME operator

Syntax:

```
expression operator ANY (subquery)
expression operator SOME (subquery)
```

The ANY operator compares the value to each value returned by the subquery. Therefore ANY keyword (which must follow a comparison operator) returns TRUE if the comparison is TRUE for ANY of the values in the column that the subquery returns.
SOME is a synonym for ANY. IN is equivalent to = ANY.

Example: PostgreSQL Subquery, ANY operator

The following query selects any employee who works in the location 1700. The subquery finds the department id in the 1700 location, and then the main query selects the employees who work in any of these departments.

departments table:

Code:

```
SELECT first_name, last_name,department_id
FROM employees
WHERE department_id= ANY
(SELECT DEPARTMENT_ID
FROM departments WHERE location_id=1700);
```

Sample Output:

first_name	last_name	department_id
Steven	King	9
Neena	Kochhar	9
Lex	De Haan	9
Nancy	Greenberg	10
Daniel	Faviet	10
John	Chen	10
Ismael	Sciarra	10
Jose Manuel	Urman	10
Luis	Popp	10
Den	Raphaely	3
Alexander	Khoo	3
Shelli	Baida	3
Sigal	Tobias	3
Guy	Himuro	3
Karen	Colmenares	3
Jennifer	Whalen	1
Shelley	Higgins	11
William	Gietz	11

Note: We have used ANY keyword in this query, because it is likely that the subquery will find more than one departments in 1700 location. If you use the ALL keyword instead of the ANY keyword, no data is selected because no employee works in all departments of 1700 location

5.2.5 PostgreSQL Subqueries with IN operator

Syntax:

```
expression IN (subquery)
```

The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result.

1. The result of IN is true if any equal subquery row is found.
2. The result is “false” if no equal row is found (including the case where the subquery returns no rows).
3. If the left-hand expression yields null, or if there are no equal right-hand values and at least one right-hand row yields null, the result of the IN construct will be null, not false.

Example: PostgreSQL Subquery, IN operator

The following query selects those employees who work in the location 1800. The subquery finds the department id in the 1800 location, and then the main query selects the employees who work in any of these departments.

Code:

```
SELECT first_name, last_name,department_id
FROM employees
WHERE department_id IN
```

```
(SELECT DEPARTMENT_ID FROM departments
WHERE location_id=1800);
```

Sample Output:

first_name	last_name	department_id
Michael	Hartstein	2
Pat	Fay	2

5.2.6 PostegreSQL Subqueries with NOT IN operator

Syntax:

```
expression NOT IN (subquery)
```

The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result.

1. The result of NOT IN is true if any equal subquery row is found.
2. The result is "false" if no equal row is found (including the case where the subquery returns no rows).
3. If the left-hand expression yields null, or if there are no equal right-hand values and at least one right-hand row yields null, the result of the IN construct will be null, not false.

Example: PostgreSQL Subquery, NOT IN operator

The following query selects those employees who does not work in those department where the managers of ID between 100 and 200 works. The subquery finds the department id which is under the manager whose id is between 100 and 200, and then the main query selects the employees who do not work in any of these departments.

Code:

```
SELECT first_name, last_name,department_id
FROM employees
WHERE department_id NOT IN
(SELECT DEPARTMENT_ID FROM departments
WHERE manager_id
BETWEEN 100 AND 200);
```

Sample Output:

first_name	last_name	department_id
Steven	King	9
Pat	Fay	2
William	Gietz	11

5.2.7 PostgreSQL Subqueries with EXISTS operator

Syntax:

```
EXISTS (subquery)
```

The argument of EXISTS is an arbitrary SELECT statement, or subquery. The subquery is evaluated to determine whether it returns any rows. If it returns at least one row, the result of EXISTS is true; if the subquery returns no rows, the result of EXISTS is false.

Example: PostgreSQL Subqueries with EXISTS

The following query finds employees (employee_id, first_name, last_name, job_id, department_id) from employees table who have at least one person reporting to them.

