Recursive SQL

Understanding or Writing recursive query is one of the typical task in SQL. As part of this white paper, I would like to explain the concept of recursive query in DB2, the way it works and how to write them. This white paper typically caters to those who want to learn to play with recursive SQL in DB2.

Contents

Author

Venkata Rama Rajesh

DB2 DBA for Z/OS

vmallina@in.ibm.com

1. What is / Why recursive SQL ?

A recursive SQL is a type of SQL query that handles hierarchical model data in RDMS. Since recursive SQL used to handle hierarchical model data, this type of SQL is also called as hierarchical SQL.

Example 1 : Consider a table called EMP as shown in right side. That table contains employee name and his/her manager name. Suppose, our requirement is to obtain the list of all people who report to Bala, then the following simple query might help.

```
SELECT EMPLOYEE, MANAGER
FROM EMP
WHERE MANAGER = 'Bala'
```

Result

EMPLOYEE	MANAGER
Chandhu	Bala
Chaitanya	Bala

Suppose, our requirement is to obtain all people under manager Bala either directly (or) indirectly as show by the below example.

→ Chandu and Chaitanya (reporting to Bala)

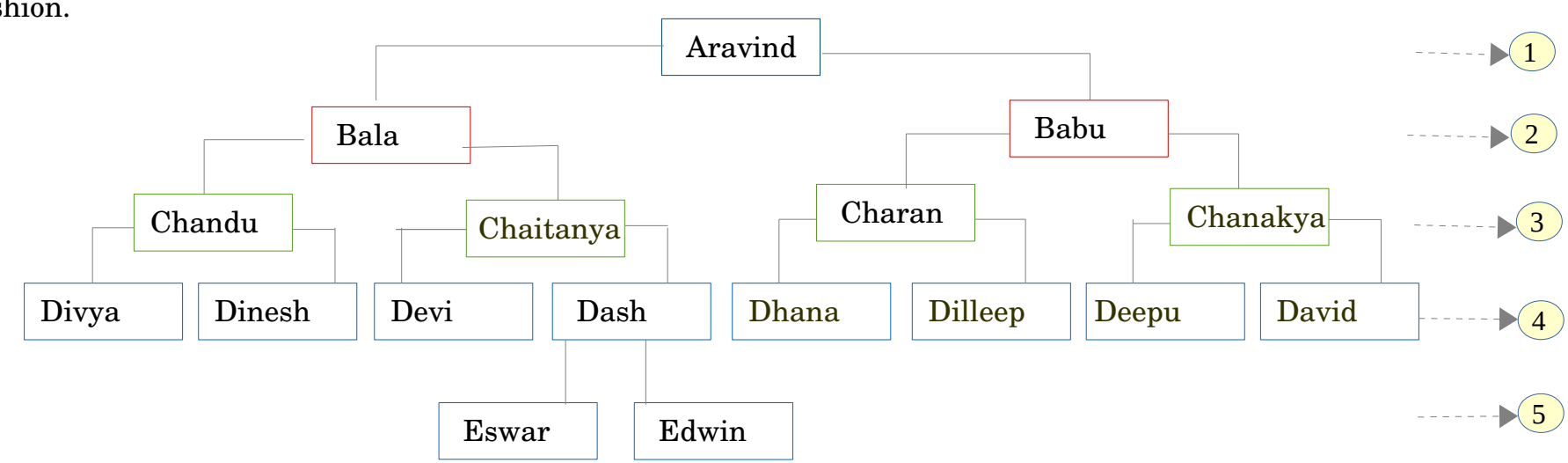
→ Divya, Dinesh, Devi, Dash (reporting to Chandu and Chaitanya)

