CS232L – Database Management System

LAB 03

SQL CLAUSES- ORDER BY, GROUP BY, HAVING

Ghulam Ishaq Khan Institute of Engineering Sciences

Faculty of Computer Science & Engineering

CS232L – Database Management system Lab

Lab 3 –SQL Clauses (Order By, Group By, Having)

Objective

The objective of this session is to get 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.

3.1 Clauses in SQL

An important component for Analyst to summarize the data such as sales, profit, cost, and salary. Data Summarization is very helpful for Analyst to create a visualization, conclude findings, and report writing. In SQL, GROUP BY Clause is one of the tools to summarize or aggregate the data series. For example, sum up the daily sales and combine in a single quarter and show it to the senior management. Similarly, if you want to count how many employees in each department of the company. It groups the databases on the basis of one or more column and aggregates the results.

After Grouping the data, you can filter the grouped record using HAVING Clause. HAVING Clause returns the grouped records which match the given condition. You can also sort the grouped records using ORDER BY. ORDER BY used after GROUP BY on aggregated column. Clauses help us filter and analyze data quickly. When we have large amounts of data stored in the database, we use Clauses to query and get data required by the user.

3.2 Introduction to GROUP BY clause

The GROUP BY clause divides the rows returned from the SELECT statement into groups. For each group, you can apply an aggregate function e.g., SUM() to calculate the sum of items or COUNT() to get the number of items in the groups. The following statement illustrates the basic syntax of the GROUP BY clause:

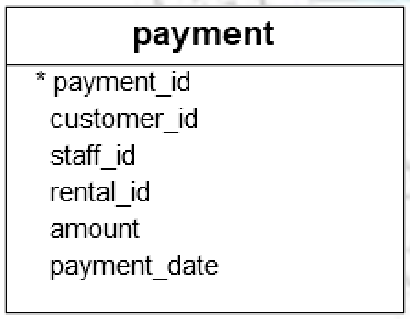
```
SELECT
   column_1,
   column_2,
   ...,
   aggregate_function(column_3)
FROM
   table_name
GROUP BY
   column_1,
   column_2,
   ...;
```

In this syntax:

- First, select the columns that you want to group e.g., column1 and column2, and column that you want to apply an aggregate function (column3).
- Second, list the columns that you want to group in the GROUP BY clause.

3.2.1 GROUP BY **clause examples**

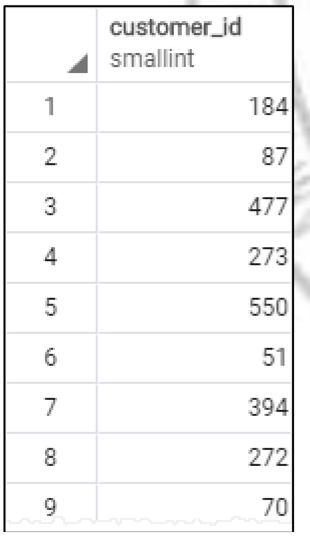
Let’s take a look at the payment table in the sample database of PostgreSQL i.e. dvdrental



3.2.2 Using GROUP BY without an aggregate function example

You can use the GROUP BY clause without applying an aggregate function. The following query gets data from the payment table and groups the result by customer id.

```
SELECT
  customer_id
FROM
  payment
GROUP BY
  customer_id;
```



	customer_id smallint
1	184
2	87
3	477
4	273
5	550
6	51
7	394
8	272
9	70

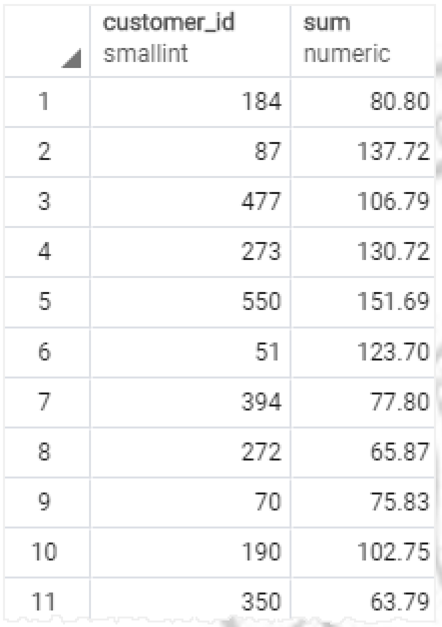
In this case, the GROUP BY works like the DISTINCT clause that removes duplicate rows from the result set.

