Informix 14.10 Now Supports Common Table Expressions!

Tuesday , 26th May 2020
11:00 AM Eastern Daylight Time

Jinming Xiao
Informix SQL/Optimizer Engineer
HCL Software

IBM Cloud

© 2019 IBM Corporation

Disclaimers

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

© 2019 IBM Corporation

Common Table Expression Syntax

```
Syntax
WITH cte_name (Column1, Column2,.....ColumnN)
AS (CTE Definition) -- Write a query
        [,cte_name2 AS (CTE2 definition)…] – multiple/recursive CTEs
SELECT ……
FROM cte_name – Using CTE
```

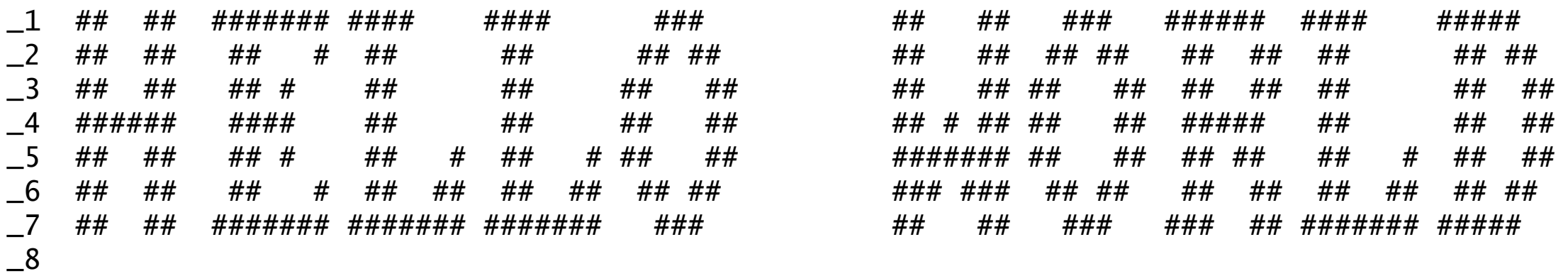
SELECT/INSERT/UPDATE/DELETE … reference CTE name