→ Eswar and Edwin reporting to Dash

To get the above hierarchical result, a recursive SQL is needed

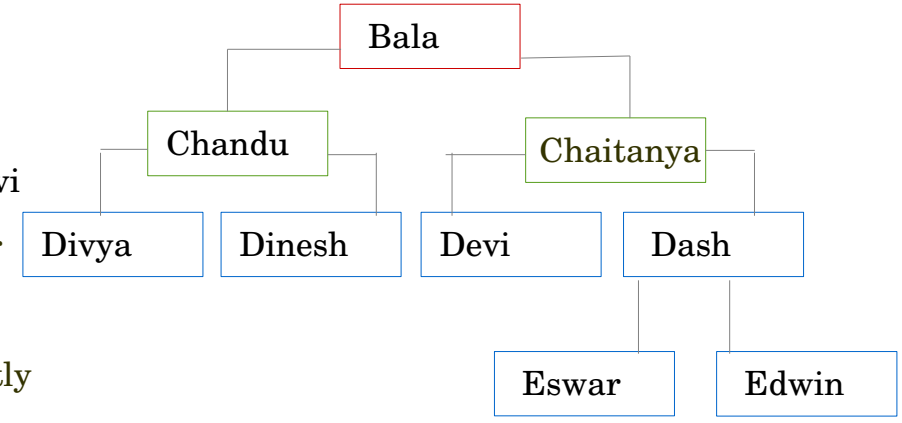
Employee	Manager
Bala	Aravind
Babu	Aravind
Chandhu	Bala
Chaitanya	Bala
Charan	Babu
Chanakya	Babu
Divya	Chandhu
Dinesh	Chandhu
Devi	Chaitanya
Dash	Chaitanya
Dhana	Charan
Dilleep	Charan
Deepu	Chanakya
David	Chanakya
Eswar	Dash
Edwin	Dash

For better understanding, the employee and manager data in the EMP table be represented in the below hierarchical fashion.



Result:

The hierarchical model for the expected result is shown on right hand side.



Explanation:

→ Chandhu and Chaitanya are directly reporting to Bala

→ Divya and Dinesh are directly reporting to Chandhu. Devi and Dash are directly reporting to Chaitanya. These four people are indirectly reporting to Bala

→ Eswar and Edwin directly reporting to Dash and indirectly reporting to Bala

2. Prerequisite for writing recursive SQL

There are 3 SQL fundamentals which should be known to programmer(s) / administrator(s) while writing recursive SQL. In other words, the following three form the basic components of recursive SQL.

(a) UNION ALL

(b) Inner join

(c) Common Table Expression (CTE)

UNION ALL

This SQL feature helps to combine the result sets of 2 or more SELECT statements. For example

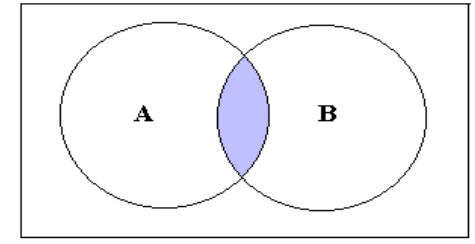
```
SELECT EMP_ID, EMP_NAME
FROM EMP_SUPPLY
UNION ALL
SELECT E_ID, E_NAME
FROM EMP_ORDERS
```

Thumb rule →

All queries should have same number of columns with compatible data types All the queries together can contain only one ORDER BY clause

Inner join

The INNER JOIN is used to select records from multiple tables using single SELECT based on the matched criteria specified in WHERE clause or ON clause.




```
SELECT SUPPLY.ID ,NAME , ORD_DATE
 FROM SUPPLY, ORDER
 WHERE SUPPLY.ID=ORDER.ID  -- Matched criteria specified in WHERE clause
```

```
SELECT SUPPLY.ID, NAME, ORD_DATE
FROM SUPPLY
     INNER JOIN
    ORDER
ON SUPPLY.ID=ORDER.ID  -- Matched criteria specified in ON clause
```

Instead of INNER JOIN we can use JOIN keyword also. JOIN keyword by default refers to INNER JOIN

