

Practical 3: 23070521156 Sushmit Partakke

Part 4:

SQL Numeric Functions

SQL Numeric Functions are essential tools for performing **mathematical** and arithmetic operations on numeric data. These functions allow you to manipulate numbers, perform calculations, and **aggregate data** for reporting and analysis purposes.

Note: In this Lab Manual, examples are based on ORACLE SQLPLUS and MYSQL for your support but you are free to use any platform.

Numeric Functions in SQL*Plus (Oracle) and MySQL

Function	Description
ABS(n)	Returns the absolute value of n
ACOS(n)	Returns the arc cosine (inverse cosine) of n
ASIN(n)	Returns the arc sine (inverse sine) of n
ATAN(n)	Returns the arc tangent (inverse tangent) of n
ATN2(y, x) (SQL Server only)	Returns the angle whose tangent is the quotient of two arguments (Not available in Oracle/MySQL)
AVG(expression)	Returns the average of a set of values

CEILING(n)	Returns the smallest integer greater than or equal to n
-------------------	---

COUNT(expression)	Returns the number of rows matching a condition
COS(n)	Returns the cosine of n (in radians)
COT(n)	Returns the cotangent of n
DEGREES(n)	Converts radians to degrees
EXP(n)	Returns e raised to the power of n
FLOOR(n)	Returns the largest integer less than or equal to n
LOG(n)	Returns the natural logarithm (base e) of n
LOG10(n)	Returns the base-10 logarithm of n
MAX(expression)	Returns the maximum value in a column
MIN(expression)	Returns the minimum value in a column
PI()	Returns the value of π (pi)
POWER(x, y)	Returns x raised to the power of y
RADIANS(n)	Converts degrees to radians
RAND()	Returns a random number between 0 and 1
ROUND(n, d)	Rounds n to d decimal places
SIGN(n)	Returns -1 , 0 , or 1 depending on the sign of n

SIN(n)	Returns the sine of <code>n</code> (in radians)
SQRT(n)	Returns the square root of <code>n</code>
SQUARE(n)	Returns the square of <code>n</code> (same as <code>POWER(n, 2)</code>)
SUM(expression)	Returns the sum of a set of values
TAN(n)	Returns the tangent of <code>n</code> (in radians)

2. Examples in SQL*Plus (Oracle) /skip if you want to use mysql platform

2.1 Absolute Value (ABS)

```
SQL> SELECT ABS(-10) FROM dual;
          ABS(-10)
-----
          10
SELECT ABS (-10) FROM dual; -- Result: 10
```

2.2 Arc Cosine (ACOS)

```
SQL> SELECT ACOS(0.5) FROM dual;
          ACOS(0.5)
-----
        1.04719755
SELECT ACOS (0.5) FROM dual; -- Result: 1.04719755 (in radians)
```

2.3 Arc Sine (ASIN)

```
SQL> SELECT ASIN(0.15) FROM dual;  
  
ASIN(0.15)  
-----  
.150568273
```

SELECT ASIN(0.5) FROM dual; -- Result: 0.523598775 (in radians)

2.4 Arc Tangent (ATAN)

```
SQL> SELECT ATAN(0) FROM dual;  
  
ATAN(0)  
-----  
0
```

SELECT ATAN(1) FROM dual; -- Result: 0.785398163 (in radians)

2.5 Average (AVG)

```
SQL> SELECT AVG(salary) FROM employees;  
  
AVG(SALARY)  
-----  
43666.6667
```

SELECT AVG(salary) FROM employees; -- Finds the average salary

2.6 Ceiling (CEILING)

```
SQL> SELECT CEIL(4.2) FROM dual;  
  
CEIL(4.2)  
-----  
      5
```

SELECT CEIL(4.2) FROM dual; -- Result: 5

2.7 Count (COUNT)

```
SQL> SELECT COUNT(*) FROM employees WHERE post = 'Intern';  
  
COUNT(*)  
-----  
      1
```

SELECT COUNT(*) FROM employees WHERE department = 'Sales'; -- Counts employees in Sales

2.8 Cosine (COS)

```
SQL> SELECT COS(0) FROM dual;  
  
COS(0)  
-----  
      1
```

SELECT COS(0) FROM dual; -- Result: 1

2.9 Cotangent (COT)

```
SQL> SELECT 1/TAN(1/4) FROM dual;  
1/TAN(1/4)  
-----  
3.91631736
```

```
SELECT 1/TAN(PI()/4) FROM dual; -- Result: 1
```

2.10 Convert Radians to Degrees (DEGREES)

```
+-----+  
| DEGREES(PI()/2) |  
+-----+  
| 90 |  
+-----+
```

```
SELECT DEGREES(PI()/2) FROM dual; -- Result: 90
```