- CTE makes SQL more readable
- CTE provides programming feature to SQL

© 2019 IBM Corporation

HELLO WORLD – SQL Version

```
Database selected.
```



1 row(s) retrieved.

```
Database closed.
```

© 2019 IBM Corporation

SQL Version Banner with CTE

```
with fonts(c, c1, c2,c3,c4,c5,c6,c7,c8) as (
select * from table(multiset{ -- vga 8x8 fonts
ROW(' ', '0x00', '0x00', '0x00', '0x00', '0x00', '0x00', '0x00', '0x00'), ROW('!', '0x18', '0x3C', '0x3C', '0x18', '0x18', '0x00', '0x18', '0x00'),
ROW('A', '0x0C', '0x1E', '0x33', '0x33', '0x3F', '0x33', '0x33', '0x00'), ROW('B', '0x3F', '0x66', '0x66', '0x3E', '0x66', '0x66', '0x3F', '0x00'),
ROW('C', '0x3C', '0x66', '0x03', '0x03', '0x03', '0x66', '0x3C', '0x00'), ROW('D', '0x1F', '0x36', '0x66', '0x66', '0x66', '0x36', '0x1F', '0x00'),
ROW('E', '0x7F', '0x46', '0x16', '0x1E', '0x16', '0x46', '0x7F', '0x00'), ROW('F', '0x7F', '0x46', '0x16', '0x1E', '0x16', '0x06', '0x0F', '0x00'),
ROW('G', '0x3C', '0x66', '0x03', '0x03', '0x73', '0x66', '0x7C', '0x00'), ROW('H', '0x33', '0x33', '0x33', '0x3F', '0x33', '0x33', '0x33', '0x00'),
ROW('I', '0x1E', '0x0C', '0x0C', '0x0C', '0x0C', '0x0C', '0x1E', '0x00'), ROW('J', '0x78', '0x30', '0x30', '0x30', '0x33', '0x33', '0x1E', '0x00'),
ROW('K', '0x67', '0x66', '0x36', '0x1E', '0x36', '0x66', '0x67', '0x00'), ROW('L', '0x0F', '0x06', '0x06', '0x06', '0x46', '0x66', '0x7F', '0x00'),
ROW('M', '0x63', '0x77', '0x7F', '0x7F', '0x6B', '0x63', '0x63', '0x00'), ROW('N', '0x63', '0x67', '0x6F', '0x7B', '0x73', '0x63', '0x63', '0x00'),
ROW('O', '0x1C', '0x36', '0x63', '0x63', '0x63', '0x36', '0x1C', '0x00'), ROW('P', '0x3F', '0x66', '0x66', '0x3E', '0x06', '0x06', '0x0F', '0x00'),
ROW('Q', '0x1E', '0x33', '0x33', '0x33', '0x3B', '0x1E', '0x38', '0x00'), ROW('R', '0x3F', '0x66', '0x66', '0x3E', '0x36', '0x66', '0x67', '0x00'),
ROW('S', '0x1E', '0x33', '0x07', '0x0E', '0x38', '0x33', '0x1E', '0x00'), ROW('T', '0x3F', '0x2D', '0x0C', '0x0C', '0x0C', '0x0C', '0x1E', '0x00'),
ROW('U', '0x33', '0x33', '0x33', '0x33', '0x33', '0x33', '0x3F', '0x00'), ROW('V', '0x33', '0x33', '0x33', '0x33', '0x33', '0x1E', '0x0C', '0x00'),
ROW('W', '0x63', '0x63', '0x63', '0x6B', '0x7F', '0x77', '0x63', '0x00'), ROW('X', '0x63', '0x63', '0x36', '0x1C', '0x1C', '0x36', '0x63', '0x00'),
ROW('Y', '0x33', '0x33', '0x33', '0x1E', '0x0C', '0x0C', '0x1E', '0x00'), ROW('Z', '0x7F', '0x63', '0x31', '0x18', '0x4C', '0x66', '0x7F', '0x00'),
ROW('~', '0x6E', '0x3B', '0x00', '0x00', '0x00', '0x00', '0x00', '0x00')})),
text(str) as (select 'HELLO WORLD'), -- text string to display
xword(str, o, c, c1, c2,c3,c4,c5,c6,c7,c8) as ( -- get all fonts data for above string
    select str, 1, c, c1, c2,c3,c4,c5,c6,c7,c8 from text, fonts where c = substr(str, 1, 1)
    union all
    select str, o+1, c, c1, c2,c3,c4,c5,c6,c7,c8 from fonts, xword where fonts.c = substr(str, o+1, 1) and o < length(str)),
x(n, p) as (    select 1, 128 union all select n+1, p/2 from x where n < 8), -- for (p =128, n = 1; i <= 8; n++) p>>1;
show as (select  sum ((case when bitand(to_number(c1)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c1,
          -- pixel to '#' or ' '
        sum ((case when bitand(to_number(c2)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c2,
        sum ((case when bitand(to_number(c3)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c3,
        sum ((case when bitand(to_number(c4)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c4,
        sum ((case when bitand(to_number(c5)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c5,
        sum ((case when bitand(to_number(c6)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c6,
        sum ((case when bitand(to_number(c7)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c7,
        sum ((case when bitand(to_number(c8)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c8
          from x cross join xword group by xword.o order by xword.o desc)
select  sum(c1::lvarchar) as _1, sum(c2::lvarchar) as _2, sum(c3::lvarchar) as _3, sum(c4::lvarchar) as _4, -- display the result
        sum(c5::lvarchar) as _5, sum(c6::lvarchar) as _6, sum(c7::lvarchar) as _7, sum(c8::lvarchar) as _8 from show;
```

© 2019 IBM Corporation

Recursive CTE

A recursive query starts with either one non-recursive sub-query or several non-recursive sub-queries joined by UNION or UNION ALL, and ends with exactly one recursive sub-query joined by UNION ALL. A recursive sub-query references the CTE being defined.

An example of a recursive query computing the factorial of numbers from 0 to 9 is the following:

```
WITH temp (n, fact) AS (
  SELECT 0, 1                                -- Initial Subquery
  UNION ALL
  SELECT n+1, (n+1)*fact FROM temp -- Recursive Subquery
     WHERE n < 9)
SELECT * FROM temp;
```

C: for (n = 0, fact = 1; n < 9; n = n+1) { fact *= (n + 1); }

© 2019 IBM Corporation

Informix CTE supports

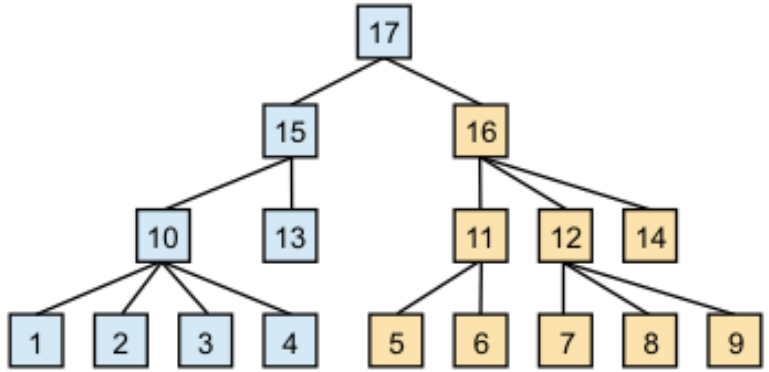
- Simple CTE
- Recursive CTE
- Multiple CTEs in one query
- CTE with Select, Insert, Update, Delete statements
- CTE in view
- CTE in SPL
- CTE in Trigger
- Recursive CTE CYCLE clause
- Other SQL changes
  - Select <expression> without ‘from table’