Here ID column is qualified by table name to avoid ambiguous error (error caused when the same column name is present in multiple tables which are joined). Instead of qualifying column with table name we can use correlation names as shown on the right hand side. Here A and B are correlation names.

```
SELECT A.ID, A.NAME, B.ORD_DATE
FROM SUPPLY A
     INNER JOIN
     ORDER B
ON A.ID=B.ID
```

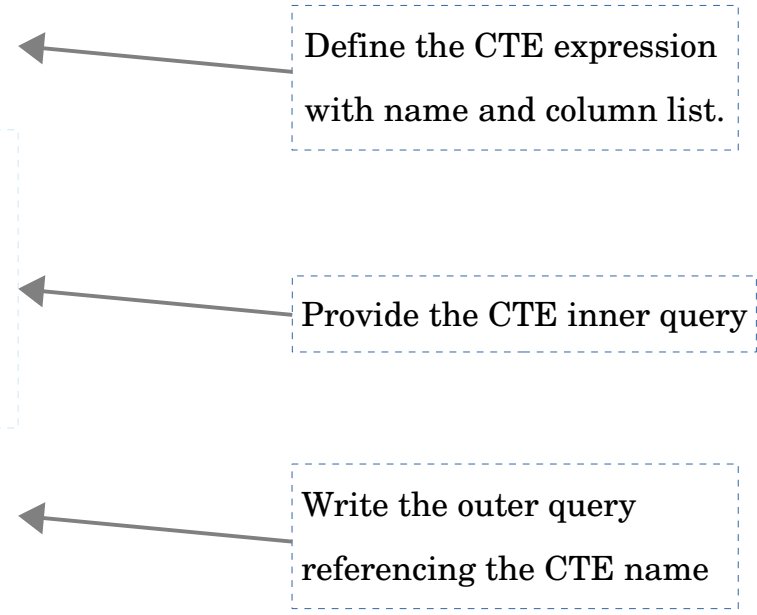
Common Table Expression (CTE)

When a SELECT query is executed, the result is stored in a temporary table called result table or result set. If user wants to query that result table, either a VIEW or MQT or table expression can be used. As part of any CTE definition, a temporary table using 'WITH' clause is defined, followed by the SELECT query as part of the 'AS' clause, which is then followed by an outer SELECT query to re-query this result table. General syntax is shown below.

```
WITH <common_table_expression_name (CTE name)> ( column_name [,...n] )
AS
(
     CTE inner SELECT query
)
CTE outer query (query on CTE name)
```

Example :

```
WITH Sales_CTE (ID, OrderID, SalesYear)
AS
(
    SELECT SalesID, Ord_ID, YEAR(O_Date) AS SalesYear
    FROM Sales
    WHERE SalesID IS NOT NULL
)
SELECT ID, COUNT(OrderID) AS TotalSales, SalesYear
FROM Sales_CTE
GROUP BY SalesYear, ID
ORDER BY ID, SalesYear;
```



Note:

- The CTE inner query can be any complex select query. It can have JOINS, UNIONS, GROUP BY and ORDER BY and other complex clauses. There is no restrictions for inner query. Only thumb rule is number of columns specified in CTE definition much match that of the CTE inner query.
- Nested table expression (NTE) is another alternative for CTE. But for the sake of performance CTE is recommended.

3. How to write recursive SQL

There is no standard syntax for recursive SQL. For beginners understanding, the below structure can be considered which will be used to handle hierarchical data using recursive SQL.

```
Syntax:
WITH cte_name ( column_name [,...n] )
AS
(
    Anchor query (Initialization query on the base table)
    UNION ALL
    Iterative query (Inner join b/w base table & CTE)
)
CTE outer query (query on cte_name)
```

CTE Inner query

```
Example: (consider EMP table in 2nd slide)
WITH EMP_CTE (EMP_NAME,MGR_NAME)
AS
(
     SELECT EMPLOYEE,MANAGER
     FROM EMP
     WHERE MANAGER = 'BALA'
     UNION ALL
     SELECT A.EMPLOYEE, A.MANAGER
     FROM EMP A, EMP_CTE B
     WHERE A.MANAGER = B.EMP_NAME
)
SELECT EMP_NAME,MGR_NAME
FROM EMP_CTE
```