3.2.3) Using GROUP BY **with** SUM() **function example**

The GROUP BY clause is useful when it is used in conjunction with an aggregate function. For example, to select the total amount that each customer has been paid, you use the GROUP BY clause to divide the rows in the payment table into groups grouped by customer id. For each group, you calculate the total amounts using the SUM() function.

The following query uses the GROUP BY clause to get the total amount that each customer has been paid:

```
SELECT
        customer_id,
        SUM (amount)
FROM
        payment
GROUP BY
        customer_id;
```



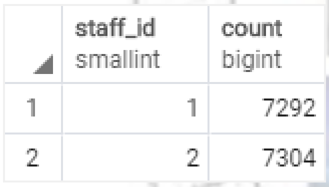
	customer_id smallint	sum numeric
1	184	80.80
2	87	137.72
3	477	106.79
4	273	130.72
5	550	151.69
6	51	123.70
7	394	77.80
8	272	65.87
9	70	75.83
10	190	102.75
11	350	63.79

The GROUP BY clause sorts the result set by customer id and adds up the amount that belongs to the same customer. Whenever the customer_id changes, it adds the row to the returned result set.

3.2.4) Using GROUP BY with COUNT() function example

To find the number of payment transactions that each staff has processed, you group the rows in the payment table by the values in the staff_id column and use the COUNT() function to get the number of transactions:

```
SELECT
        staff_id,
        COUNT (payment_id)
FROM
        payment
GROUP BY
        staff_id;
```



	staff_id smallint	count bigint
1	1	7292
2	2	7304

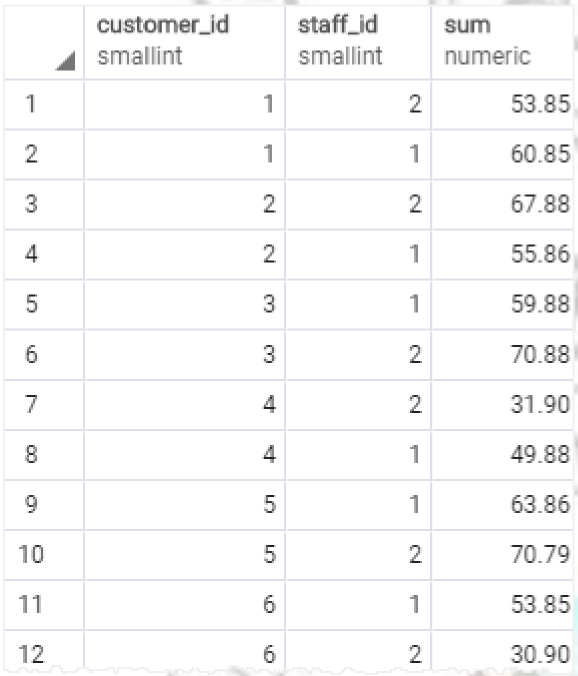
The GROUP BY clause divides the rows in the payment into groups and groups them by value in the staff_id column. For each group, it returns the number of rows by using the COUNT() function.

3.2.5) Using PostgreSQL GROUP BY with multiple columns

The following example uses multiple columns in the GROUP BY clause:

```
SELECT
	customer_id,
	staff_id,
	SUM(amount)
FROM
	payment
GROUP BY
	staff_id,
	customer_id
ORDER BY
    customer_id;
```

In this example, the GROUP BY clause divides the rows in the payment table by the values in the customer_id and staff_id columns. For each group of (customer_id, staff_id), the SUM() calculates the total amount.



