Experiment No. 4

Environment: Microsoft Windows
Tools/ Language: Oracle

OBJECTIVE: To implement the concept of aggregating & grouping of Data.
Theory:

Group Functions:
Group functions operate on set of rows, result is based on group of rows rather than one result per row as returned by single row functions.

1) Avg return average value of n
Syntax:
Avg ([distinct/all] n)

2) Min return minimum value of expr.
Syntax:
MIN([distinct/all] expr)

3) Count Returns the no of rows where expr is not null
Syntax:
Count ([distinct/all] expr)
Count (*) Returns the no rows in the table, including duplicates and those with nulls.

4) Max Return max value of expr
Syntax:
Max ([distinct/all]expr)

5) Sum Returns sum of values of n
Syntax:
Sum ([distinct/all]n)

Example Queries:
Aggregate Functions Examples:
select AVG(GPA) as "Average GPA" from student;
Average GPA

3.56666667

Select AVG(DISTINCT GPA) as "Distinct GPA" from student
Distinct GPA

3.54285714

Select MIN(GPA) as "Minimum GPA" from student
Minimum GPA

2.9

Select COUNT(*) from student

```
COUNT(*)
--------
      12
```

Select COUNT(sName) from student

```
COUNT(SNAME)
--------
      12
```

select COUNT(distinct sName) from student

```
COUNT(DISTINCTSNAME)
------------------------
            10
```

Select MAX(GPA) from student

```
MAX(GPA)
------
   3.9
```

Select SUM(GPA) from student

```
SUM(GPA)
--------
   42.8
```

Q.Find average GPA without using Average function.

Select SUM(GPA)/COUNT(GPA) as "Average" from student

```
  Average
-------
3.56666667
```

<u>Grouping Data From Tables:</u>

There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples, we specify this wish in SQL using the group by clause. The attribute or attributes given in the group by clause are used to form group. Tuples with the same value on all attributes in the group by clause are placed in one group.

Syntax:

```
SELECT columnname, columnname
FROM tablename
GROUP BY columnname;
```

<u>HAVING clause</u>:

The HAVING clause can be used in conjunction with the GROUP BY clause. HAVING imposes a condition on the GROUP BY clause, this further filters the groups created by the GROUP BY clause.

HAVING clause can be used to find duplicates in a relation or in other words find unique values in the situations where DISTINCT cannot apply.

NOTE: Each column specifications specified in HAVING clause must occur in the list of columns named in the GROUP BY clause.

Example

1. select state, count(cname) from college GROUP BY state;

STATE	COUNT(CNAME)
CA	2
MA	1
NY	1

2. select state, count(cname) from college GROUP BY state having count(cname)>1;

STATE	COUNT(CNAME)
CA	2

Practical Assignment - 4

Department: Computer Engineering & Applications

Course: B.Tech. (CSE)

Subject: Database Management System Lab (BCSE 0802)

Year: 2nd **Semester:** 3rd

Run the following Script:

```
BEGIN
 FOR cur_rec IN (SELECT object_name, object_type
                 FROM user_objects
                WHERE object_type IN
                      ('TABLE',
                       'VIEW',
                       'PACKAGE',
                       'PROCEDURE',
                       'FUNCTION',
                       'SEQUENCE'
                      ))
 LOOP
   BEGIN
     IF cur_rec.object_type = 'TABLE'
     THEN
       EXECUTE IMMEDIATE  'DROP '
                       || cur_rec.object_type
                       || ' ""
                       || cur_rec.object_name
                       || '" CASCADE CONSTRAINTS';
     ELSE
       EXECUTE IMMEDIATE  'DROP '
                       || cur_rec.object_type
                       || ' ""
                       || cur_rec.object_name
                       || "";
     END IF;
   EXCEPTION
     WHEN OTHERS
     THEN
       DBMS_OUTPUT.put_line (  'FAILED: DROP '
                            || cur_rec.object_type
                            || ' ""
                            || cur_rec.object_name
                            || ""
                           );
   END;
END LOOP;
END;
/

commit;
```

```
drop  table  College;
drop  table  Student;
drop table  Apply;

create table College(cName varchar2(10) primary key, state
varchar2(10), enrollment int);
create table Student(sID int primary key, sName varchar2(10), GPA
real, sizeHS int);
create table Apply(sID int, cName varchar2(10), major varchar2(20),
decision char(1), primary key(sID, major, cName), constraint sID_fk
Foreign key(sID) references Student, constraint cName_fk Foreign
key(cName) references College);

delete from Student;
delete from College;
delete from Apply;

insert into Student values (123, 'Amy', 3.9, 1000);
insert into Student values (234, 'Bob', 3.6, 1500);
insert into Student values (345, 'Craig', 3.5, 500);
insert into Student values (456, 'Doris', 3.9, 1000);
insert into Student values (567, 'Edward', 2.9, 2000);
insert into Student values (678, 'Fay', 3.8, 200);
insert into Student values (789, 'Gary', 3.4, 800);
insert into Student values (987, 'Helen', 3.7, 800);
insert into Student values (876, 'Irene', 3.9, 400);
insert into Student values (765, 'Jay', 2.9, 1500);
insert into Student values (654, 'Amy', 3.9, 1000);
insert into Student values (543, 'Craig', 3.4, 2000);
insert into College values ('Stanford', 'CA', 15000);
insert into College values ('Berkeley', 'CA', 36000);
insert into College values ('MIT', 'MA', 10000);
insert into College values ('Cornell', 'NY', 21000);
insert into College values ('Harvard', 'MA', 50040);
insert into Apply values (123, 'Stanford', 'CS', 'Y');
insert into Apply values (123, 'Stanford', 'EE', 'N');
insert into Apply values (123, 'Berkeley', 'CS', 'Y');
insert into Apply values (123, 'Cornell', 'EE', 'Y');
insert into Apply values (234, 'Berkeley', 'biology', 'N');
insert into Apply values (345, 'MIT', 'bioengineering', 'Y');
insert into Apply values (345, 'Cornell', 'bioengineering', 'N');
insert into Apply values (345, 'Cornell', 'CS', 'Y');
insert into Apply values (345, 'Cornell', 'EE', 'N');
insert into Apply values (678, 'Stanford', 'history', 'Y');
insert into Apply values (987, 'Stanford', 'CS', 'Y');
insert into Apply values (987, 'Berkeley', 'CS', 'Y');
insert into Apply values (876, 'Stanford', 'CS', 'N');
insert into Apply values (876, 'MIT', 'biology', 'Y');
insert into Apply values (876, 'MIT', 'marine biology', 'N');
insert into Apply values (765, 'Stanford', 'history', 'Y');
insert into Apply values (765, 'Cornell', 'history', 'N');
insert into Apply values (765, 'Cornell', 'psychology', 'Y');
insert into Apply values (543, 'MIT', 'CS', 'N');
commit;
```