2.11 Exponential (EXP)

```
SQL> SELECT EXP(2) FROM dual;  
EXP(2)  
-----  
7.3890561
```

```
SELECT EXP(2) FROM dual; -- Result: 7.389056099
```

2.12 Floor (FLOOR)

```
SQL> SELECT FLOOR(4.8) FROM dual;  
  
FLOOR(4.8)  
-----  
        4
```

SELECT FLOOR(4.8) FROM dual; -- Result: 4

2.13 Natural Logarithm (LOG)

SELECT LOG(2.718281828) FROM dual; -- Result: 1 (since $e^1 = e$)

```
SELECT LOG(2.718281828) FROM dual; | LOG(2.718281828) |  
+-----+  
| 0.9999999998311266 |
```

2.14 Logarithm Base 10 (LOG10)

```
SQL> SELECT LOG(10, 100) FROM dual;  
  
LOG(10,100)  
-----  
        2
```

SELECT LOG(10, 100) FROM dual; -- Result: 2 (since $10^2 = 100$)

2.15 Maximum (MAX)

```
SQL> SELECT MAX(salary) FROM employees;  
  
MAX(SALARY)  
-----  
      101000
```

```
SELECT MAX(salary) FROM employees; -- Finds the highest salary
```

2.16 Minimum (MIN)

```
SQL> SELECT MIN(salary) FROM employees;  
  
MIN(SALARY)  
-----  
10000
```

```
SELECT MIN(salary) FROM employees; -- Finds the lowest salary
```

2.17 Pi (PI)

```
SQL> SELECT ACOS(-1) FROM dual;  
  
ACOS(-1)  
-----  
3.14159265
```

```
SELECT ACOS(-1) FROM dual; -- Result: 3.14159265
```

2.18 Power (POWER)

```
SQL> SELECT POWER(3, 2) FROM dual;  
  
POWER(3,2)  
-----  
9
```

```
SELECT POWER(3, 2) FROM dual; -- Result: 9
```

2.19 Convert Degrees to Radians (RADIAN)

```
SELECT RADIANS(180) FROM dual;
```

RADIANS(180)
3.141592653589793

```
SELECT RADIANS(180) FROM dual; -- Result: 3.14159265
```

2.20 Random Number (RAND)

```
SQL> SELECT DBMS_RANDOM.VALUE FROM dual;
```

VALUE
.626158473

```
SELECT DBMS_RANDOM.VALUE FROM dual; -- Returns a random number between 0 and 1
```

2.21 Round Number (ROUND)

```
SQL> SELECT ROUND(3.14159265, 2) FROM dual;
```

ROUND(3.14159265,2)
3.14

```
SELECT ROUND(3.14159265, 2) FROM dual; -- Result: 3.14
```

2.22 Sign of Number (SIGN)

```
SQL> SELECT SIGN(10) FROM dual;
```

SIGN(10)
1

```
SELECT SIGN(-10) FROM dual; -- Result: -1
SELECT SIGN(0) FROM dual; -- Result: 0
SELECT SIGN(10) FROM dual; -- Result: 1
```

2.23 Si

```
SQL> SELECT SIN(PI()/2) FROM dual;
```

SIN(PI()/2)
1

ne (SIN)

```
SELECT SIN(PI() /2) FROM dual; -- Result: 1
```

2.24 Square Root (SQRT)

```
SQL> SELECT SQRT(16) FROM dual;
```

SQRT(16)
4

```
SELECT SQRT(16) FROM dual; -- Result: 4
```

2.25 Square (SQUARE)

```
SQL> SELECT POWER(4, 2) FROM dual;
```

POWER(4,2)
16

```
SELECT POWER(4, 2) FROM dual; -- Result: 16
```

2.26 Sum (SUM)

```
SQL> SELECT SUM(salary) FROM employees;  
SUM(SALARY)  
-----  
131000
```

```
SELECT SUM(salary) FROM employees; -- Sum of all salaries
```

2.27 Tangent (TAN)

```
SELECT TAN(PI()/4) FROM dual; | TAN(PI()/4) |  
+-----+  
| 0.9999999999999999 |
```

```
SELECT TAN(PI() / 4) FROM dual; -- Result: 1
```