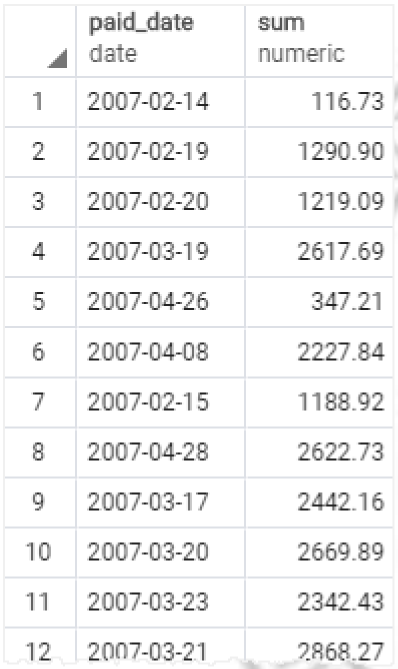
	customer_id smallint	staff_id smallint	sum numeric
1	1	2	53.85
2	1	1	60.85
3	2	2	67.88
4	2	1	55.86
5	3	1	59.88
6	3	2	70.88
7	4	2	31.90
8	4	1	49.88
9	5	1	63.86
10	5	2	70.79
11	6	1	53.85
12	6	2	30.90

3.2.6) Using GROUP BY clause with a date column

The payment_date is a timestamp column. To group payments by dates, you use the DATE() function to convert timestamps to dates first and then group payments by the result date:

```
SELECT
	DATE(payment_date) paid_date,
	SUM(amount) sum
FROM
	payment
GROUP BY
	DATE(payment_date);
```

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



	paid_date date	sum numeric
1	2007-02-14	116.73
2	2007-02-19	1290.90
3	2007-02-20	1219.09
4	2007-03-19	2617.69
5	2007-04-26	347.21
6	2007-04-08	2227.84
7	2007-02-15	1188.92
8	2007-04-28	2622.73
9	2007-03-17	2442.16
10	2007-03-20	2669.89
11	2007-03-23	2342.43
12	2007-03-21	2868.27

3.3 Introduction to HAVING clause.

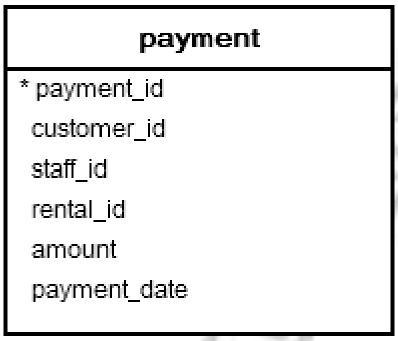
The HAVING clause is often used with the GROUP BY clause to filter groups or aggregates based on a specified condition. The following statement illustrates the basic syntax of the HAVING clause:

```
SELECT
        column1,
        aggregate_function (column2)
FROM
        table_name
GROUP BY
        column1
HAVING
        condition;
```

In this syntax, the group by clause returns rows grouped by the column1.The HAVING clause specifies a condition to filter the groups. PostgreSQL evaluates the HAVING clause after the FROM, WHERE, GROUP BY, and before the SELECT, DISTINCT, ORDER BY and LIMIT clauses. Since the HAVING clause is evaluated before the SELECT clause, you cannot use column aliases in the HAVING clause. Because at the time of evaluating the HAVING clause, the column aliases specified in the SELECT clause are not available.

3.3.1 HAVING clause examples

Let’s take a look at the payment table in the sample database.