© 2019 IBM Corporation

RCTE Use Case – Tree/Hierarchical Dataset

```
CREATE TABLE employee(
                empid  INTEGER,
                name   CHAR(10),
                mgrid  INTEGER
);
INSERT INTO employee VALUES  ( 1, 'Jones',     10);")
INSERT INTO employee VALUES  ( 2, 'Hall',      10);")
INSERT INTO employee VALUES  ( 3, 'Kim',       10);")
INSERT INTO employee VALUES  ( 4, 'Lindsay',   10);")
INSERT INTO employee VALUES  ( 5, 'McKeough',  11);")
INSERT INTO employee VALUES  ( 6, 'Barnes',    11);")
……
INSERT INTO employee VALUES  (16, 'Goyal',     17);")
INSERT INTO employee VALUES  (17, 'Urbassek',  NULL);")
```

Figure 1. Relationships of Elements in a Reporting Hierarchy

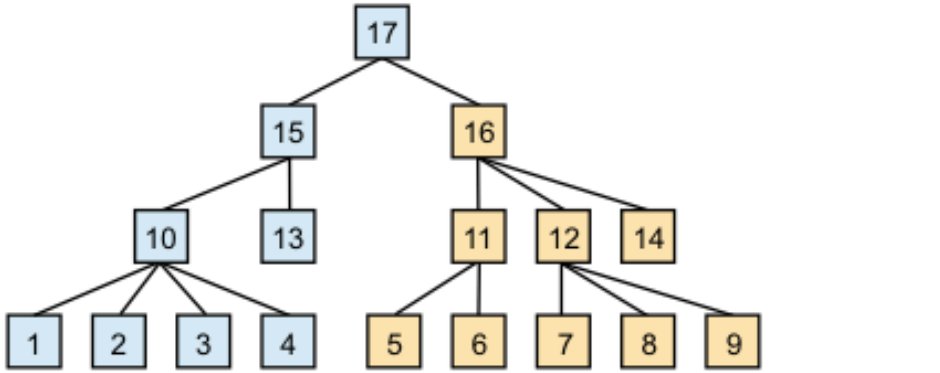


© 2019 IBM Corporation

RCTE Use Case – Tree/Hierarchical Dataset

```
WITH cte AS (
  SELECT empid, name, mgrid, 1 AS level
    FROM employee
    WHERE name = 'Goyal'
  UNION ALL
  SELECT t.empid, t.name, t.mgrid,
                  (cte.level + 1)
    FROM employee t, cte
  where t.mgrid = cte.empid
  )
SELECT * from cte ;
```

Figure 1. Relationships of Elements in a Reporting Hierarchy

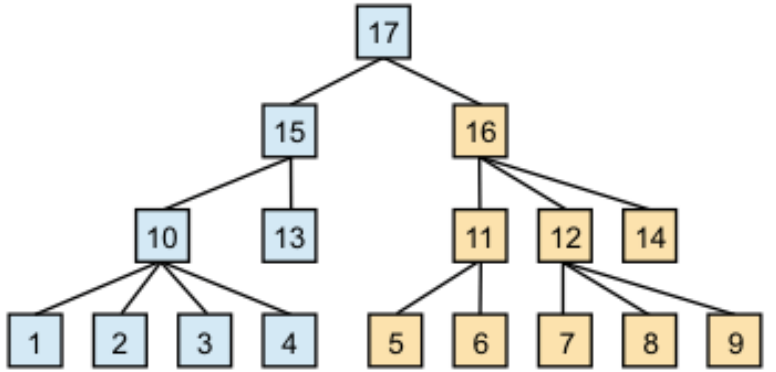


© 2019 IBM Corporation

RCTE Use Case – Tree/Hierarchical Dataset

empid	name	mgrid	level
16	Goyal	17	1
14	Scott	16	2
12	Henry	16	2
9	Shoeman	12	3
8	Smith	12	3
7	ONeil	12	3
11	Zander	16	2
6	Barnes	11	3
5	McKeough	11	3

