Reporting Aggregated Data Using the Group Functions

What Are Group Functions?

Group functions operate on sets of rows to give one result per group.

EMPLOYEES

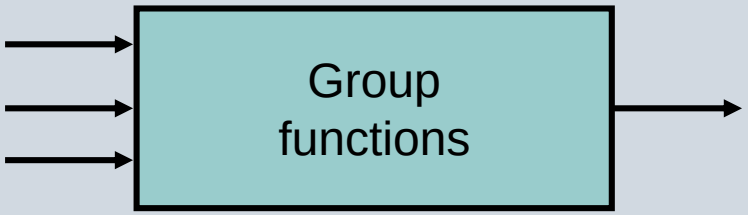
	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
...		
18	80	11000
19	80	8600
20	(null)	7000

Maximum salary in EMPLOYEES table

MAX(SALARY)
24000

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE



Group Functions: Syntax

```
SELECT      group_function(column), ...
FROM        table
[WHERE      condition];
```

Using the AVG and SUM Functions

You can use AVG and SUM for numeric data.

Find the average, max, min and sum of the salary's of REPresentatives.

```
SELECT  AVG(salary), MAX(salary),
        MIN(salary), SUM(salary)
FROM    employees
WHERE   job_id LIKE '%REP%';
```

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

Using the MIN and MAX Functions

You can use MIN and MAX for numeric, character, and date data types.

Find the minimum and maximum hire dates of employees. (most recently and the oldest hire dates)

```
SELECT MIN(hire_date), MAX(hire_date)
FROM       employees;
```

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	17-JUN-87	29-JAN-00

Using the COUNT Function

`COUNT(*)` returns the number of rows in a table:

Find the number of employees working in the department 50.

1

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(*)
1	5

`COUNT(expr)` returns the number of rows with non-null values for *`expr`*.

Find the number of employees taking commissions in department 50. (Take care of null values)

2

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(COMMISSION_PCT)
1	0

Using the `DISTINCT` Keyword

`COUNT(DISTINCT expr)` returns the number of distinct non-null values of *`expr`*.

To display the number of distinct department values in the `EMPLOYEES` table:

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

Group Functions and Null Values

Group functions ignore null values in the column &
The `NVL` function forces group functions to include null values.

Q1: Find the average commission_pct of employees. (Real average)

Q2: Find the average commission_pct of all employees. (General average)

Group Functions and Null Values

Group functions ignore null values in the column:

Find the average commission_pct of employees. (Real average)

1

```
SELECT AVG(commission_pct)
FROM   employees;
```

	AVG(COMMISSION_PCT)
1	0.2125

The `NVL` function forces group functions to include null values:

Find the average commission_pct of all employees. (General average)

2

```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

Creating Groups of Data

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	80	8600
...		
18	110	8300
19	110	12000
20	(null)	7000

4400

9500

3500

6400

10033

Average salary in the EMPLOYEES table for each department

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	20	9500
3	90	19333.333333333333...
4	110	10150
5	50	3500
6	80	10033.333333333333...
7	10	4400
8	60	6400

Creating Groups of Data: `GROUP BY` Clause Syntax

You can divide rows in a table into smaller groups by using the `GROUP BY` clause.

```
SELECT     column, group_function(column)
FROM       table
[WHERE     condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

Using the GROUP BY Clause

All the columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

Find the average salaries of all departments.

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY department_id ;
```

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	20	9500
3	90	19333.333333333333...
4	110	10150
5	50	3500
6	80	10033.333333333333...
7	10	4400
8	60	6400

Using the GROUP BY Clause

The GROUP BY column does not have to be in the SELECT list.