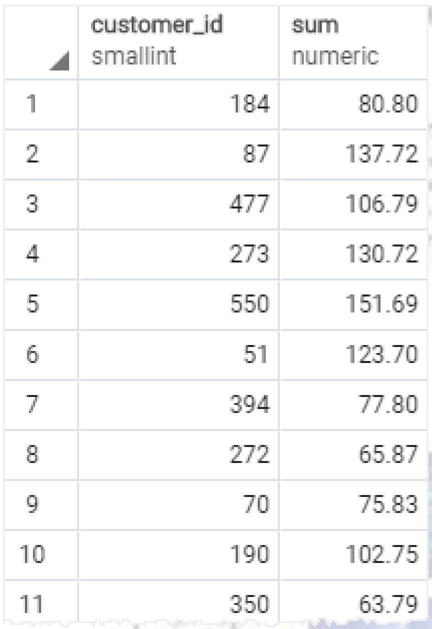


payment
* payment_id
customer_id
staff_id
rental_id
amount
payment_date

3.3.2 HAVING clause with SUM function example

The following query uses the GROUP BY clause with the SUM() function to find the total amount of each customer:

```
SELECT
        customer_id,
        SUM (amount)
FROM
        payment
GROUP BY
        customer_id;
```

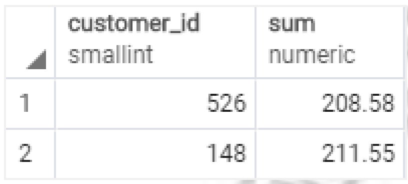


	customer_id smallint	sum numeric
1	184	80.80
2	87	137.72
3	477	106.79
4	273	130.72
5	550	151.69
6	51	123.70
7	394	77.80
8	272	65.87
9	70	75.83
10	190	102.75
11	350	63.79

The following statement adds the HAVING clause to select the only customers who have been spending more than 200:

```
SELECT
        customer_id,
        SUM (amount)
FROM
        payment
GROUP BY
        customer_id
HAVING
```

```
        SUM (amount) > 200;
```



	customer_id smallint	sum numeric
1	526	208.58
2	148	211.55

3.4 Difference between HAVING and WHERE Clause

The following table shows the comparisons between these two clauses, but the main difference is that the WHERE clause uses condition for filtering records before any groupings are made, while HAVING clause uses condition for filtering values from a group.

HAVING	WHERE
1. The HAVING clause is used in database systems to fetch the data/values from the groups according to the given condition.	1. The WHERE clause is used in database systems to fetch the data/values from the tables according to the given condition.
2. The HAVING clause is always executed with the GROUP BY clause.	2. The WHERE clause can be executed without the GROUP BY clause.
3. The HAVING clause can include SQL aggregate functions in a query or statement.	3. We cannot use the SQL aggregate function with WHERE clause in statements.
4. We can only use SELECT statement with HAVING clause for filtering the records.	4. Whereas, we can easily use WHERE clause with UPDATE, DELETE, and SELECT statements.
5. The HAVING clause is used in SQL queries after the GROUP BY clause.	5. The WHERE clause is always used before the GROUP BY clause in SQL queries.
6. It is a post-filter.	6. It is a pre-filter.
7. It is used to filter groups.	7. It is used to filter the single record of the table.

3.5 Introduction to ORDER BY clause

When you query data from a table, the SELECT statement returns rows in an unspecified order. To sort the rows of the result set, you use the ORDER BY clause in the SELECT statement.

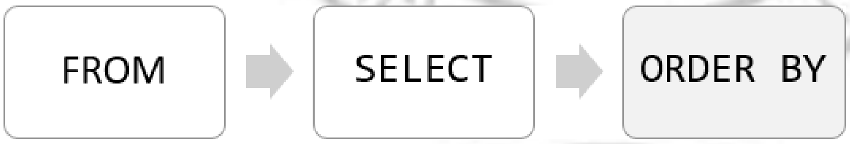
The ORDER BY clause allows you to sort rows returned by a SELECT clause in ascending or descending order based on a sort expression.The following illustrates the syntax of the ORDER BY clause:

```
SELECT
        select_list
FROM
        table_name
ORDER BY
        sort_expression1 [ASC | DESC],
        ...
        sort_expressionN [ASC | DESC];
```

In this syntax:

- First, specify a sort expression, which can be a column or an expression, that you want to sort after the ORDER BY keywords. If you want to sort the result set based on multiple columns or expressions, you need to place a comma (,) between two columns or expressions to separate them.
- Second, you use the ASC option to sort rows in ascending order and the DESC option to sort rows in descending order. If you omit the ASC or DESC option, the ORDER BY uses ASC by default.

