Database Management System

EXPERIMENT 8

NAME: Madhan RV

ROLL: 241801142

Let's add more employees with various job types and salaries for better aggregation demonstrations:

```
-- Add more employees with different job types
INSERT INTO employees VALUES
(121, 'Adam', 'Fripp', 'AFRIPP', '650.123.2234',
 TO_DATE('10-APR-1997', 'DD-MON-YYYY'), 'ST_MAN', 8200,
NULL, 100, 50);

INSERT INTO employees VALUES
(122, 'Payam', 'Kaufling', 'PKAUFLIN', '650.123.3234',
 TO_DATE('01-MAY-1995', 'DD-MON-YYYY'), 'ST_MAN', 7900,
NULL, 100, 50);

INSERT INTO employees VALUES
(123, 'Shanta', 'Vollman', 'SVOLLMAN', '650.123.4234',
 TO_DATE('10-OCT-1997', 'DD-MON-YYYY'), 'ST_MAN', 6500,
NULL, 100, 50);

INSERT INTO employees VALUES
(125, 'Julia', 'Nayer', 'JNAYER', '650.124.1214',
 TO_DATE('16-JUL-1997', 'DD-MON-YYYY'), 'ST_CLERK', 3200,
NULL, 120, 50);
```

```
INSERT INTO employees VALUES
(126, 'Irene', 'Mikkilineni', 'IMIKKILI', '650.124.1224',
 TO_DATE('28-SEP-1998', 'DD-MON-YYYY'), 'ST_CLERK', 2700,
NULL, 120, 50);

INSERT INTO employees VALUES
(127, 'James', 'Landry', 'JLANDRY', '650.124.1334',
 TO_DATE('14-JAN-1999', 'DD-MON-YYYY'), 'ST_CLERK', 2400,
NULL, 120, 50);

INSERT INTO employees VALUES
(128, 'Steven', 'Markle', 'SMARKLE', '650.124.1434',
 TO_DATE('08-MAR-2000', 'DD-MON-YYYY'), 'ST_CLERK', 2200,
NULL, 120, 50);

INSERT INTO employees VALUES
(129, 'Laura', 'Bissot', 'LBISSOT', '650.124.5234',
 TO_DATE('20-AUG-1997', 'DD-MON-YYYY'), 'ST_CLERK', 3300,
NULL, 121, 50);

INSERT INTO employees VALUES
(130, 'Mozhe', 'Atkinson', 'MATKINSO', '650.124.6234',
 TO_DATE('30-OCT-1997', 'DD-MON-YYYY'), 'ST_CLERK', 2800,
NULL, 121, 50);

INSERT INTO employees VALUES
(131, 'James', 'Marlow', 'JMARLOW', '650.124.7234',
 TO_DATE('16-FEB-1997', 'DD-MON-YYYY'), 'ST_CLERK', 2500,
NULL, 121, 50);

INSERT INTO employees VALUES
(132, 'TJ', 'Olson', 'TJOLSON', '650.124.8234',
 TO_DATE('10-APR-1999', 'DD-MON-YYYY'), 'ST_CLERK', 2100,
NULL, 121, 50);
```

```
INSERT INTO employees VALUES
(133, 'Jason', 'Mallin', 'JMALLIN', '650.127.1934',
 TO_DATE('14-JUN-1996', 'DD-MON-YYYY'), 'ST_CLERK', 3300,
NULL, 122, 50);

INSERT INTO employees VALUES
(134, 'Michael', 'Rogers', 'MROGERS', '650.127.1834',
 TO_DATE('26-AUG-1998', 'DD-MON-YYYY'), 'ST_CLERK', 2900,
NULL, 122, 50);

INSERT INTO employees VALUES
(135, 'Ki', 'Gee', 'KGEE', '650.127.1734',
 TO_DATE('12-DEC-1999', 'DD-MON-YYYY'), 'ST_CLERK', 2400,
NULL, 122, 50);

INSERT INTO employees VALUES
(136, 'Hazel', 'Philtanker', 'HPHILTAN', '650.127.1634',
 TO_DATE('06-FEB-2000', 'DD-MON-YYYY'), 'ST_CLERK', 2200,
NULL, 122, 50);

INSERT INTO employees VALUES
(137, 'Renske', 'Ladwig', 'RLADWIG', '650.121.1234',
 TO_DATE('14-JUL-1995', 'DD-MON-YYYY'), 'ST_CLERK', 3600,
NULL, 123, 50);

INSERT INTO employees VALUES
(138, 'Stephen', 'Stiles', 'SSTILES', '650.121.2034',
 TO_DATE('26-OCT-1997', 'DD-MON-YYYY'), 'ST_CLERK', 3200,
NULL, 123, 50);

INSERT INTO employees VALUES
(139, 'John', 'Seo', 'JSEO', '650.121.2019',
 TO_DATE('12-FEB-1998', 'DD-MON-YYYY'), 'ST_CLERK', 2700,
```

```
NULL, 123, 50);

INSERT INTO employees VALUES
(140, 'Joshua', 'Patel', 'JPATEL', '650.121.1834',
 TO_DATE('06-APR-1998', 'DD-MON-YYYY'), 'ST_CLERK', 2500,
NULL, 123, 50);

COMMIT;
```