Figure 1. Relationships of Elements in a Reporting Hierarchy



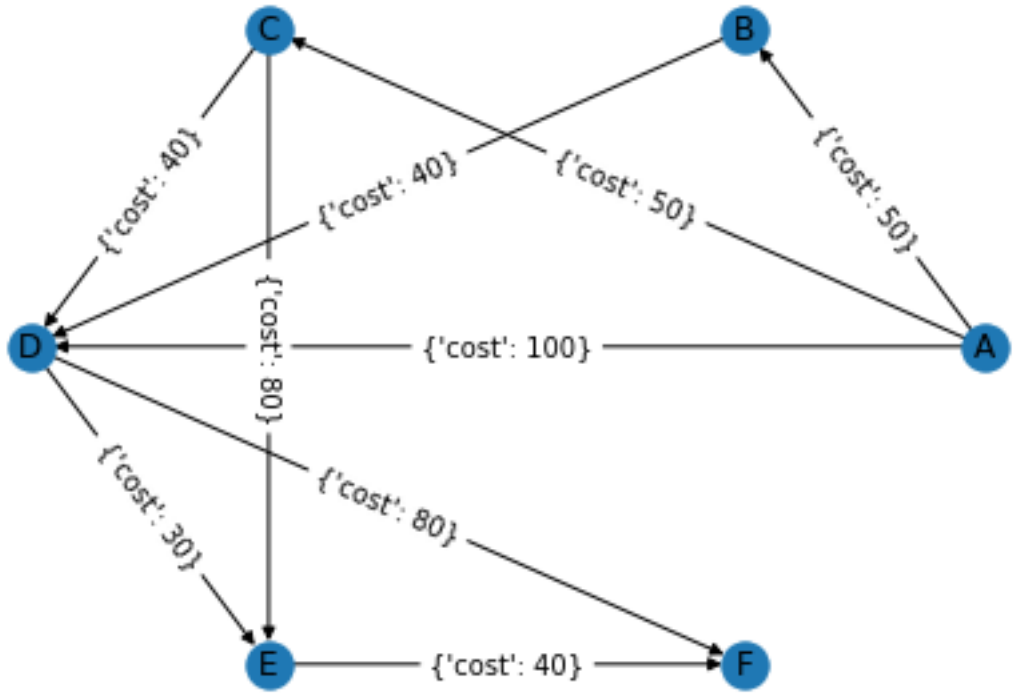
© 2019 IBM Corporation

RCTE Use Case – Graph Dataset

6 cities A, B, C, D, E, and F, with cost between cities.

```
create table loc (id int, name char(10));
insert into loc values (1, 'A');
insert into loc values (2, 'B');
insert into loc values (3, 'C');
insert into loc values (4, 'D');
insert into loc values (5, 'E');
insert into loc values (6, 'F');

create table road(start int, end int, cost int);
insert into road values (1, 2, 50);
insert into road values (1, 3, 50);
insert into road values (1, 4, 100);
insert into road values (2, 4, 40);
insert into road values (3, 4, 40);
insert into road values (3, 5, 80);
insert into road values (4, 5, 30);
insert into road values (4, 6, 80);
insert into road values (5, 6, 40);
```



© 2019 IBM Corporation

RCTE Use Case – Graph Dataset

```
-- list all paths from ‘A’ to ‘F’
with cte(id, path, cost, name, level) as (
   SELECT f.id, (trim(a.name) || " -> " || f.name)::lvarchar, r.cost, f.name, 1
      FROM loc a, road r, loc f
      WHERE a.name = 'A' AND r.start = a.id AND r.end = f.id
   UNION ALL
   SELECT f.id, trim(a.path) || " -> " || trim(f.name), a.cost + r.cost, f.name, level + 1
      FROM cte a, road r, loc f
      WHERE r.start = a.id AND r.end = f.id
)
select path::char(24) as path, cost, level from cte
where name = 'F’ order by cost;
```

© 2019 IBM Corporation

Use Case – Graph Dataset

path	cost	level
A -> B -> D -> E -> F	160	4
A -> C -> D -> E -> F	160	4
A -> D -> E -> F	170	3
A -> C -> D -> F	170	3
A -> B -> D -> F	170	3
A -> C -> E -> F	170	3
A -> D -> F	180	2

© 2019 IBM Corporation

Recursive CTE Cycle Detection

Recursive CTE Optional Cycle Clause

CYCLE <column list> SET <cycle pseudo column>

TO <value1> DEFAULT <value2>

The CYCLE option allows you to safeguard against cyclic data. Not only will it terminate repeating cycles but it also allows you to optionally output a cycle mark indicator that may lead you to find cyclic data.