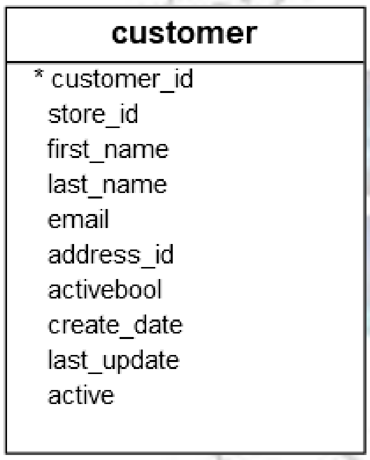
PostgreSQL evaluates the clauses in the SELECT statment in the following order: FROM, SELECT, and ORDER BY:



Due to the order of evaluation, if you have a column alias in the SELECT clause, you can use it in the ORDER BY clause.

3.5.1 ORDER BY Clauses examples

We will use the customer table in the sample database for the demonstration.

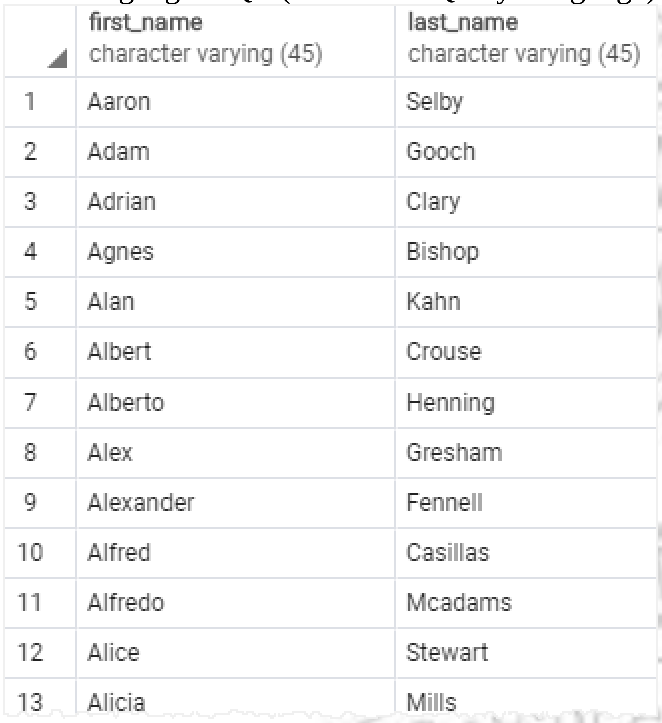


3.5.2) ORDER BY clause to sort rows by one column

The following query uses the ORDER BY clause to sort customers by their first names in ascending order:

```
SELECT
        first_name,
        last_name
FROM
        customer
ORDER BY
        first_name ASC;
```

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



	first_name character varying (45)	last_name character varying (45)
1	Aaron	Selby
2	Adam	Gooch
3	Adrian	Clary
4	Agnes	Bishop
5	Alan	Kahn
6	Albert	Crouse
7	Alberto	Henning
8	Alex	Gresham
9	Alexander	Fennell
10	Alfred	Casillas
11	Alfredo	Mcadams
12	Alice	Stewart
13	Alicia	Mills