3. Examples in MySQL //SKIP IF DONE WITH ORACLE SQLPLUS

- ◊ The MySQL syntax is almost the same as Oracle, except for some functions.

```
SELECT ABS(-10); -- 10  
SELECT ACOS(0.5); -- 1.04719755  
SELECT ASIN(0.5); -- 0.523598775  
SELECT ATAN(1); -- 0.785398163  
SELECT AVG(salary) FROM employees;  
SELECT CEIL(4.2); -- 5  
SELECT COUNT(*) FROM employees WHERE department =  
'Sales';  
SELECT COS(0); -- 1  
SELECT COT(1); -- 0.6420926159  
SELECT DEGREES(PI() / 2); -- 90  
SELECT EXP(2); -- 7.389056099
```

```

SELECT FLOOR(4.8); -- 4
SELECT LOG(2.718281828); -- 1
SELECT LOG10(100); -- 2
SELECT MAX(salary) FROM employees;
SELECT MIN(salary) FROM employees;
SELECT PI(); -- 3.1415926535
SELECT POWER(3, 2); -- 9
SELECT RADIANS(180); -- 3.1415926535
SELECT RAND(); -- Random number
SELECT ROUND(3.14159265, 2); -- 3.14
SELECT SIGN(-10); -- -1
SELECT SIN(PI()/2); -- 1
SELECT SQRT(16); -- 4
SELECT SUM(salary) FROM employees;
SELECT TAN(PI()/4); -- 1

```

Advanced SQL Numeric Function Use Cases (Oracle & MySQL)

Following are **complex queries** using **numeric functions** in **real-world applications** for **financial analysis, scientific calculations, data analytics, and system performance monitoring**.

1 Financial Analytics: Compound Interest Calculation

Use Case: Calculate compound interest for a bank's customer accounts.

ACCOUNT_ID	PRINCIPAL	INTEREST_RATE	YEARS	FUTURE_VALUE
1	10000	5	3	11576.25
2	25000	4.5	5	31154.55
3	15000	6	4	18937.15
4	30000	3.8	7	38949.58

```
SELECT
account_id,
principal,
interest_rate,
years,
    ROUND(principal * POWER((1 + interest_rate / 100),
years), 2) AS future_value
FROM savings_accounts;
```

Formula Used:

$$FV = P \times (1 + r/n)^{nt}$$

Where:

- `principal`: Initial deposit
 - `interest_rate`: Annual interest rate
 - `years`: Time period
 - `POWER()` function computes exponentiation **Result Example:**

account_id	principal	interest_rate	years	future_value
101	1000	5	10	1628.89

2 Customer Segmentation: Standard Deviation of Purchases

Use Case: Find **customer spending variability** to create better promotions.

```
SELECT
customer_id,
    ROUND(AVG(purchase_amount), 2) AS avg_spend,
    ROUND(STDDEV(purchase_amount), 2) AS
spending_variability
FROM orders
GROUP BY customer_id
HAVING COUNT(*) > 5; -- Filter frequent customers
```

CUSTOMER_ID	AVG_SPEND	SPENDING_VARIABILITY
102	775.63	35.18
101	712.7	342.37

Key Insights:

- Uses `STDDEV()` to measure customer spending consistency.
Filters for customers with at least **5 purchases** (`HAVING COUNT(*) >`

-

5).

Result Example:

customer_id	avg_spend	spending_variability
201	250.00	50.25
305	500.00	120.75

3 Real-Time Performance Monitoring: CPU Load Analysis

Use Case: Compute CPU load trend for a cloud server system.

```
SQL> SELECT
  2      server_id,
  3      ROUND(AVG(cpu_usage), 2) AS avg_cpu,
  4      ROUND(MAX(cpu_usage), 2) AS max_cpu,
  5      ROUND(MIN(cpu_usage), 2) AS min_cpu,
  6      ROUND(SQRT(POWER(MAX(cpu_usage) - MIN(cpu_usage), 2)), 2) AS load_variance
  7  FROM server_logs
  8 WHERE log_date >= SYSDATE - INTERVAL '7' DAY -- Last 7 days
  9 GROUP BY server_id;
```

SERVER_ID	AVG_CPU	MAX_CPU	MIN_CPU	LOAD_VARIANCE
201	57.13	70.1	45.5	24.6
202	78.2	78.2	78.2	0
203	60.4	60.4	60.4	0

```
SELECT
server_id,
      ROUND (AVG (cpu_usage) , 2) AS avg_cpu,
      ROUND (MAX (cpu_usage) , 2) AS max_cpu,
      ROUND (MIN (cpu_usage) , 2) AS min_cpu,
      ROUND (SQRT (POWER (MAX (cpu_usage) - MIN (cpu_usage) ,
```

```
●  
2)), 2) AS load_variance FROM  
server_logs  
WHERE log_date >= SYSDATE - INTERVAL '7' DAY -- Last 7  
days  
GROUP BY server_id;
```

Key Metrics:

- `AVG()` to measure **average CPU usage**.
- `MAX()` & `MIN()` for **peak & lowest usage**.
- `SQRT(POWER())` to calculate variance in load.