Expected Output:

```
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row created.
Commit complete.
```

Exercise Solutions:

True/False Questions:

1. **Group functions work across many rows to produce one result per group.**
    a. **Answer: True**
2. **Group functions include nulls in calculations.**
    a. **Answer: False** (Group functions ignore NULL values)
3. **The WHERE clause restricts rows prior to inclusion in a group calculation.**
    a. **Answer: True**

4. Find highest, lowest, sum, and average salary of all employees

```
SELECT
    ROUND(MAX(salary)) AS Maximum,
    ROUND(MIN(salary)) AS Minimum,
    ROUND(SUM(salary)) AS Sum,
    ROUND(AVG(salary)) AS Average
FROM employees;
```

Expected Output:

```
  MAXIMUM    MINIMUM        SUM    AVERAGE
---------- ---------- ---------- ----------
     24000       2100     304600       6640
```

5. Display min, max, sum, and average salary for each job type

```
SELECT
    job_id AS job_type,
    ROUND(MIN(salary)) AS Minimum,
    ROUND(MAX(salary)) AS Maximum,
```

```
    ROUND(SUM(salary)) AS Sum,
    ROUND(AVG(salary)) AS Average
FROM employees
GROUP BY job_id
ORDER BY job_type;
```

Expected Output:

JOB_TYPE	MINIMUM	MAXIMUM	SUM	AVERAGE
AC_MGR	12000	12000	12000	12000
AD_ASST	4400	4400	4400	4400
AD_PRES	24000	24000	24000	24000
AD_VP	17000	17000	34000	17000
HR_REP	6500	6500	6500	6500
IT_PROG	4200	9000	19200	6400
MK_MAN	13000	13000	13000	13000
MK_REP	6000	6000	6000	6000
PR_REP	10000	10000	10000	10000
PU_CLERK	2500	3100	15900	2650
PU_MAN	11000	11000	11000	11000
SA_MAN	10500	10500	10500	10500
SA_REP	8600	11000	19600	9800
ST_CLERK	2100	3600	49400	2744
ST_MAN	6500	8200	28800	7200

6. Count people with the same job (with user prompt)

```
-- This will prompt for job type
SELECT job_id AS job_type, COUNT(*) AS Number_of_People
FROM employees
WHERE job_id = UPPER('&job_type')
```

```
GROUP BY job_id;
```

When prompted with 'ST_CLERK': Expected Output:

```
Enter value for job_type: ST_CLERK
old   3: WHERE job_id = UPPER('&job_type')
new   3: WHERE job_id = UPPER('ST_CLERK')

JOB_TYPE   NUMBER_OF_PEOPLE
---------- ----------------
ST_CLERK                18
```

7. Determine number of managers without listing them

```
SELECT COUNT(DISTINCT manager_id) AS "Number of Managers"
FROM employees
WHERE manager_id IS NOT NULL;
```

Expected Output:

```
Number of Managers
------------------
                13
```

8. Find difference between highest and lowest salaries

```
SELECT
    ROUND(MAX(salary) - MIN(salary)) AS DIFFERENCE
FROM employees;
```

Expected Output:

```
DIFFERENCE
----------
     21900
```

9. Display manager number and lowest-paid employee for that manager

```
SELECT
    manager_id AS Manager_Number,
    MIN(salary) AS Lowest_Salary
FROM employees
WHERE manager_id IS NOT NULL
GROUP BY manager_id
HAVING MIN(salary) > 6000
ORDER BY Lowest_Salary DESC;
```

Expected Output:

```
MANAGER_NUMBER LOWEST_SALARY
-------------- -------------
           100          6500
           149          8600
```

10. Display total employees and count by hire year

```
SELECT
    COUNT(*) AS Total_Employees,
    COUNT(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1995 THEN
1 END) AS Hired_1995,
    COUNT(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1996 THEN
1 END) AS Hired_1996,
    COUNT(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1997 THEN
1 END) AS Hired_1997,
    COUNT(CASE WHEN EXTRACT(YEAR FROM hire_date) = 1998 THEN
```

```
1 END) AS Hired_1998
FROM employees;
```