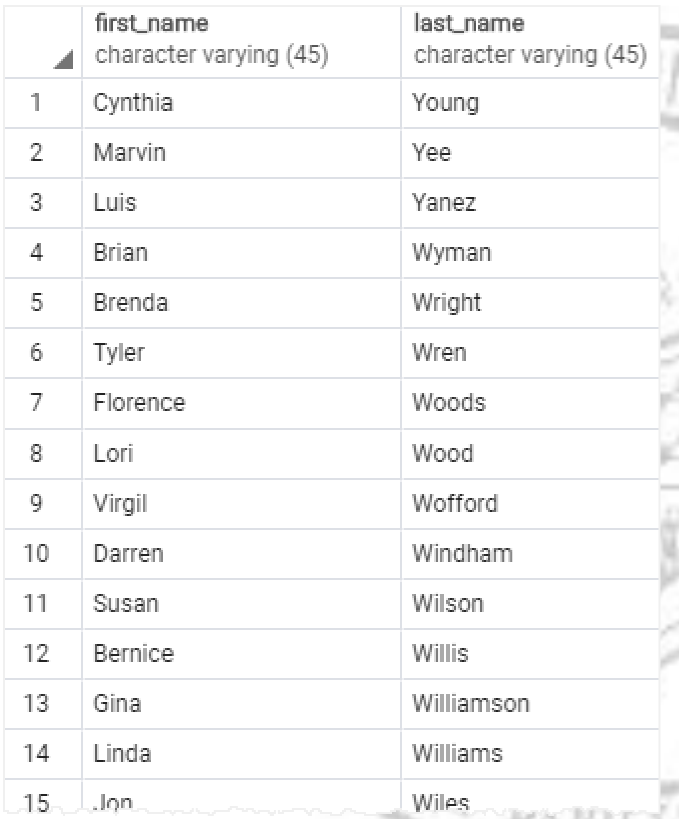
Since the ASC option is the default, you can omit it in the ORDER BY.

3.5.3 ORDER BY clause to sort rows by one column in descending order

The following statement selects the first name and last name from the customer table and sorts the rows by values in the last name column in descending order:

```
SELECT
       first_name,
       last_name
FROM
       customer
ORDER BY
       last_name DESC;
```

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



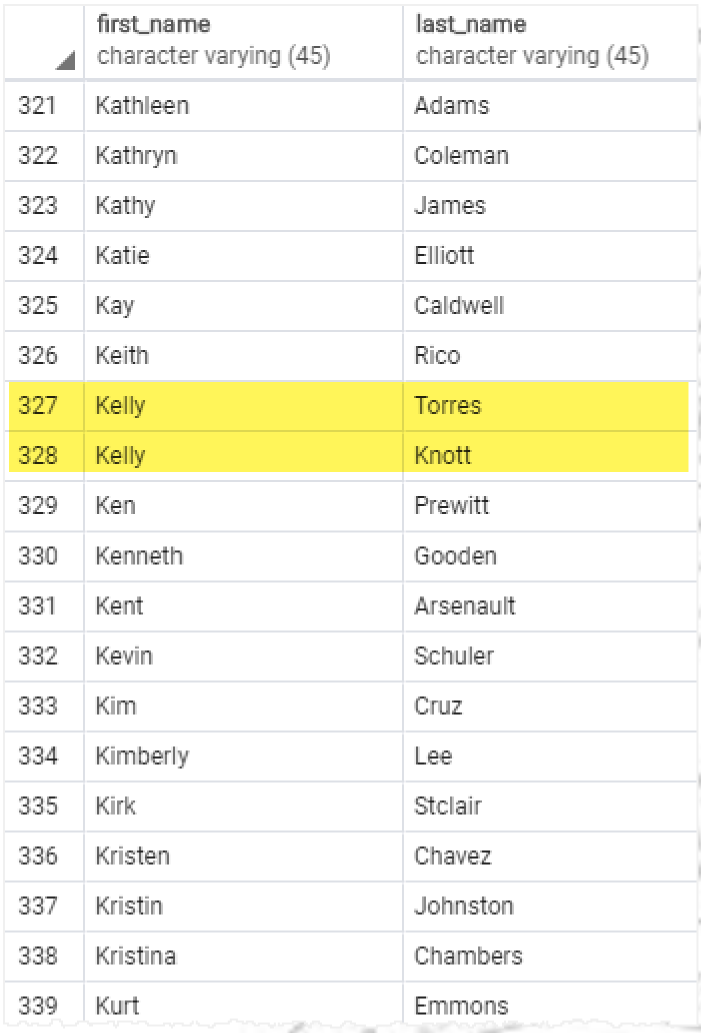
	first_name character varying (45)	last_name character varying (45)
1	Cynthia	Young
2	Marvin	Yee
3	Luis	Yanez
4	Brian	Wyman
5	Brenda	Wright
6	Tyler	Wren
7	Florence	Woods
8	Lori	Wood
9	Virgil	Wofford
10	Darren	Windham
11	Susan	Wilson
12	Bernice	Willis
13	Gina	Williamson
14	Linda	Williams
15	Jon	Wiles