© 2019 IBM Corporation

Recursive CTE Cycle Detection

```
create table cycle (id int, pid int);
insert into cycle values (1,2); insert into cycle values (2,1);
WITH cte AS (
        select id, pid from cycle where id = 1
    UNION ALL
        select t.id, t.pid from cycle t, cte where t.pid = cte.id)
cycle id set iscycle to "yes" default "no"
SELECT id, pid, iscycle from cte ;
         id         pid iscycle
          1           2 no
          2           1 yes
```

SELECT id, pid, CONNECT_BY_ISLEAF leaf, CONNECT_BY_ISCYCLE cycle FROM cycle
        START WITH id = 1 **CONNECT BY** nocycle PRIOR id = pid;

© 2019 IBM Corporation

Informix Recursive CTE Performance

- 12.10 Connect By 1,000,000 loops 75 seconds
- 14.10 Connect By 1,000,000 loops 35 seconds

```
select count(*) from (
SELECT LEVEL, MOD(LEVEL,2), MOD(LEVEL,4)
FROM sysmaster:sysdual CONNECT BY LEVEL <= 1000000)
-- because of extra join, cycle detection, and internal temp table writing operations
```

- 14.10 RCTE 1,000,000 loops 3 seconds

```
select count(*) from ( with cte(n)
as (select 1 as n
   union all select n+1 from cte where n < 1000000)
   select n, mod(n,2), mod(n,4) from cte);
```

© 2019 IBM Corporation

Informix Recursive CTE Limit

- Does not support multiple recursive subqueries in one RCTE
- Make sure type cast and enough space in the recursive subquery because of the recursive CTE columns are pre-defined from the non-recursive subquery

© 2019 IBM Corporation

CTE Fun Queries in Informix

- Helper function for string aggregation and dbaccess setup
  - We need a string aggregation helper function for fun queries

```
create function plus(s1 lvarchar, s2 lvarchar)
    returning lvarchar;
    return s1 || s2;
end function;
```

  - Set environment variable DBACCESS_COLUMNS=180
- Mandelbrot Set
- Julia Set – a beautiful tree like image
- SQL Version Banner program