Code:

```
SELECT employee_id, first_name, last_name, job_id, department_id
FROM employees E
WHERE EXISTS
(SELECT * FROM employees
WHERE manager_id = E.employee_id);
```

Sample Output:

employee_id	first_name	last_name	job_id	department_id
100	Steven	King	4	9
101	Neena	Kochhar	5	9
102	Lex	De Haan	5	9
103	Alexander	Hunold	9	6
108	Nancy	Greenberg	7	10
114	Den	Raphaely	14	3
120	Matthew	Weiss	19	5
123	Shanta	Vollman	19	5
201	Michael	Hartstein	10	2
205	Shelley	Higgins	2	11

5.2.8 PostgreSQL Row Subqueries

A row subquery is a subquery that returns a single row and more than one column value. You can use = , >, <, >=, <=, <>, !=, <=> comparison operators. See the following examples:

Syntax:

```
row_constructor operator (subquery)
```

Example: PostgreSQL Row Subqueries

In the following examples, queries shows different result according to above conditions :

Code:

```
SELECT first_name
FROM employees
WHERE ROW(department_id, manager_id) =
(SELECT department_id, manager_id
FROM departments
WHERE location_id = 1800);
```

Sample Output:

first_name
Michael
Pat

5.2.9 PostgreSQL Subqueries in the FROM Clause

Although subqueries are more commonly placed in a WHERE clause, they can also form part of the FROM clause. Such subqueries are commonly called derived tables. If a subquery is used in this way, you must also use an AS clause to name the result of the subquery.

Examples

```
CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES

  ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),

  ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),

  ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),

  ('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);
```

Assume that, given the data above, you want to return the average total for all students. In other words, the average of Chun's 148 (75+73), Esben's 74 (43+31), etc.

You cannot do the following:

```
SELECT AVG(SUM(score)) FROM student GROUP BY name;

ERROR 1111 (HY000): Invalid use of group function
```

A subquery in the FROM clause is however permitted:

```
SELECT AVG(sq_sum) FROM (SELECT SUM(score) AS sq_sum FROM student GROUP BY name) A
S t;
```

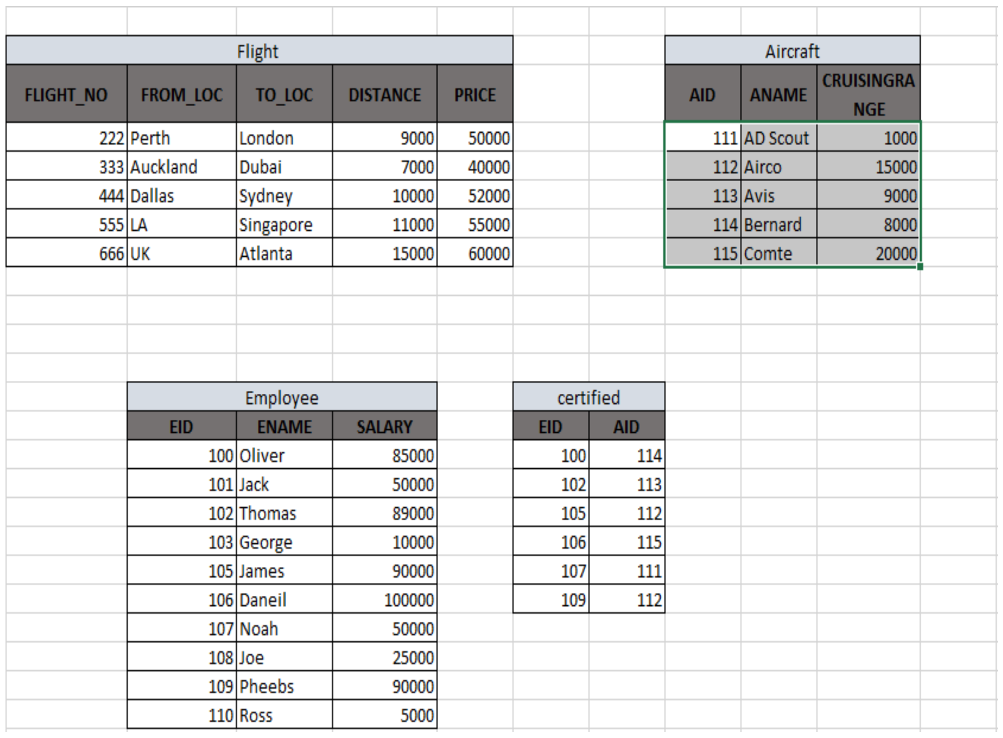
Practice exercises:

Dane country Airport officials decided that all the information related to the airline flight should be organized using a DBMS, and you are hired to design the database. Your first task is to organize the information about all the airplanes stationed and maintained at the airport. The following relations keep track of airline flight information**:**

Flights (flno, from_loc, to_loc, distance, price) //details about all the flights
Aircraft (aid, aname, cruisingrange) //details of the aircraft including the registration number assigned, model of the craft and the maximum capacity of the craft in terms of distance it can travel.
Certified (eid, aid) //Certification details of the pilots for specific crafts.
Employees (eid, ename, salary) //details of employees

Part A:

1. Display employee id along with the name of the aircraft for which he is certified for.
2. Modify above to display employee names too.
3. Display name of employee who is certified for the aircraft with the highest of cruising range.
4. Modify above and show his salary too.
5. Display name of the suitable air craft for the flight from LA (based upon the distance and cruising range of craft).



Flight

FLIGHT_NO	FROM_LOC	TO_LOC	DISTANCE	PRICE
222	Perth	London	9000	50000
333	Auckland	Dubai	7000	40000
444	Dallas	Sydney	10000	52000
555	LA	Singapore	11000	55000
666	UK	Atlanta	15000	60000

Aircraft

AID	ANAME	CRUISINGRANGE
111	AD Scout	1000
112	Airco	15000
113	Avis	9000
114	Bernard	8000
115	Comte	20000

Employee

EID	ENAME	SALARY
100	Oliver	85000
101	Jack	50000
102	Thomas	89000
103	George	10000
105	James	90000
106	Daneil	100000
107	Noah	50000
108	Joe	25000
109	Pheebs	90000
110	Ross	5000

certified

EID	AID
100	114
102	113
105	112
106	115
107	111
109	112

Solution:

```
-1
Select result2.EID,ANAME from Aircraft,
(Select Employee.EID,result1.AID from Employee,(Select EID,AID from Certified) result1
where Employee.EID = result1.EID) result2
 where Aircraft.AID = result2.AID ;

--2
Select result2.EID,result2.Ename,ANAME from Aircraft,
(Select Employee.EID,Employee.Ename,result1.AID from Employee,(Select EID,AID from
Certified) result1 where Employee.EID = result1.EID) result2
 where Aircraft.AID = result2.AID ;

--3
SELECT * FROM
(Select Ename,result3.cruisingrange from (
Select result2.EID,result2.Ename,ANAME,Cruisingrange from Aircraft,
(Select Employee.EID,Employee.Ename,result1.AID from Employee,(Select EID,AID from
Certified) result1 where Employee.EID = result1.EID) result2
```

```
 where Aircraft.AID = result2.AID
)result3 ORDER BY result3.cruisingrange DESC
) WHERE ROWNUM=1;

--4
SELECT * FROM
(Select Ename,result3.salary from (
Select result2.EID,result2.salary,result2.Ename,ANAME,Cruisingrange from Aircraft,
(Select Employee.EID,Employee.salary,Employee.Ename,result1.AID from Employee,(Select
EID,AID from Certified) result1 where Employee.EID = result1.EID) result2
 where Aircraft.AID = result2.AID
)result3 ORDER BY result3.cruisingrange DESC
) WHERE ROWNUM=1;

--5
Select Cruisingrange from Aircraft where Cruisingrange >=
(Select Distance from Flight where from_loc='LA');
```