```
SELECT    AVG(salary)
FROM      employees
GROUP BY department_id ;
```

	AVG(SALARY)
1	7000
2	9500
3	19333.333333333333333333...
4	10150
5	3500
6	10033.333333333333333333...
7	4400
8	6400

Grouping by More Than One Column

EMPLOYEES

	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	2500
5	50	ST_CLERK	2600
6	50	ST_CLERK	3100
7	50	ST_CLERK	3500
8	50	ST_MAN	5800
9	60	IT_PROG	9000
10	60	IT_PROG	6000
11	60	IT_PROG	4200
12	80	SA_REP	11000
13	80	SA_REP	8600
14	80	SA_MAN	10500

...

19	110	AC_MGR	12000
20	(null)	SA_REP	7000

Add the salaries in the EMPLOYEES table for each job, grouped by department.

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	110	AC_ACCOUNT	8300
2	110	AC_MGR	12000
3	10	AD_ASST	4400
4	90	AD_PRES	24000
5	90	AD_VP	34000
6	60	IT_PROG	19200
7	20	MK_MAN	13000
8	20	MK_REP	6000
9	80	SA_MAN	10500
10	80	SA_REP	19600
11	(null)	SA_REP	7000
12	50	ST_CLERK	11700
13	50	ST_MAN	5800

Q: For every department those have department_id greater than 40, find the sum of the salaries of every job.

Using the GROUP BY Clause on Multiple Columns

For every department those have department_id greater than 40, find the sum of the salaries of every job.

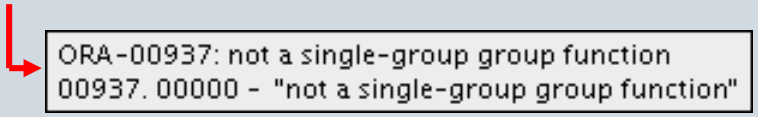
```
SELECT    department_id, job_id, SUM(salary)
FROM      employees
WHERE      department_id > 40
GROUP BY department_id, job_id
ORDER BY department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12000

Illegal Queries Using Group Functions

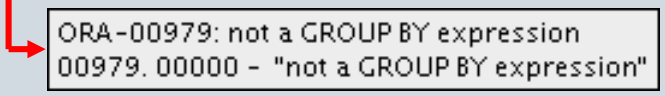
Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause:

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```



A GROUP BY clause must be added to count the last names for each department_id.

```
SELECT department_id, job_id, COUNT(last_name)
FROM   employees
GROUP BY department_id;
```



Either add job_id in the GROUP BY or remove the job_id column from the SELECT list.

Illegal Queries Using Group Functions

- You cannot use the `WHERE` clause to restrict groups.
- You use the `HAVING` clause to restrict groups.
- You cannot use group functions in the `WHERE` clause.

```
SELECT   department_id, AVG(salary)
FROM     employees
WHERE    AVG(salary) > 8000
GROUP BY department_id;
```

```
ORA-00934: group function is not allowed here
00934. 00000 - "group function is not allowed here"
*Cause:
*Action:
Error at Line: 3 Column: 9
```

Cannot use the `WHERE` clause to restrict groups

Restricting Group Results

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	80	8600
...		
18	110	8300
19	110	12000
20	(null)	7000

The maximum salary per department when it is greater than $10,000

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	90	24000
3	110	12000
4	80	11000

Restricting Group Results with the `HAVING` Clause

When you use the `HAVING` clause, the Oracle server restricts groups as follows:

1. Rows are grouped.
2. The group function is applied.
3. Groups matching the `HAVING` clause are displayed.

```
SELECT     column, group_function
FROM       table
[WHERE     condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```

Q: For every department, find the maximum salaries, if it is greater than 10.000.

Using the HAVING Clause

For every department, find the maximum salaries, if it is greater than 10.000.

```
SELECT    department_id, MAX(salary)
FROM      employees
GROUP BY department_id
HAVING    MAX(salary)>10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	90	24000
3	110	12000
4	80	11000

Q: List the jobs and the sum of the salary's of that job, if it is more than 1300$ and they are not any kind of representatives.

Using the HAVING Clause

List the jobs and the sum of the salary's of that job, if it is more than 1300$ and they are not any kind of representatives.