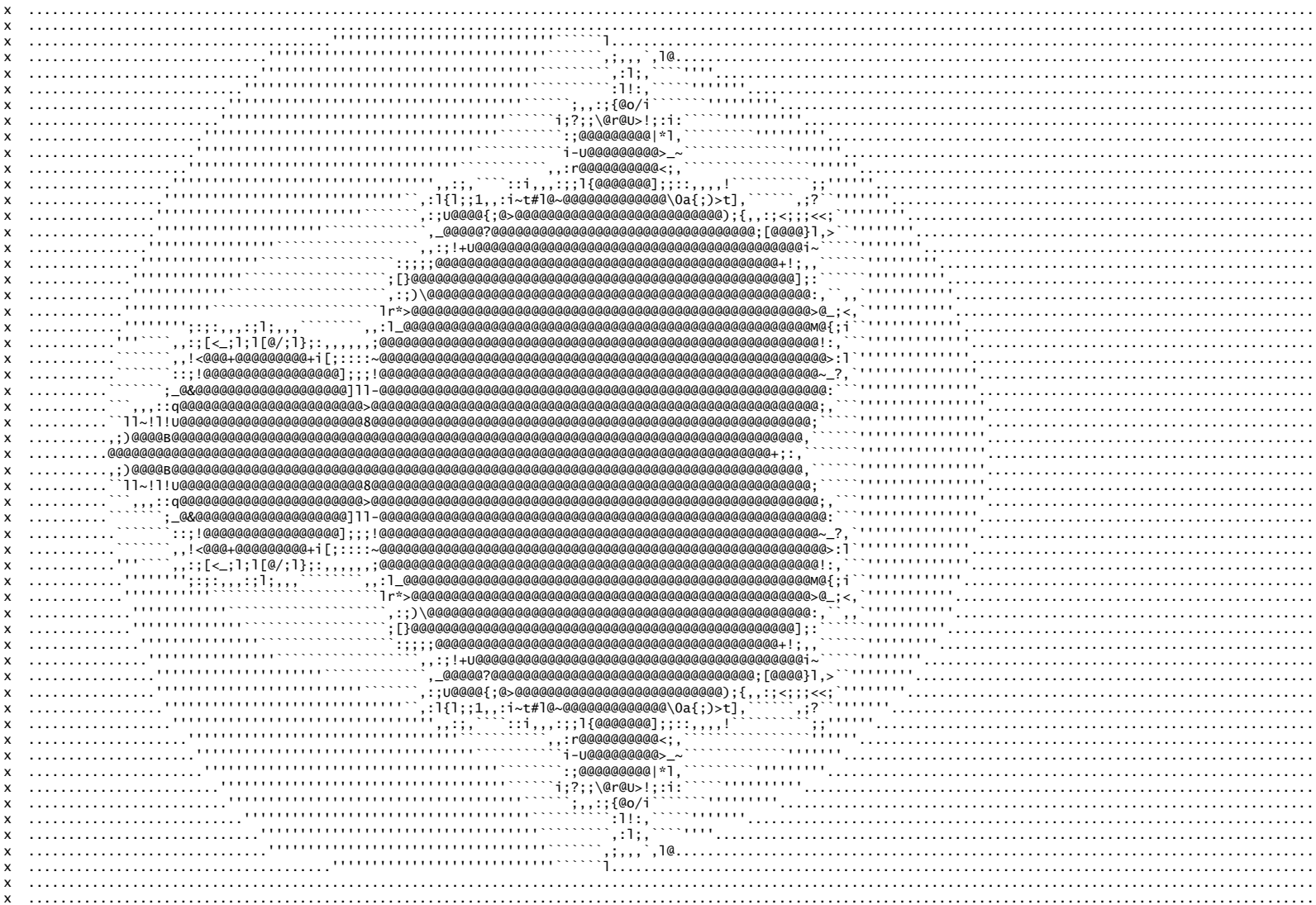
© 2019 IBM Corporation

CTE Fun Queries in Informix – Mandelbrot Set

```
WITH
x(n) AS ( select -70 UNION ALL select n+1 from x where n < 40),
y(j) AS ( select -30 UNION ALL select j+1 from y where j < 30),
q (r, i, rx, ix, g) AS (
    SELECT  n::float * 0.02, j::float * 0.04, 0::float, 0::float, 0
        FROM (select n from x) cross join (select j from y)
    UNION ALL
    SELECT  r, i,
        CASE WHEN ABS(rx * rx + ix * ix) <= 2 THEN rx * rx - ix * ix END + r,
        CASE WHEN ABS(rx * rx + ix * ix) <= 2 THEN 2 * rx * ix END + i,
        g + 1 FROM q WHERE rx IS NOT NULL AND g < 100),
Zt (i, r, x) AS (
    SELECT i, r, SUBSTR('@@B%8&WM#*oaqwmOOUcrt/\|()1{}[]?-_+~<>i!l;;:,```.',
        50 - (MAX(g)/2)::int, 1) as x
        FROM q GROUP BY i, r order by r desc)
SELECT  sum(x::lvarchar) as x from Zt group by i order by i;
-- port from PostgreSQL https://explainextended.com/2013/12/31/happy-new-year-5/
-- dbaccess cte Mandelbrot.sql | grep '^x'
```

