**Name.Surajit Halder**

**PRN.24070521509**

**SEM.iv(b2)**

**SEC.B**

**Pructicul 3:**

**Purt 4:**

**SǪL Numgric Functions**

SǪL **Numgric Functions** are essential tools for performing **muthgmuticul** and arithmetic operations on numeric data. These functions allow you to manipulate numbers, perform calculations, and **uggrggutg dutu** for reporting and analysis purposes.

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

# Numgric Functions in SǪL\*Plus (Oruclg) und MySǪL

|  |  |
| --- | --- |
| **Function** | **Dgscription** |
| **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) (SǪL**  **Sgrvgr only)** | Returns the angle whose tangent is the quotient of two arguments (Not available in Oracle/MySǪL) |
| **AVG(gxprgssion)** | Returns the average of a set of values |
| **CEILING(n)** | Returns the smallest integer greater than or equal to n |

|  |  |
| --- | --- |
| **COUNT(gxprgssion)** | 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 |
| **LOGfiO(n)** | Returns the base-10 logarithm of n |
| **MAX(gxprgssion)** | Returns the maximum value in a column |
| **MIN(gxprgssion)** | 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 n (in radians) |
| **SǪRT(n)** | Returns the square root of n |
| **SǪUARE(n)** | Returns the square of n (same as POWER(n, 2)) |

|  |  |
| --- | --- |
| **SUM(gxprgssion)** | Returns the sum of a set of values |
| **TAN(n)** | Returns the tangent of n (in radians) |

**2. Exumplgs in SǪL\*Plus (Oruclg) /skip if you wunt to usg mysql plutform**

1. **fi Absolutg Vulug (ABS)**

SELECT ABS(-10) FROM dual; -- Result: 10

* 1. **Arc Cosing (ACOS)**

SELECT ACOS(0.5) FROM dual; -- Result: 1.04719755 (in

radians)

* 1. **Arc Sing (ASIN)**

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

radians)

* 1. **Arc Tunggnt (ATAN)**

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

radians)

* 1. **Avgrugg (AVG)**

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

* 1. **Cgiling (CEILING)**

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

* 1. **Count (COUNT)**

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

* 1. **Cosing (COS)**

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

* 1. **Cotunggnt (COT)**

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

**2.fiO Convgrt Rudiuns to Dggrggs (DEGREES)**

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

**2.fifi Expongntiul (EXP)**

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

**2.fi2 Floor (FLOOR)**

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

**2.fi3 Nuturul Logurithm (LOG)**

SELECT LOG(2.718281828) FROM dual; -- Result: 1 (since

e^1 = e)

**2.fi4 Logurithm Busg fiO (LOGfiO)**

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **2.fi5 Muximum (MAX)** |  | | | | | |
| SELECT MAX(salary) salary | FROM | employees; | -- | Finds | the | highest |
| **2.fi6 Minimum (MIN)** |  |  |  |  |  |  |
| SELECT MIN(salary) salary | FROM | employees; | -- | Finds | the | lowest |
| **2.fi7 Pi (PI)** |  |  |  |  |  |  |

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

**2.fi8 Powgr (POWER)**

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

**2.fi9 Convgrt Dggrggs to Rudiuns (RADIANS)**

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

**2.2O Rundom Numbgr (RAND)**

SELECT DBMS\_RANDOM.VALUE FROM dual; -- Returns a random number between 0 and 1

**2.2fi Round Numbgr (ROUND)**

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

* 1. **Sign of Numbgr (SIGN)**

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

* 1. **Sing (SIN)**

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

* 1. **Squurg Root (SǪRT)**

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

|  |  |  |
| --- | --- | --- |
| **2.25 Squurg (SǪUARE)** |  | |
| SELECT POWER(4, 2)  **2.26 Sum (SUM)** | FROM | dual; -- Result: 16 |
| SELECT SUM(salary)  **2.27 Tunggnt (TAN)** | FROM | employees; -- Sum of all salaries |
| SELECT TAN(PI()/4) | FROM | dual; -- Result: 1 |

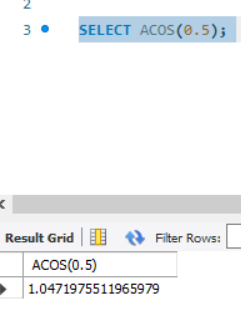
# 3. Exumplgs in MySǪL //SKIP IF DONE WITH ORACLE SǪLPLUS

🔹 The MySǪL syntax is almost the same as Oracle, except for some functions.