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

Nesting Group Functions

Display the maximum average salary:

```
SELECT   MAX(AVG(salary))
FROM     employees
GROUP BY department_id;
```

	MAX(AVG(SALARY))
1	19333.333333333333333333333333333333333

Q: Find names, job_id and salary of employees whose salary is equal to the minimum salary of department 50.

Using Group Functions in a Subquery

Find names, job_id and salary of employees whose salary is equal to the minimum salary of department 50.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees
                 WHERE department_id = 50);
```

2500

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

`HAVING` Clause with Subqueries

- The Oracle server executes the subqueries first.
- The Oracle server returns results into the `HAVING` clause of the main query.

Q: Find the department IDS and minimum salaries of those departments which have higher minimum salary than the department 50’s minimum salary.

HAVING Clause with Subqueries

- The Oracle server executes the subqueries first.
- The Oracle server returns results into the HAVING clause of the main query.

Find the department IDS and minimum salaries of those departments which have higher minimum salary than the department 50's minimum salary.

```
SELECT    department_id, MIN(salary)
FROM      employees
GROUP BY  department_id
HAVING    MIN(salary) >
                                    (SELECT MIN(salary)
                                     FROM   employees
                                     WHERE  department_id = 50);
```

2500

	DEPARTMENT_ID	MIN(SALARY)
1	(null)	7000
2	20	6000
3	90	17000
4	110	8300
5	80	8600
6	10	4400
7	60	4200

What Is Wrong with This Statement?

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
                (SELECT   MIN(salary)
                 FROM     employees
                 GROUP BY department_id);
```

```
ORA-01427: single-row subquery returns more than one row
01427. 00000 - "single-row subquery returns more than one row"
*Cause:
*Action:
```

Single-row operator with multiple-row subquery

Aggregate Functions with Relational Algebra:

COUNT, MAXIMUM, MINIMUM, AVERAGE are the aggregate functions that can be used with relational algebra.

<grouping attributes> $\mathfrak{I}$ **<function list> (<relation name>)**

- Retrieve each department number, number of employees in the department and their average salary.

R(DNO,#_OF_EMP,AVRG_SAL) $\leftarrow_{\text{DNO}} \mathfrak{I}_{\text{COUNT (SSN), AVERAGE(SALARY)}}$ (EMPLOYEE)

(Use COMPANY DATABASE)

List the names of all employees with two or more dependents.

$$T1(SSN,NO\text{-}OF\text{-}DEPTS) \leftarrow {}_{ESSN}\Im_{COUNT(DEPENDENT\_NAME)}(DEPENDENT)$$

$$T2 \leftarrow \delta_{NO\text{-}OF\text{-}DEPTS > 2}(T1)$$

$$RESULT \leftarrow \pi_{LNAME,FNAME}(T2 \infty_{SSN=ESSN} EMPLOYEE)$$

```
SELECT FNAME, LNAME
FROM   employee
WHERE  2 <= (SELECT   COUNT(*)
             FROM     dependent
             WHERE SSN = ESSN);
```

```
SELECT FNAME, LNAME, COUNT(*)
FROM   employee, dependent
WHERE SSN = ESSN
GROUP BY FNAME, LNAME
HAVING COUNT(*) >= 2;
```

Find the names of departments which has higher average salary than the overall average salary of the company.

$$A(DNO,AVGS) \leftarrow {}_{DNO}\Im_{AVG(SALARY)}(EMPLOYEE)$$

$$B(ALLAVGS) \leftarrow \Im_{AVG(SALARY)}(EMPLOYEE)$$

$$C \leftarrow A \times B$$

$$D \leftarrow \delta_{AVGS > ALLAVGS}(C)$$

$$RESULT \leftarrow \pi_{DNAME}(D \infty_{DNO=DNUMBER} DEPARTMENT)$$

List the employees and salaries those earn more than the average salary in their department.