3.5.4 ORDER BY clause to sort rows by multiple columns

The following statement selects the first name and last name from the customer table and sorts the rows by the first name in ascending order and last name in descending order:

```
SELECT
        first_name,
        last_name
FROM
        customer
ORDER BY
        first_name ASC,
        last_name DESC;
```

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



	first_name character varying (45)	last_name character varying (45)
321	Kathleen	Adams
322	Kathryn	Coleman
323	Kathy	James
324	Katie	Elliott
325	Kay	Caldwell
326	Keith	Rico
327	Kelly	Torres
328	Kelly	Knott
329	Ken	Prewitt
330	Kenneth	Gooden
331	Kent	Arsenault
332	Kevin	Schuler
333	Kim	Cruz
334	Kimberly	Lee
335	Kirk	Stclair
336	Kristen	Chavez
337	Kristin	Johnston
338	Kristina	Chambers
339	Kurt	Emmons

In this example, the ORDER BY clause sorts rows by values in the first name column first. And then it sorts the sorted rows by values in the last name column. As you can see clearly from the output, two customers with the same first name Kelly have the last name sorted in descending order.

Practice Problem:

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. Create the above-mentioned tables and insert data in tables as given below.

```
create table flight
  (
     flight_no number,
     from_loc varchar(20),
     to_loc varchar(20),
     distance number,
     price number
  );

create table aircraft
  (
     aid number,
     aname varchar(20),
     cruisingrange number
  );

create table certified
  (
     eid number,
     aid number
  );

create table employee
  (
     eid number,
     ename varchar(20),
     salary number
  );
insert all
  into flight values
(222,'Perth','London',9000,50000)
  , (333,'Auckland','Dubai',7000,40000)
  , (444,'Dallas','Sydeny',10000,52000)
  , (555, 'LA','Singapore',11000,55000)
  , (666, 'UK','Atlanta',15000,60000);
insert all
  into aircraft values (111, 'AD Scout',1000)
  into aircraft values (112, 'Airco', 15000)
  into aircraft values (113, 'Avis', 9000)
  into aircraft values (114, 'Bernard', 8000)
  into aircraft values (115, 'Comte', 20000)
  select * from dual;
insert all
  into employee values (100, 'Oliver', 85000)
  into employee values (101, 'Jack', 50000)
  into employee values (102, 'Thomas',
89000)
  into employee values (103, 'George', 10000)
  into employee values (105, 'James', 90000)
  into employee values (106, 'Daneil', 100000)
  into employee values (107, 'Noah', 50000)
  into employee values (108, 'Joe', 25000)
  into employee values (109, 'Pheebs', 90000)
  into employee values (110, 'Ross', 5000)
  select * from dual;

insert all
  into certified values (100,114)
  into certified values (102,113)
  into certified values (105,112)
  into certified values (106,115)
  into certified values (107,111)
  into certified values (108,112)
 select * from dual;
```

Part B:

1. Calculate total tickets price sold for all flights.

```
select sum(PRICE) from Flight;
```

2. Display employees name in ascending order.

```
SELECT ENAME FROM EMPLOYEE ORDER BY ENAME;
```

3. Show total bonus given to employees.

```
SELECT SUM(BONUS) FROM EMPLOYEE;
```

4. For each bonus value, check number of employees given same bonus.

```
SELECT COUNT(EID) AS NO_OF_EMPLOYEES,BONUS FROM
EMPLOYEE GROUP BY BONUS;
```

5. Add flight having following data:
   777,UK,Sydney,20000,65000
   888,UK,Dallas,Atlanta,25000,2000

```
INSERT ALL
INTO FLIGHT VALUES(777,'UK','Sydney',20000,65000)
, (888,'UK','Dallas',25000,2000);
```