Expected Output:

```
TOTAL_EMPLOYEES HIRED_1995 HIRED_1996 HIRED_1997 HIRED_1998
--------------- ---------- ---------- ---------- ----------
             45          2          3          9          8
```

11. Matrix query to display job salaries by department

```
SELECT
    job_id AS job,
    SUM(CASE WHEN department_id = 20 THEN salary ELSE 0 END)
AS Salary_Dept_20,
    SUM(CASE WHEN department_id = 50 THEN salary ELSE 0 END)
AS Salary_Dept_50,
    SUM(CASE WHEN department_id = 80 THEN salary ELSE 0 END)
AS Salary_Dept_80,
    SUM(CASE WHEN department_id = 90 THEN salary ELSE 0 END)
AS Salary_Dept_90,
    SUM(salary) AS Total_Salary
FROM employees
WHERE department_id IN (20, 50, 80, 90)
GROUP BY job_id
ORDER BY job_id;
```

Expected Output:

```
JOB         SALARY_DEPT_20 SALARY_DEPT_50 SALARY_DEPT_80
SALARY_DEPT_90 TOTAL_SALARY
---------- -------------- -------------- -------------- ----
---------- ------------
```

```
MK_MAN              13000               0               0
0        13000
MK_REP               6000               0               0
0         6000
SA_MAN                  0               0           10500
0        10500
SA_REP                  0               0           19600
0        19600
ST_CLERK                0           49400               0
0        49400
ST_MAN                  0           28800               0
0        28800
                                                        Total:
117900
```

12. Display department summary with location, employee count, and average salary

```
SELECT
    d.dept_name || ' - ' || l.city AS "name-Location",
    COUNT(e.employee_id) AS "Number of people",
    ROUND(AVG(e.salary), 2) AS salary
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.department_id
LEFT JOIN location l ON d.location_id = l.location_id
GROUP BY d.dept_name, d.dept_id, l.city
ORDER BY d.dept_name;
```

Expected Output:

```
name-Location                          Number of people
SALARY
---------------------------------- ---------------- -----
```

```
-----
Accounting - Seattle                              1
12000
Administration - Seattle                          1
4400
Executive - Seattle                               3
17666.67
Finance - Seattle                                 0
Human Resources - Southlake                       1
6500
IT - South San Francisco                          3
6400
Marketing - Toronto                               2
9500
Public Relations - Southlake                      1
10000
Purchasing - Seattle                              6
2650
Sales - Oxford                                    3
9800
Shipping - South San Francisco                   20
3470
Treasury - Seattle                                0

12 rows selected.
```

Additional Group Function Examples:

13. Using multiple group functions together

```
SELECT
    department_id,
    COUNT(*) AS employee_count,
    AVG(salary) AS avg_salary,
```

```
    STDDEV(salary) AS salary_stddev,
    VARIANCE(salary) AS salary_variance
FROM employees
WHERE department_id IS NOT NULL
GROUP BY department_id
ORDER BY department_id;
```

Expected Output:

```
DEPARTMENT_ID EMPLOYEE_COUNT AVG_SALARY SALARY_STDDEV
SALARY_VARIANCE
------------- -------------- ---------- ------------- ------
---------
           10              1       4400             0
0
           20              2       9500          4949
24500000
           30              6       2650     284.2165
80779.3
           40              1       6500             0
0
           50             20       3470     1733.365
3004553
           60              3       6400     2450.000
6002500
           70              1      10000             0
0
           80              3  9800.0000     1202.772
1446666.7
           90              3  17666.6667    4041.452
16333333.3
          110              1      12000             0
0
```

14. GROUP BY with multiple columns

```
SELECT
    department_id,
    job_id,
    COUNT(*) AS employee_count,
    AVG(salary) AS avg_salary
FROM employees
WHERE department_id IS NOT NULL
GROUP BY department_id, job_id
ORDER BY department_id, job_id;
```

Expected Output:

```
DEPARTMENT_ID JOB_ID     EMPLOYEE_COUNT AVG_SALARY
------------- ---------- -------------- ----------
           10 AD_ASST                 1       4400
           20 MK_MAN                  1      13000
           20 MK_REP                  1       6000
           30 PU_CLERK                5       2650
           30 PU_MAN                  1      11000
           40 HR_REP                  1       6500
           50 ST_CLERK               18  2744.4444
           50 ST_MAN                  4       7200
           60 IT_PROG                 3       6400
           70 PR_REP                  1      10000
           80 SA_MAN                  1      10500
           80 SA_REP                  2       9800
           90 AD_PRES                 1      24000
           90 AD_VP                   2      17000
          110 AC_MGR                  1      12000

15 rows selected.
```