$$A(DNUM,AVGS) \leftarrow {}_{DNO}\Im_{AVG(SALARY)}(EMPLOYEE)$$

$$B \leftarrow A \infty_{DNO=DNUM} EMPLOYEE$$

$$C \leftarrow \delta_{SALARY > AVGS}(B)$$

$$RESULT \leftarrow \pi_{FNAME, LNAME, SALARY, AVGS}(C)$$

SOLVE THE FOLLOWING QUERIES WITH SQL AND RELATIOANAL ALGEBRA BY USING COMPANY DATABASE:

1. Find the names of projects in which more than 5 employees works on.

2. Find the names of departments in which more than 2 employees working.

3. Find the names of departments those controls more than 10 projects.

4. Find the names of projects in which more than 3 employees of the research departments works on.

Retrieving Data by Using Subqueries

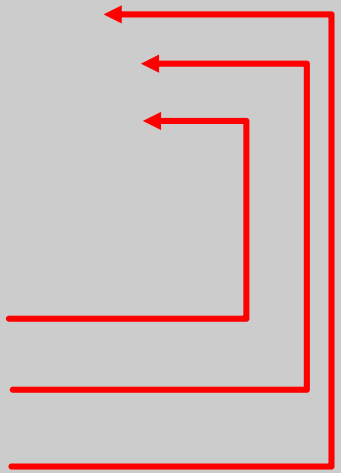
Multiple-Column Subqueries

Main query

```
WHERE (MANAGER_ID, DEPARTMENT_ID)  IN
```

Subquery

100	90
102	60
124	50



Each row of the main query is compared to values from a multiple-row and multiple-column subquery.

Column Comparisons

Multiple-column comparisons involving subqueries can be:

- Nonpairwise comparisons
- Pairwise comparisons

Pairwise Comparison Subquery

Display the details of the employees who are managed by the same manager and work in the same department as employees with the first name of “Bruce”

```
SELECT  employee_id, manager_id, department_id
FROM    employees
WHERE   (manager_id, department_id) IN
                    (SELECT manager_id, department_id
                     FROM employees
                     WHERE first_name = 'Bruce')
AND first_name <> 'Bruce';
```

Nonpairwise Comparison Subquery

Display the details of the employees who are managed by the same manager as the employees with the first name of “John” and work in the same department as the employees with the first name of “John.”

```
SELECT  employee_id, manager_id, department_id
FROM    employees
WHERE   manager_id IN
                (SELECT manager_id
                 FROM employees
                 WHERE first_name = 'John')
AND department_id IN
               (SELECT department_id
                FROM employees
                WHERE first_name = 'John')
AND first_name <> 'John';
```

Scalar Subquery Expressions

- A scalar subquery expression is a subquery that returns exactly one column value from one row.

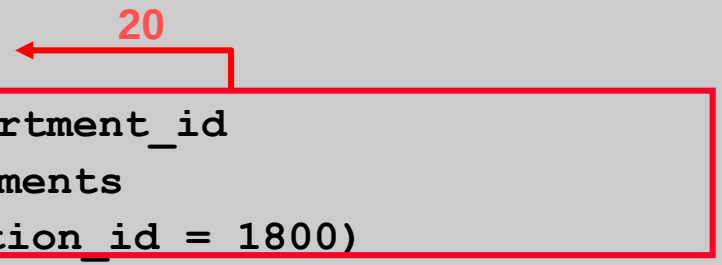
- Scalar subqueries can be used in:
  - The condition and expression part of `DECODE` and `CASE`
  - All clauses of `SELECT` except `GROUP BY`
  - The `SET` clause and `WHERE` clause of an `UPDATE` statement

Scalar Subqueries: Examples

- Scalar subqueries in `CASE` expressions:

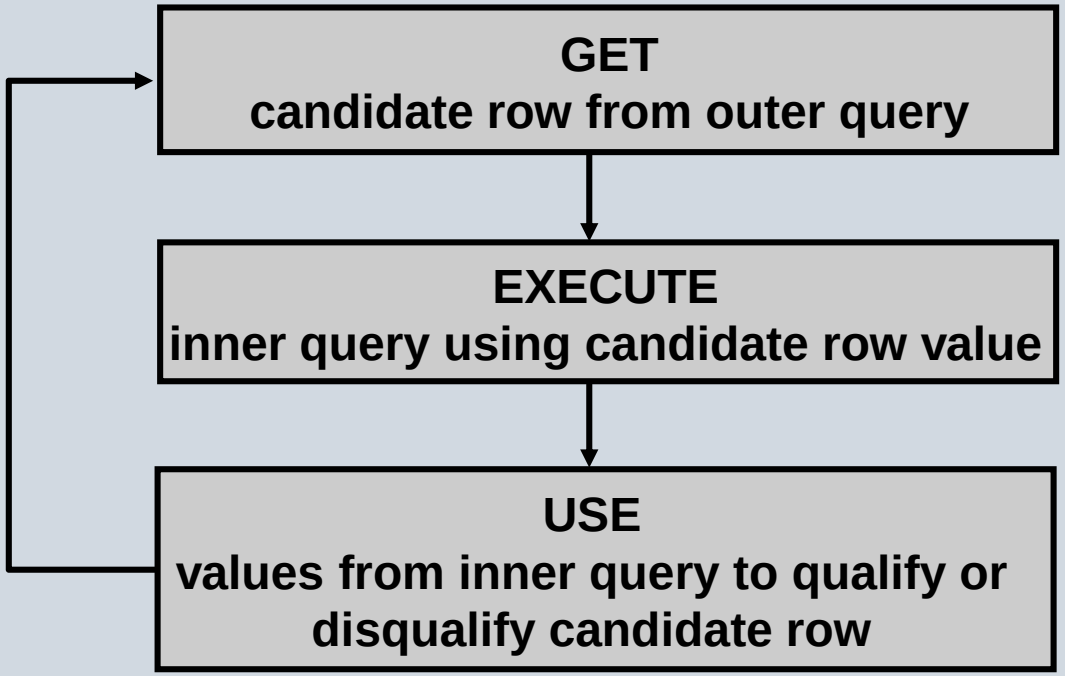
List the employee_id, last name and country information such that country is CANADA for location number 1800 and USA otherwise.

```
SELECT employee_id, last_name,
       (CASE
        WHEN department_id =
                (SELECT department_id
                 FROM departments
                 WHERE location_id = 1800)
         THEN 'Canada' ELSE 'USA' END) location
FROM   employees;
```



Correlated Subqueries

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.



Correlated Subqueries

The subquery references a column from a table in the parent query.

```
SELECT column1, column2, ...
FROM   table1   Outer_table
WHERE  column1 operator
                      (SELECT   column1, column2
                       FROM     table2
                       WHERE    expr1 =
                                Outer_table.expr2);
```

Using Correlated Subqueries

Q: Find all employees who earn more than the average salary in their department.

Using Correlated Subqueries

Find all employees who earn more than the average salary in their department.

```
SELECT last_name, salary, department_id
FROM   employees outer_table
WHERE  salary >
       (SELECT AVG(salary)
        FROM   employees inner_table
        WHERE inner_table.department_id =
        outer_table.department_id);
```

Each time a row from the outer query is processed, the inner query is evaluated.