Result

Employee	Manager
Chandhu	Bala
Chaitanya	Bala
Divya	Chandhu
Dinesh	Chandhu
Devi	Chaitanya
Dash	Chaitanya
Eswar	Dash
Edwin	Dash

Anchor query

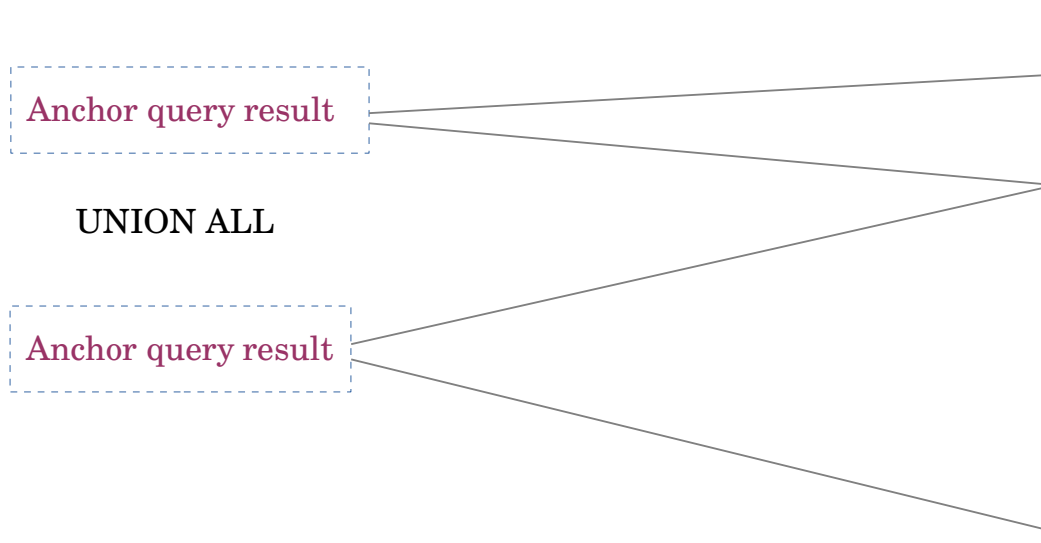
Below one is an Anchor query (or) Initialization query, as referred above, which obtains the set of people who are directly reporting to Bala.

```
SELECT EMPLOYEE,MANAGER
FROM EMP
WHERE MANAGER = 'BALA'
```

Iterative query

Below one is the Iterative query as referred above, which obtains the set of people who are indirectly reporting to Bala.

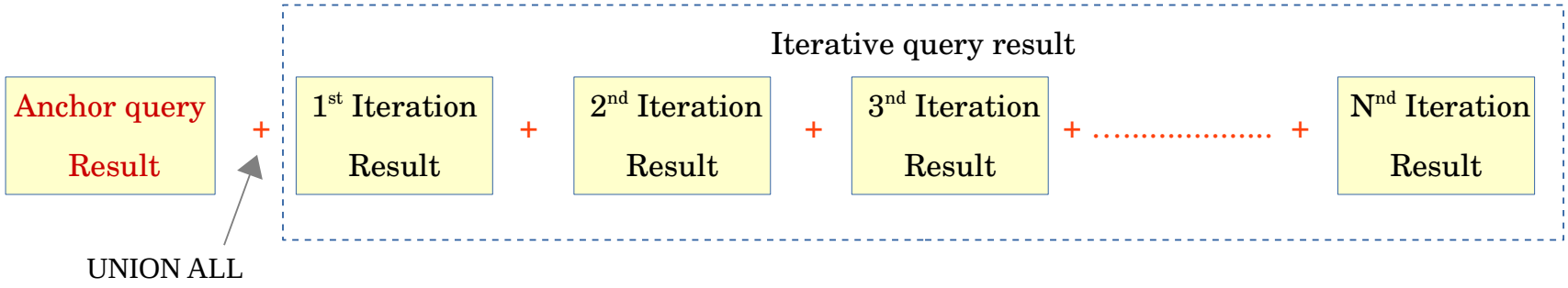
```
SELECT A.EMPLOYEE, A.MANAGER
 FROM EMP A, EMP_CTE B
 WHERE A.MANAGER = B.EMP_NAME
```