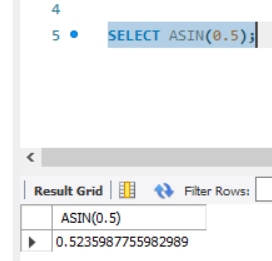
## SELECT ABS(-10); -- 10

## 

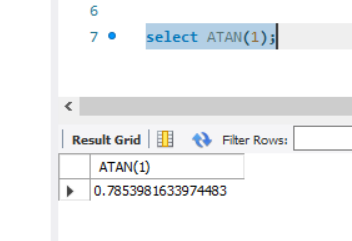
SELECT ACOS(0.5); -- 1.04719755



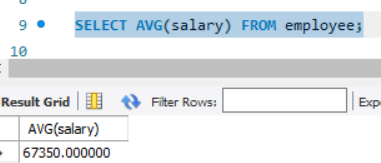
SELECT ASIN(0.5); -- 0.523598775



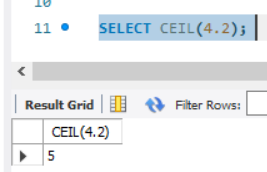
SELECT ATAN(1); -- 0.785398163



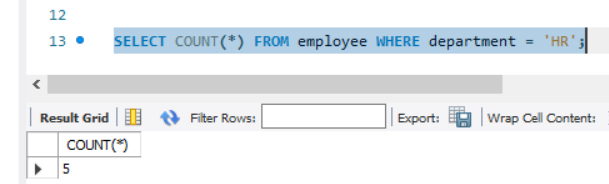
SELECT AVG(salary) FROM employees;



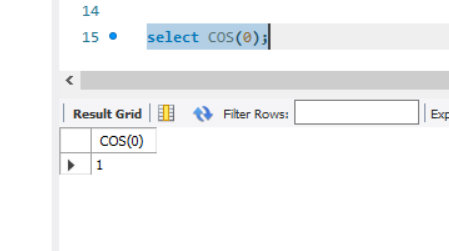
SELECT CEIL(4.2); -- 5



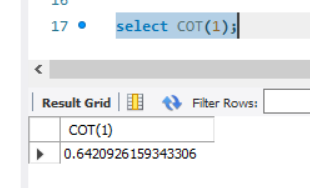
SELECT COUNT(\*) FROM employees WHERE department = 'Sales';



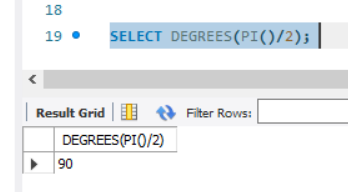
SELECT COS(0); -- 1



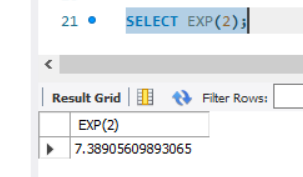
SELECT COT(1); -- 0.6420926159



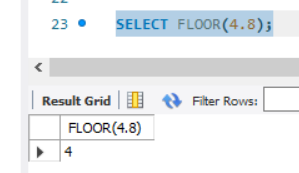
SELECT DEGREES(PI()/2); -- 90



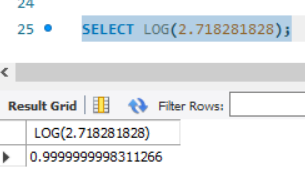
SELECT EXP(2); -- 7.389056099



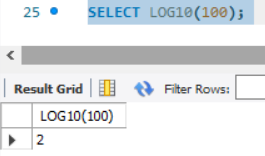
SELECT FLOOR(4.8); -- 4



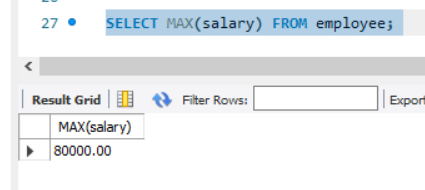
SELECT LOG(2.718281828); -- 1



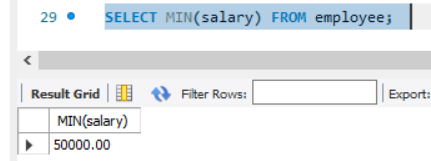
SELECT LOG10(100); -- 2



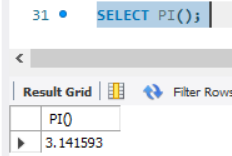
SELECT MAX(salary) FROM employees;



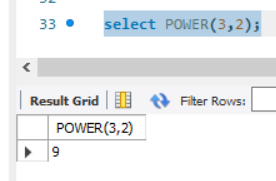
SELECT MIN(salary) FROM employees;