© 2019 IBM Corporation

CTE Fun Queries in Informix – Mandelbrot Set



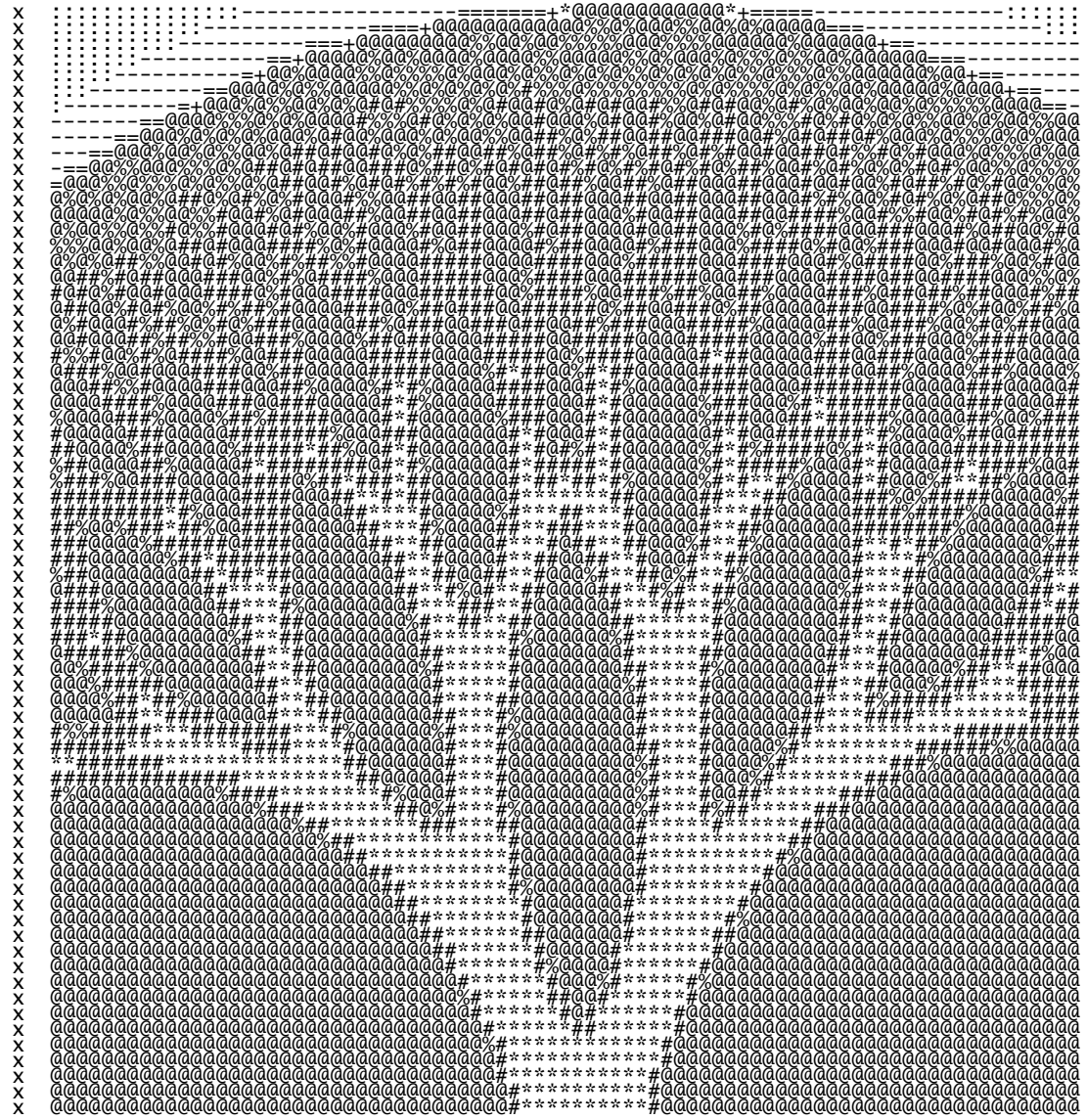
© 2019 IBM Corporation

CTE Fun Queries in Informix – Julia Set

```
WITH
x(n) AS (
    select -40 UNION ALL
    select n+1 from x where n < 40),
q (r, i, rx, ix, g) AS (
    SELECT  x + r::float * step, y + i::float * step,
            x + r::float * step, y + i::float * step, 0 FROM    (
        SELECT  0.25 x, -0.55 y, 0.002 step, r, I FROM (select n as r from x)
        cross join (select n as i from x))
        UNION ALL
        SELECT  r, i,
                CASE WHEN (rx * rx + ix * ix) < 1E8 THEN
                pow((rx * rx + ix * ix), 0.75) * COS(1.5 * ATAN2(ix, rx)) END - 0.2,
                CASE WHEN (rx * rx + ix * ix) < 1E8 THEN
                pow((rx * rx + ix * ix), 0.75) * SIN(1.5 * ATAN2(ix, rx)) END,
                g + 1
        FROM    q
        WHERE   rx IS NOT NULL AND g < 99),
Zt (i, r, x) AS (
    SELECT i, r, SUBSTR(" .:-=+*#%@", (MAX(g) /10 +1)::int, 1) as x
    FROM q GROUP BY i, r order by r)
SELECT  sum(x::lvarchar) as x from Zt group by i order by i;
-- port from PostgreSQL https://explainextended.com/2013/12/31/happy-new-year-5/
-- dbaccess cte Julia.sql | grep '^x'
```