Employee	Manager
Chandhu	Bala
Chaitanya	Bala
Divya	Chandhu
Dinesh	Chandhu
Devi	Chaitanya
Dash	Chaitanya
Eswar	Dash
Edwin	Dash

4. How recursive SQL works

The total result is the aggregate of Anchor query result and Iterative query result.



Anchor query result:

```
SELECT EMPLOYEE,MANAGER
FROM EMP
WHERE MANAGER = 'BALA'
```

Employee	Manager
Chandhu	Bala
Chaitanya	Bala

Iterative query → 1st Iteration

```
SELECT A.EMPLOYEE, A.MANAGER
FROM EMP A, EMP_CTE B
WHERE A.MANAGER = B.EMP_NAME
```

EMP A

Employee	Manager
Bala	Aravind
Babu	Aravind
Chandhu	Bala
Chaitanya	Bala
Charan	Babu
Chanakya	Babu
Divya	Chandhu
Dinesh	Chandhu
Devi	Chaitanya
Dash	Chaitanya
Dhana	Charan
Dilleep	Charan
Deepu	Chanakya
David	Chanakya
Eswar	Dash
Edwin	Dash

EMP_CTE B

Employee	Manager
Chandhu	Bala
Chaitanya	Bala

1st Iteration result

Employee	Manager
Divya	Chandhu
Dinesh	Chandhu
Devi	Chaitanya
Dash	Chaitanya

→ Once the anchor query executed the iterative query starts execution .

→ Iterative query is the the inner join between EMP and EMP_CTE

→ Currently rows are populating into EMP_CTE. At this point, EMP_CTE has result of anchor query, which contains 2 rows as shown above.

→ For the inner join, it will consider the 2 rows currently stored in EMP_CTE and provide the 1st iteration result as shown above.

Iterative query → 2st Iteration

EMP A

Employee	Manager
Bala	Aravind
Babu	Aravind
Chandhu	Bala
Chaitanya	Bala
Charan	Babu
Chanakya	Babu
Divya	Chandhu
Dinesh	Chandhu
Devi	Chaitanya
Dash	Chaitanya
Dhana	Charan
Dilleep	Charan
Deepu	Chanakya
David	Chanakya
Eswar	Dash
Edwin	Dash

EMP_CTE B

Employee	Manager
Chandhu	Bala
Chaitanya	Bala
Divya	**Chandhu**
Dinesh	**Chandhu**
Devi	**Chaitanya**
Dash	**Chaitanya**

2st Iteration result

Employee	Manager
Eswar	Dash
Edwin	Dash

→ Inner join process is still NOT yet complete

→ At this point, Inner join is applied between EMP and first 2 rows in EMP_CTE only.

→ Now the next 4 rows in EMP_CTE (1st iteration result) will participate in the further joining process.

→ The 2nd iteration result is shown above.

Iterative query → 3st Iteration

EMP A

Employee	Manager
Bala	Aravind
Babu	Aravind
Chandhu	Bala
Chaitanya	Bala
Charan	Babu
Chanakya	Babu
Divya	Chandhu
Dinesh	Chandhu
Devi	Chaitanya
Dash	Chaitanya
Dhana	Charan
Dilleep	Charan
Deepu	Chanakya
David	Chanakya
Eswar	Dash
Edwin	Dash

EMP_CTE B

Employee	Manager
Chandhu	Bala
Chaitanya	Bala
Divya	Chandhu
Dinesh	Chandhu
Devi	Chaitanya
Dash	Chaitanya
Eswar	**Dash**
Edwin	**Dash**

Employee	Manager

SQLCODE : 100

→ Inner join process is still NOT yet complete

→ At this point, Inner join is applied between EMP and first 6 rows in EMP_CTE only.