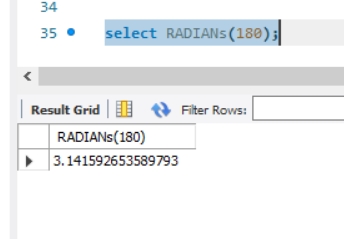
SELECT PI(); -- 3.1415926535



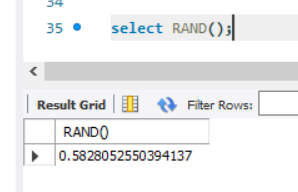
SELECT POWER(3, 2); -- 9



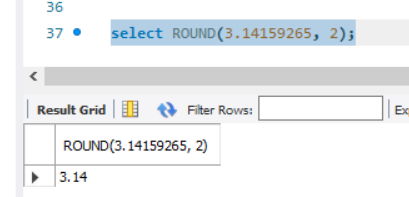
## SELECT RADIANS(180); -- 3.1415926535



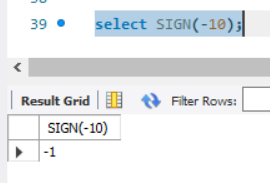
SELECT RAND(); -- Random number



SELECT ROUND(3.14159265, 2); -- 3.14



SELECT SIGN(-10); -- -1



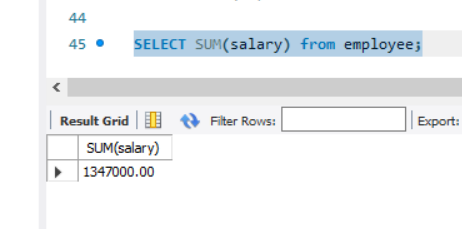
## SELECT SIN(PI()/2); -- 1

## 

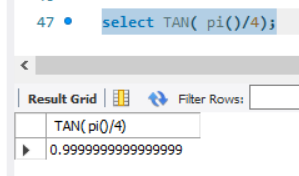
## ELECT SQRT(16); -- 4

## 

SELECT SUM(salary) FROM employees;



SELECT TAN(PI()/4); -- 1



# Advuncgd SǪL Numgric Function Usg Cusgs (Oruclg & MySǪL)

Fllowing are **complgx qugrigs** using **numgric functions** in **rgul-world upplicutions** for **finunciul unulysis, scigntific culculutions, dutu unulytics, und systgm pgrformuncg monitoring**.

# Finunciul Anulytics: Compound Intgrgst Culculution



**fi**

**Usg Cusg:** Calculate compound interest for a bank’s customer accounts.

## SELECT

account\_id, principal, interest\_rate, years,

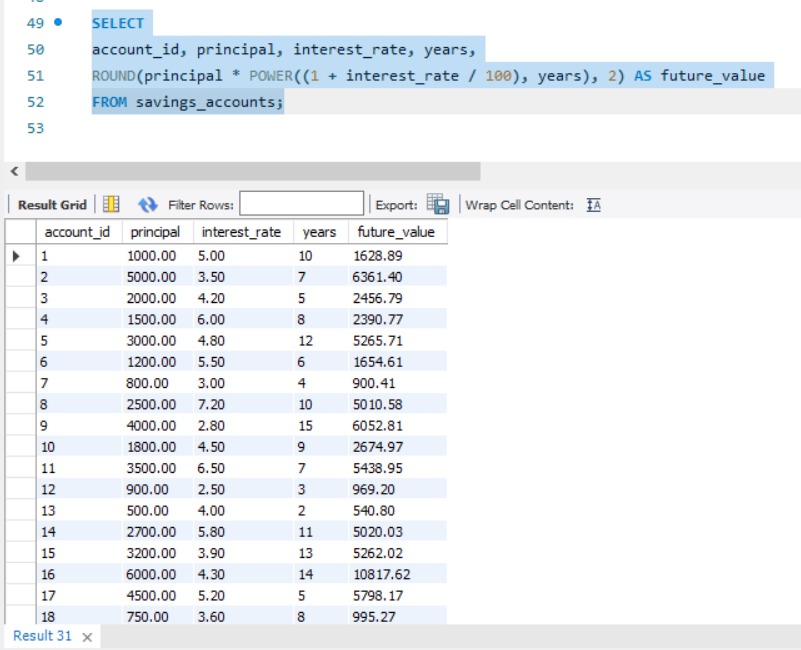
ROUND(principal \* POWER((1 + interest\_rate / 100), years), 2) AS future\_value

FROM savings\_accounts;

**Formulu Usgd:**

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

* principal: Initial deposit
* interest\_rate: Annual interest rate
* years: Time period
* POWER() function computes exponentiation