Using Correlated Subqueries

Display details of those employees who have changed jobs at least twice.

```
SELECT e.employee_id, last_name,e.job_id
FROM   employees e
WHERE  2 <= (SELECT COUNT(*)
             FROM   job_history
             WHERE  employee_id = e.employee_id);
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID
1	200	Whalen	AD_ASST
2	101	Kochhar	AD_VP
3	176	Taylor	SA_REP

Using the `EXISTS` Operator

- The `EXISTS` operator tests for existence of rows in the results set of the subquery.
- If a subquery row value is found:
  - The search does not continue in the inner query
  - The condition is flagged `TRUE`
- If a subquery row value is not found:
  - The condition is flagged `FALSE`
  - The search continues in the inner query

Using the EXISTS Operator

Find the name's, job_id's and department_id's of the managers

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees outer
WHERE  EXISTS ( SELECT 'X'
                FROM   employees
                WHERE  manager_id =
                       outer.employee_id);
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	201	Hartstein	MK_MAN	20
2	205	Higgins	AC_MGR	110
3	100	King	AD_PRES	90
4	101	Kochhar	AD_VP	90
5	102	De Haan	AD_VP	90
6	103	Hunold	IT_PROG	60
7	108	Greenberg	FI_MGR	100
8	114	Raphaely	PU_MAN	30

Find all departments that do not have any employees.

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
                  FROM   employees
                  WHERE  department_id = d.department_id);
```

	DEPARTMENT_ID	DEPARTMENT_NAME
1	120	Treasury
2	130	Corporate Tax
3	140	Control And Credit
4	150	Shareholder Services
5	160	Benefits
6	170	Manufacturing
7	180	Construction

All Rows Fetched: 16

...

SOME EXAMPLES

- List the names of employees, their commission_pct and salaries and new salaries calculated as %10 rise if they have commission, %20 rise if they do not have commission. (use NVL2)

SOME EXAMPLES

- List the names of employees, their commission-pct and salaries and new salaries calculated as %10 rise if they have commission, %20 rise if they do not have commission. (use NVL2)

```
SELECT first_name, last_name, commission_pct, salary,
       NVL2(commission_pct, salary*1.1, salary*1.2) "new salary"
FROM employees;
```

- Find the names of departments which has higher average salary than the overall average salary of the company.

- Find the names of departments which has higher average salary than the overall average salary of the company.

```
SELECT          department_name, AVG(salary)
FROM            departments JOIN employees USING (department_id)
GROUP BY        department_name
HAVING          AVG(salary)    > (SELECT AVG(salary)
                                  FROM employees);
```

SOLVE THE FOLLOWING QUERIES BY USING COMPANY DATABASE

1. Find the sum of all salaries of all employees, the maximum salary, the minimum salary, and the average salary.
2. Find the sum, max, min, average of salaries of all employees who work for 'RESEARCH' department.
3. Retrieve the total number of employees in the company.
4. Retrieve the total number of employees working for ' RESEARCH ' department.
5. Count the number of distinct salary values in the database.
6. Retrieve the names of all employees who have two or more dependents.
7. For each department retrieve the department number, the number of employees in the department, their average salary.
8. For each project retrieve the project number, project name, and the number of employees who are working on that project.
9. For each project on which more than two employees work, retrieve the project number, project name and number of employees who are working on that project.
10. For each project retrieve the project number, project name and number of employees who are working on that project if the average salaries of employees working on that project is greater than 15000.
11. For each project retrieve the project number, project name, and number of employees from department 1 who are working on that project.
12. Find the employees and the projects that he work on, for those employees who earn more than $1200.

13. For each department, find the total number of employees whose salaries exceed $12000, but only for departments where more than one employees work.
14. For each department, find the number of employees, for those departments where number of worker is more than 1 and at least one of them is earning more than $12000.
15. For each department find the number of employees, if more than 5 employees' salaries are greater than $1200 in that department.
16. For each department find the number of employees earning more than $1200, if this number is more than 1 in that department.
17. Find the names of employees whose salary is greater than all the salaries of the 'PRODUCT_X' project employees.
18. Find the employees who worked for the projects totally more than 8 hours.
19. Find each department find dname and number of projects controlled by that department if it is greater than or equal to 2.
20. For each department find the dname, and number of projects controlled by that department if the number of location for that department is more than 1.
21. Find the names of projects which are located in any location that research department located in.
22. For each department find the number of projects.
23. Find the departments, which have greater average salary than "research" department average salary.
24. Find the names of employees whose salary greater than the maximum (highest) salary of the "research" department.