© 2019 IBM Corporation

CTE Fun Queries in Informix – Julia Set



© 2019 IBM Corporation

SQL Version Banner with CTE

```
with fonts(c, c1, c2,c3,c4,c5,c6,c7,c8) as (
select * from table(multiset{ -- vga 8x8 fonts
ROW(' ', '0x00', '0x00', '0x00', '0x00', '0x00', '0x00', '0x00', '0x00'), ROW('!', '0x18', '0x3C', '0x3C', '0x18', '0x18', '0x00', '0x18', '0x00'),
ROW('A', '0x0C', '0x1E', '0x33', '0x33', '0x3F', '0x33', '0x33', '0x00'), ROW('B', '0x3F', '0x66', '0x66', '0x3E', '0x66', '0x66', '0x3F', '0x00'),
ROW('C', '0x3C', '0x66', '0x03', '0x03', '0x03', '0x66', '0x3C', '0x00'), ROW('D', '0x1F', '0x36', '0x66', '0x66', '0x66', '0x36', '0x1F', '0x00'),
ROW('E', '0x7F', '0x46', '0x16', '0x1E', '0x16', '0x46', '0x7F', '0x00'), ROW('F', '0x7F', '0x46', '0x16', '0x1E', '0x16', '0x06', '0x0F', '0x00'),
ROW('G', '0x3C', '0x66', '0x03', '0x03', '0x73', '0x66', '0x7C', '0x00'), ROW('H', '0x33', '0x33', '0x33', '0x3F', '0x33', '0x33', '0x33', '0x00'),
ROW('I', '0x1E', '0x0C', '0x0C', '0x0C', '0x0C', '0x0C', '0x1E', '0x00'), ROW('J', '0x78', '0x30', '0x30', '0x30', '0x33', '0x33', '0x1E', '0x00'),
ROW('K', '0x67', '0x66', '0x36', '0x1E', '0x36', '0x66', '0x67', '0x00'), ROW('L', '0x0F', '0x06', '0x06', '0x06', '0x46', '0x66', '0x7F', '0x00'),
ROW('M', '0x63', '0x77', '0x7F', '0x7F', '0x6B', '0x63', '0x63', '0x00'), ROW('N', '0x63', '0x67', '0x6F', '0x7B', '0x73', '0x63', '0x63', '0x00'),
ROW('O', '0x1C', '0x36', '0x63', '0x63', '0x63', '0x36', '0x1C', '0x00'), ROW('P', '0x3F', '0x66', '0x66', '0x3E', '0x06', '0x06', '0x0F', '0x00'),
ROW('Q', '0x1E', '0x33', '0x33', '0x33', '0x3B', '0x1E', '0x38', '0x00'), ROW('R', '0x3F', '0x66', '0x66', '0x3E', '0x36', '0x66', '0x67', '0x00'),
ROW('S', '0x1E', '0x33', '0x07', '0x0E', '0x38', '0x33', '0x1E', '0x00'), ROW('T', '0x3F', '0x2D', '0x0C', '0x0C', '0x0C', '0x0C', '0x1E', '0x00'),
ROW('U', '0x33', '0x33', '0x33', '0x33', '0x33', '0x33', '0x3F', '0x00'), ROW('V', '0x33', '0x33', '0x33', '0x33', '0x33', '0x1E', '0x0C', '0x00'),
ROW('W', '0x63', '0x63', '0x63', '0x6B', '0x7F', '0x77', '0x63', '0x00'), ROW('X', '0x63', '0x63', '0x36', '0x1C', '0x1C', '0x36', '0x63', '0x00'),
ROW('Y', '0x33', '0x33', '0x33', '0x1E', '0x0C', '0x0C', '0x1E', '0x00'), ROW('Z', '0x7F', '0x63', '0x31', '0x18', '0x4C', '0x66', '0x7F', '0x00'),
ROW('~', '0x6E', '0x3B', '0x00', '0x00', '0x00', '0x00', '0x00', '0x00')})),
text(str) as (select 'THANK YOU'), -- text string to display
xword(str, o, c, c1, c2,c3,c4,c5,c6,c7,c8) as ( -- get all fonts data for above string
    select str, 1, c, c1, c2,c3,c4,c5,c6,c7,c8 from text, fonts where c = substr(str, 1, 1)
    union all
    select str, o+1, c, c1, c2,c3,c4,c5,c6,c7,c8 from fonts, xword where fonts.c = substr(str, o+1, 1) and o < length(str)),
x(n, p) as (    select 1, 128 union all select n+1, p/2 from x where n < 8), -- for (p =128, n = 1; i <= 8; n++) p>>1;
show as (select  sum ((case when bitand(to_number(c1)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c1,
          -- pixel to '#' or ' '
        sum ((case when bitand(to_number(c2)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c2,
        sum ((case when bitand(to_number(c3)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c3,
        sum ((case when bitand(to_number(c4)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c4,
        sum ((case when bitand(to_number(c5)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c5,
        sum ((case when bitand(to_number(c6)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c6,
        sum ((case when bitand(to_number(c7)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c7,
        sum ((case when bitand(to_number(c8)::int, p) > 0 then '#' else ' ' end)::lvarchar)::char(8) as c8
          from x cross join xword group by xword.o order by xword.o desc)
select  sum(c1::lvarchar) as _1, sum(c2::lvarchar) as _2, sum(c3::lvarchar) as _3, sum(c4::lvarchar) as _4, -- display the result
        sum(c5::lvarchar) as _5, sum(c6::lvarchar) as _6, sum(c7::lvarchar) as _7, sum(c8::lvarchar) as _8 from show;
```

© 2019 IBM Corporation

Thank You

```
Database selected.

_1  ######  ##  ##    ##    ##   ## ###  ##          ##  ##    ###   ##  ##
_2  # ## #  ##  ##   ####   ###  ##  ##  ##          ##  ##   ## ##  ##  ##
_3    ##    ##  ##  ##  ##  #### ##  ## ##          ##  ##  ##   ## ##  ##
_4    ##    ######  ##  ##  ## ####  ####            ####   ##   ## ##  ##
_5    ##    ##  ##  ######  ##  ###  ## ##            ##    ##   ## ##  ##
_6    ##    ##  ##  ##  ##  ##   ##  ##  ##           ##     ## ##  ##  ##
_7   ####   ##  ##  ##  ##  ##   ## ###  ##          ####     ###   ######
_8

1 row(s) retrieved.

Database closed.
```

© 2019 IBM Corporation

Questions

© 2019 IBM Corporation