Result Example:

server_id	avg_cpu	max_cpu	min_cpu	load_variance
A123	65.25	92.75	45.50	47.25
B456	40.10	75.00	20.20	54.80

4 Fraud Detection: Identifying Abnormal Transactions

Use Case: Detect transactions that are significantly **higher** than the usual customer behavior.

●

TRANSACTION_ID	CUSTOMER_ID	AMOUNT	AVG_AMOUNT	STDDEV_AMOUNT
1	101	500	600.3375	182.930284
2	102	1200.5	1275.65	106.278149
3	101	700.75	600.3375	182.930284
4	103	1500.25	1500.25	0
5	101	400	600.3375	182.930284
6	102	1350.8	1275.65	106.278149
7	101	800.6	600.3375	182.930284

7 rows selected.

```

SELECT
transaction_id,
customer_id,
amount,
    (SELECT AVG(amount) FROM transactions WHERE
customer_id = t.customer_id) AS avg_amount,
    (SELECT STDDEV(amount) FROM transactions WHERE
customer_id = t.customer_id) AS stddev_amount
FROM transactions t
WHERE amount > (SELECT AVG(amount) + 2 * STDDEV(amount)
FROM transactions WHERE customer_id = t.customer_id);

```

Logic:

Outlier transactions are those **greater than 2 standard deviations** from the average.

- Uses `AVG()` and `STDDEV()` per customer to personalize fraud detection.

Result Example:

- | transaction_id | customer_id | amount | avg_amount | stddev_amount |
|----------------|-------------|--------|------------|---------------|
| 90872 | 201 | 12000 | 3000 | 4500 |

- ◊ If a customer usually spends $\$3,000 \pm \$4,500$, a $\$12,000$ transaction is flagged as suspicious.

5 Logistics: Estimating Delivery Time Based on Distance

Use Case: Predict **delivery time (in hours)** for orders based on **distance and speed factors**.

```

SELECT
order_id,
distance_km,
    ROUND(distance_km / avg_speed, 2) AS
estimated_delivery_time
FROM (
    SELECT order_id, distance_km,
    CASE

```

```

        WHEN distance_km < 50 THEN 60 -- Urban: 60
km/h

        WHEN distance_km BETWEEN 50 AND 200 THEN 80
-- Suburban: 80 km/h

        ELSE 100 -- Highway: 100 km/h

    END AS avg_speed
FROM orders
) ;

```

Business Insight:

- Uses **speed categories** (`CASE`) to calculate **realistic delivery estimates**.
- Uses `ROUND()` to **format** the estimated time.

Result Example:

order_id	distance_km	estimated_delivery_time (hrs)
101	120	1.50
202	20	0.33

ORDER_ID	DISTANCE_KM	ESTIMATED_DELIVERY_TIME
1	30.5	.51
2	120.7	1.51
3	45.2	.75
4	250.8	2.51
5	90	1.13
6	15.3	.26
7	180.4	2.26
8	300.9	3.01

8 rows selected.

6 Astronomy/Physics: Calculating Earthquake Magnitude (Logarithmic Formula)

Use Case: Compute earthquake **Richter magnitude** based on **seismic wave amplitude**.

```
SELECT
  earthquake_id,
  station_id,
  amplitude,
  ROUND(LOG10(amplitude) + 3, 2) AS magnitude
FROM seismic_readings;
```

Richter Scale Formula:

$$M = \log_{10}(A) + 3$$

- Uses `LOG10 ()` to calculate **magnitude from amplitude**.

Result Example:

earthquake_id	station_id	amplitude	magnitude
EQ001	ST1001	5000	6.70

7 E-Commerce: Personalized Discount Calculation

Use Case: Apply dynamic **discount rates** based on **spending history**.

```

SELECT
customer_id,
total_spent,
CASE
    WHEN total_spent > 10000 THEN ROUND(total_spent *
0.15, 2)           WHEN total_spent BETWEEN 5000 AND
10000 THEN
ROUND(total_spent * 0.10, 2)
    ELSE ROUND(total_spent * 0.05, 2)
END AS discount
FROM (
    SELECT customer_id, SUM(order_value) AS total_spent
FROM orders GROUP BY customer_id
);

```

Discount Strategy:

- **15% off** for VIP customers (> \$10,000)
 - **10% off** for mid-level (\$5,000 - \$10,000)
 - **5% off** for casual shoppers (< \$5,000)
- Result Example:**

customer_id	total_spent	discount
101	12000	1800
202	7500	750