Student

sID	sName	GPA	sizeHS
123	Amy	3.9	1000
234	Bob	3.6	1500
345	Craig	3.5	500
456	Doris	3.9	1000
567	Edward	2.9	2000
678	Fay	3.8	200
789	Gary	3.4	800
987	Helen	3.7	800
876	Irene	3.9	400
765	Jay	2.9	1500
654	Amy	3.9	1000
543	Craig	3.4	2000

College

cName	state	enrollment
Stanford	CA	15000
Berkeley	CA	36000
MIT	MA	10000
Cornell	NY	21000
Harvard	MA	50040

Apply

sID	cName	major	decision
123	Stanford	CS	Y
123	Stanford	EE	N
123	Berkeley	CS	Y
123	Cornell	EE	Y
234	Berkeley	biology	N
345	MIT	bioengineering	Y
345	Cornell	bioengineering	N
345	Cornell	CS	Y
345	Cornell	EE	N
678	Stanford	history	Y
987	Stanford	CS	Y
987	Berkeley	CS	Y
876	Stanford	CS	N
876	MIT	biology	Y
876	MIT	marine biology	N
765	Stanford	history	Y
765	Cornell	history	N
765	Cornell	psychology	Y
543	MIT	CS	N

Solve the following:

Q1. Count the total number of Students.

Q2. Calculate the average GPA of all Student.

Q3. Determine the minimum and maximum GPA. Rename the titles as ‘max_GPA’ and ‘min_GPA’ respectively.

Q4. Count the number of students having GPA greater than or equal to 3.7.

Q5. Find Maximum, Average, Minimum, total GPA of all student.

Q6. Find total number of colleges in our Application Database.

Q7. Find how many different majors student had applied in.

Q8. Find total no. of Applications in our Application System’s Database.

Q9. Find average of all distinct GPA.

Q10. Display the total number of application accepted.

Q11. Find number of students having GPA>3.4 and coming from high school having size>1000.

Q12. Find how many student applied to ‘marine biology’.

Q13. Find how many applications were rejected and accepted by the colleges.

Q14. Find how many students applied to a particular major. (show count(sid) as No_of_applications).

Q15. Find number of applications received by particular college.

Q16. Find number of applications received in a particular major at a particular college.

Q17. Give the college name and major, where number of applications received are greater than or equal to 2.

Q18. Give the name and no of applications of all those colleges which receives applications from 3 or more students.

Q19. Give state and number of colleges of a state that has more than 1 college.

Q20. Find the name of students that are duplicate.

EXERCISE

Q1. Find how many applications are filed by each student. [*Hint: use left join as we need information about all 12 students here. If they applied nowhere than show zero in front of them*]

Q2. Provide name of students that file 3 or more applications.

Q3. Provide name of student who have not applied to any college.

Q4. Find maximum GPA, Average GPA, and minimum GPA among applicants of each college. (i.e. *say sID 123, 324 and 987 had applied to Berkley then compute and display max GPA among these three*)

Q5. Find how many student have same GPA among all students. (*provide this frequency in two column table as GPA 3.9 is 4 times, GPA 2.9 is 2 times*)

Q6. Find how many student have their name started from A, B or C.

Pre Experiment Questions

1. Difference between order by clause and group by clause?
2. Can we apply group by clause while joining two tables?

Post Experiment Questions

1. Can we select an attribute which we have not grouped?
2. How to filter results of group?
3. Do count function count NULL values?