15. HAVING clause with complex conditions

```
SELECT
    department_id,
    COUNT(*) AS employee_count,
    AVG(salary) AS avg_salary,
    MAX(salary) AS max_salary
FROM employees
WHERE department_id IS NOT NULL
GROUP BY department_id
HAVING COUNT(*) > 2 AND AVG(salary) > 5000
ORDER BY avg_salary DESC;
```

Expected Output:

```
DEPARTMENT_ID EMPLOYEE_COUNT AVG_SALARY MAX_SALARY
------------- -------------- ---------- ----------
           90              3  17666.6667      24000
           80              3       9800      11000
           60              3       6400       9000
```

16. ROLLUP for subtotals

```
SELECT
    department_id,
    job_id,
    COUNT(*) AS employee_count,
    SUM(salary) AS total_salary
FROM employees
WHERE department_id IN (50, 80)
GROUP BY ROLLUP(department_id, job_id)
ORDER BY department_id, job_id;
```

Expected Output:

```
DEPARTMENT_ID JOB_ID     EMPLOYEE_COUNT TOTAL_SALARY
------------- ---------- -------------- ------------
           50 ST_CLERK               18        49400
           50 ST_MAN                  4        28800
           50                                  78200
           80 SA_MAN                  1        10500
           80 SA_REP                  2        19600
           80                                  30100
                                                     108300

7 rows selected.
```

17. CUBE for cross-tabulation

```
SELECT
    department_id,
    job_id,
    COUNT(*) AS employee_count
FROM employees
WHERE department_id IN (50, 80)
GROUP BY CUBE(department_id, job_id)
ORDER BY department_id, job_id;
```

Expected Output:

```
DEPARTMENT_ID JOB_ID     EMPLOYEE_COUNT
------------- ---------- --------------
           50 ST_CLERK               18
           50 ST_MAN                  4
           50                        22
           80 SA_MAN                  1
           80 SA_REP                  2
```

```
            80                          3
               SA_MAN                   1
               SA_REP                   2
               ST_CLERK                 18
               ST_MAN                   4
                                       25

11 rows selected.
```

18. GROUPING SETS for multiple grouping levels

```
SELECT
    department_id,
    job_id,
    COUNT(*) AS employee_count,
    AVG(salary) AS avg_salary
FROM employees
WHERE department_id IN (50, 80)
GROUP BY GROUPING SETS((department_id, job_id),
(department_id), ())
ORDER BY department_id, job_id;
```

Expected Output:

```
DEPARTMENT_ID JOB_ID     EMPLOYEE_COUNT AVG_SALARY
------------- ---------- -------------- ----------
           50 ST_CLERK                18  2744.4444
           50 ST_MAN                  4       7200
           50                         22       3555
           80 SA_MAN                  1      10500
           80 SA_REP                  2       9800
           80                          3  10033.3333
                                      25       4332
```

```
7 rows selected.
```

19. Nesting group functions

```
SELECT
    MAX(AVG(salary)) AS "Maximum Average Salary"
FROM employees
GROUP BY department_id;
```

Expected Output:

```
Maximum Average Salary
----------------------
          17666.6667
```

20. Statistical analysis with group functions

```
SELECT
    'All Employees' AS category,
    COUNT(*) AS count,
    ROUND(AVG(salary)) AS mean,
    ROUND(MEDIAN(salary)) AS median,
    ROUND(STDDEV(salary)) AS std_dev,
    MIN(salary) AS minimum,
    MAX(salary) AS maximum
FROM employees
UNION ALL
SELECT
    'With Commission' AS category,
    COUNT(*) AS count,
    ROUND(AVG(salary)) AS mean,
    ROUND(MEDIAN(salary)) AS median,
```

```
    ROUND(STDDEV(salary)) AS std_dev,
    MIN(salary) AS minimum,
    MAX(salary) AS maximum
FROM employees
WHERE commission_pct IS NOT NULL;
```

Expected Output:

```
CATEGORY            COUNT      MEAN    MEDIAN   STD_DEV
MINIMUM    MAXIMUM
--------------- ---------- ---------- ---------- ----------
---------- ----------
All Employees           45       6640       3100       5463
2100      24000
With Commission          3       9700       8600       1202
8600      11000
```

Summary of Group Functions Demonstrated:

1. **Aggregate Functions** - COUNT, SUM, AVG, MIN, MAX
2. **Statistical Functions** - STDDEV, VARIANCE, MEDIAN
3. **GROUP BY** - Grouping rows for aggregate calculations
4. **HAVING** - Filtering groups after aggregation
5. **DISTINCT with Aggregates** - COUNT(DISTINCT column)
6. **Nested Aggregates** - Aggregate functions within other aggregates
7. **Advanced Grouping** - ROLLUP, CUBE, GROUPING SETS
8. **Conditional Aggregation** - Using CASE within aggregate functions