**Rgsult Exumplg:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **uccount\_id** | **principul** | **intgrgst\_rutg** | **ygurs** | **futurg\_vulug** |
| 101 | 1000 | 5 | 10 | 1628.89 |

# 2 Customgr Sggmgntution: Stundurd Dgviution of Purchusgs

**Usg Cusg:** Find **customgr spgnding vuriubility** 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

**Kgy Insights:**

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

**Rgsult Exumplg:**

|  |  |  |
| --- | --- | --- |
| **customgr\_id** | **uvg\_spgnd** | **spgnding\_vuriubility** |
| 201 | 250.00 | 50.25 |
| 305 | 500.00 | 120.75 |

# Rgul-Timg Pgrformuncg Monitoring: CPU Loud Anulysis



**3**

**Usg Cusg:** Compute **CPU loud trgnd** for a cloud server system.

## 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;

**Kgy Mgtrics:**

* AVG() to measure **uvgrugg CPU usugg**.
* MAX() & MIN() for **pguk & lowgst usugg**.
* SQRT(POWER()) to calculate variance in load.

**Rgsult Exumplg:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **sgrvgr\_id** | **uvg\_cpu** | **mux\_cpu** | **min\_cpu** | **loud\_vuriuncg** |
| A123 | 65.25 | 92.75 | 45.50 | 47.25 |
| B456 | 40.10 | 75.00 | 20.20 | 54.80 |

# Fruud Dgtgction: Idgntifying Abnormul Trunsuctions



**4**

**Usg Cusg:** Detect transactions that are significantly **highgr** than the usual customer behavior.

## 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:**

* **Outligr trunsuctions** are those **grgutgr thun 2 stundurd dgviutions**

from the average.

* Uses AVG() and STDDEV() **pgr customgr** to personalize fraud detection.

**Rgsult Exumplg:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **trunsuction\_i d** | **customgr\_i d** | **umoun t** | **uvg\_umou nt** | **stddgv\_umou nt** |
| 90872 | 201 | 12000 | 3000 | 4500 |

🔹 **If u customgr usuully spgnds 33,OOO ± 34,5OO, u 3fi2,OOO trunsuction is ﬂugggd us suspicious.**

# Logistics: Estimuting Dglivgry Timg Busgd on Distuncg



**5**

**Usg Cusg:** Predict **dglivgry timg (in hours)** for orders based on **distuncg und spggd fuctors**.

## SELECT

order\_id, distance\_km,

ROUND(distance\_km / avg\_speed, 2) AS estimated\_delivery\_time

## FROM (

SELECT order\_id, distance\_km, CASE

km/h

WHEN distance\_km < 50 THEN 60 -- Urban: 60

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

);

**Busingss Insight:**

* Uses **spggd cutggorigs** (CASE) to calculate **rgulistic dglivgry gstimutgs**.
* Uses ROUND() to **formut** the estimated time.

**Rgsult Exumplg:**

|  |  |  |
| --- | --- | --- |
| **ordgr\_id** | **distuncg\_km** | **gstimutgd\_dglivgry\_timg (hrs)** |
| 101 | 120 | 1.50 |
| 202 | 20 | 0.33 |

# 6 Astronomy/Physics: Culculuting Eurthquukg Mugnitudg (Logurithmic Formulu)

**Usg Cusg:** Compute earthquake **Richtgr mugnitudg** based on **sgismic wuvg umplitudg**.

## SELECT

earthquake\_id,

station\_id, amplitude,

ROUND(LOG10(amplitude) + 3, 2) AS magnitude FROM seismic\_readings;

**Richtgr Sculg Formulu:**

M=log10(A)+3M = \log\_{10} (A) + 3M=log10(A)+3

* Uses LOG10() to calculate **mugnitudg from umplitudg**.

**Rgsult Exumplg:**

|  |  |  |  |
| --- | --- | --- | --- |
| **gurthquukg\_id** | **stution\_id** | **umplitudg** | **mugnitudg** |
| EǪ001 | ST1001 | 5000 | 6.70 |

# E-Commgrcg: Pgrsonulizgd Discount Culculution



**7**

**Usg Cusg:** Apply dynamic **discount rutgs** based on **spgnding 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 Strutggy:**

* **fi5% off** for VIP customers (> $10,000)
* **fiO% off** for mid-level ($5,000 - $10,000)
* **5% off** for casual shoppers (< $5,000)

**Rgsult Exumplg:**

|  |  |  |
| --- | --- | --- |
| **customgr\_id** | **totul\_spgnt** | **discount** |
| 101 | 12000 | 1800 |
| 202 | 7500 | 750 |