6. Calculate total price of flights based upon the location from where it is flying. (from location).

```
SELECT SUM(PRICE) FROM FLIGHT GROUP BY FROM_LOC;
```

7. Display sum of salaries of all employees having name starting with 'J' and sum of salaries is above 20000.

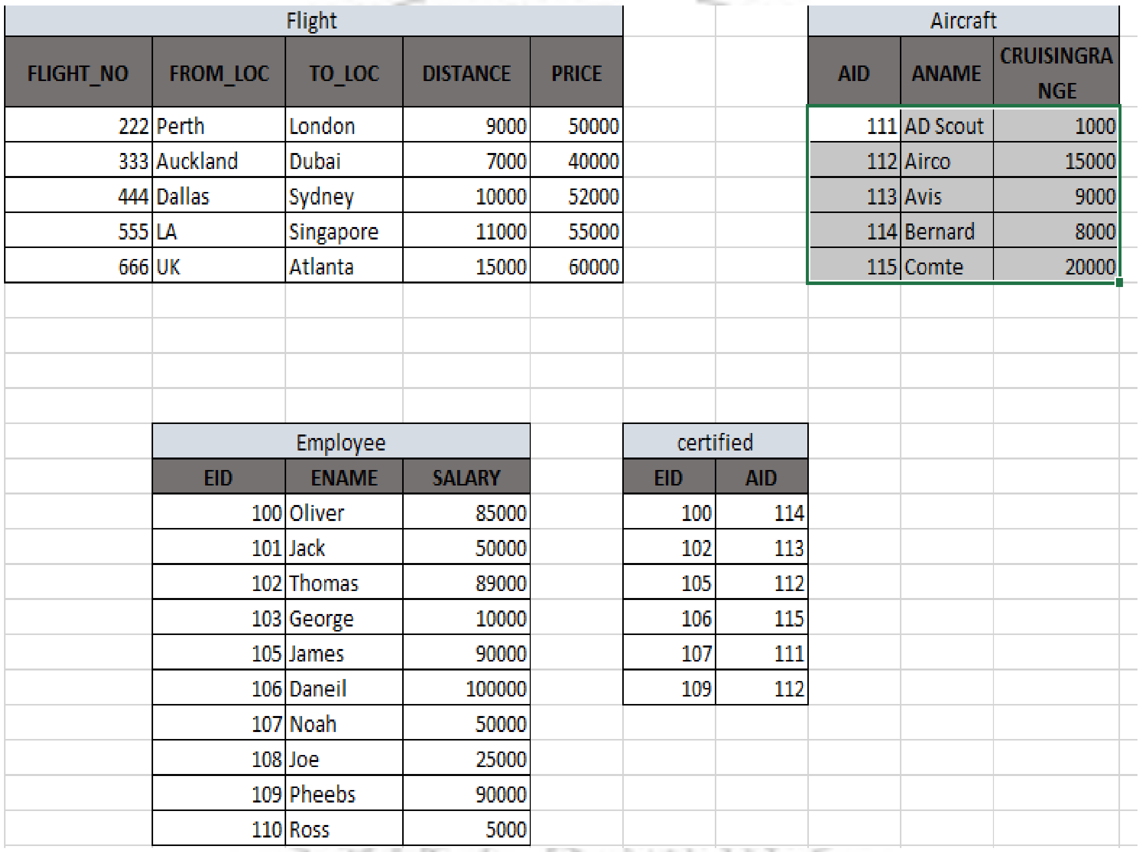
```
SELECT SUM(SALARY) FROM EMPLOYEE WHERE ENAME LIKE 'J%' AND
SALARY>20000;
```

8. Display aircraft data in descending order based upon their cruising range.

```
SELECT * FROM AIRCRAFT ORDER BY CRUISINGRANGE DESC;
```

9. Display flights row count.

```
SELECT COUNT(FLIGHT_NO) FROM FLIGHT;
```



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