→ Now next 2 rows in EMP_CTE (2st iteration result) will participate in joining process

→ The result is zero rows as Eswar and Edwin are not managers to anybody as shown in above.

So iterative query executed completly

→ Now CTE outer query will execute.

Overall result

Employee	Manager
Chandhu	Bala
Chaitanya	Bala

+

Employee	Manager
Divya	Chandhu
Dinesh	Chandhu
Devi	Chaitanya
Dash	Chaitanya

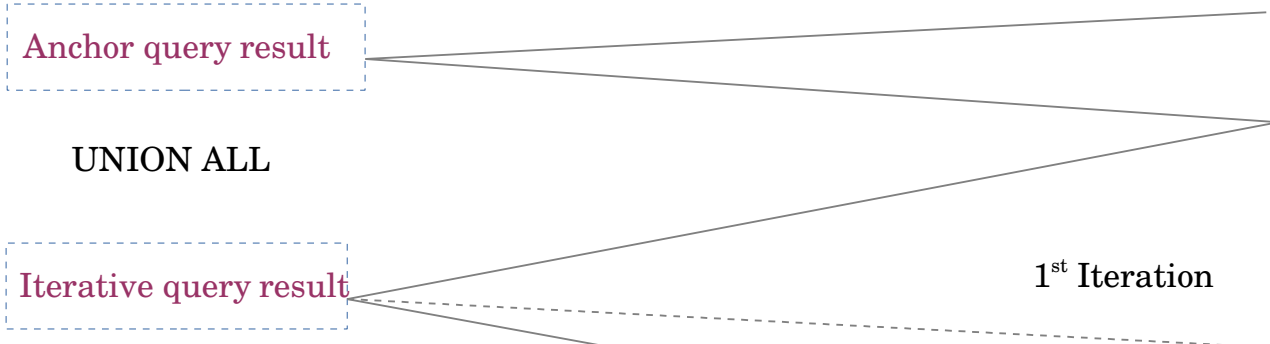
+

Employee	Manager
Eswar	Dash
Edwin	Dash

Anchor Query result

1st Iteration result

2nd Iteration result



Employee	Manager
Chandhu	Bala
Chaitanya	Bala
Divya	Chandhu
Dinesh	Chandhu
Devi	Chaitanya
Dash	Chaitanya
Eswar	Dash
Edwin	Dash

5. Recursive SQL – advanced coding

Lets consider the requirement of an addition to the above result to display hierarchical levels of Managers who are under Bala (level 2) as shown on the right hand side. The required query is shown below . Since there is no level column in the base table we can declare and initialize a dummy column in CTE as shown in query. In below query LVL is the dummy column. LEVEL is the column specified in CTE definition for this column.

Employee	Manager	Level
Chandhu	Bala	2
Chaitanya	Bala	2
Divya	Chandhu	3
Dinesh	Chandhu	3
Devi	Chaitanya	3
Dash	Chaitanya	3
Eswar	Dash	4
Edwin	Dash	4

```
WITH EMP_CTE (EMP_NAME,MGR_NAME,LEVEL)
AS
(
   SELECT EMPLOYEE, MANAGER, 2 AS LVL
   FROM EMP
   WHERE MANAGER = 'BALA'
   UNION ALL
   SELECT A.EMPLOYEE, A.MANAGER, B.LEVEL+1
   FROM EMP A, EMP_CTE B
   WHERE A.MANAGER = B.EMP_NAME
)
SELECT EMP_NAME, MGR_NAME, LEVEL
FROM EMP_CTE
```

Lets consider the requirement of displaying Head reporting manager as part of the result (in our example Head is Bala) as shown on the right hand side. The required SQL query is shown below.

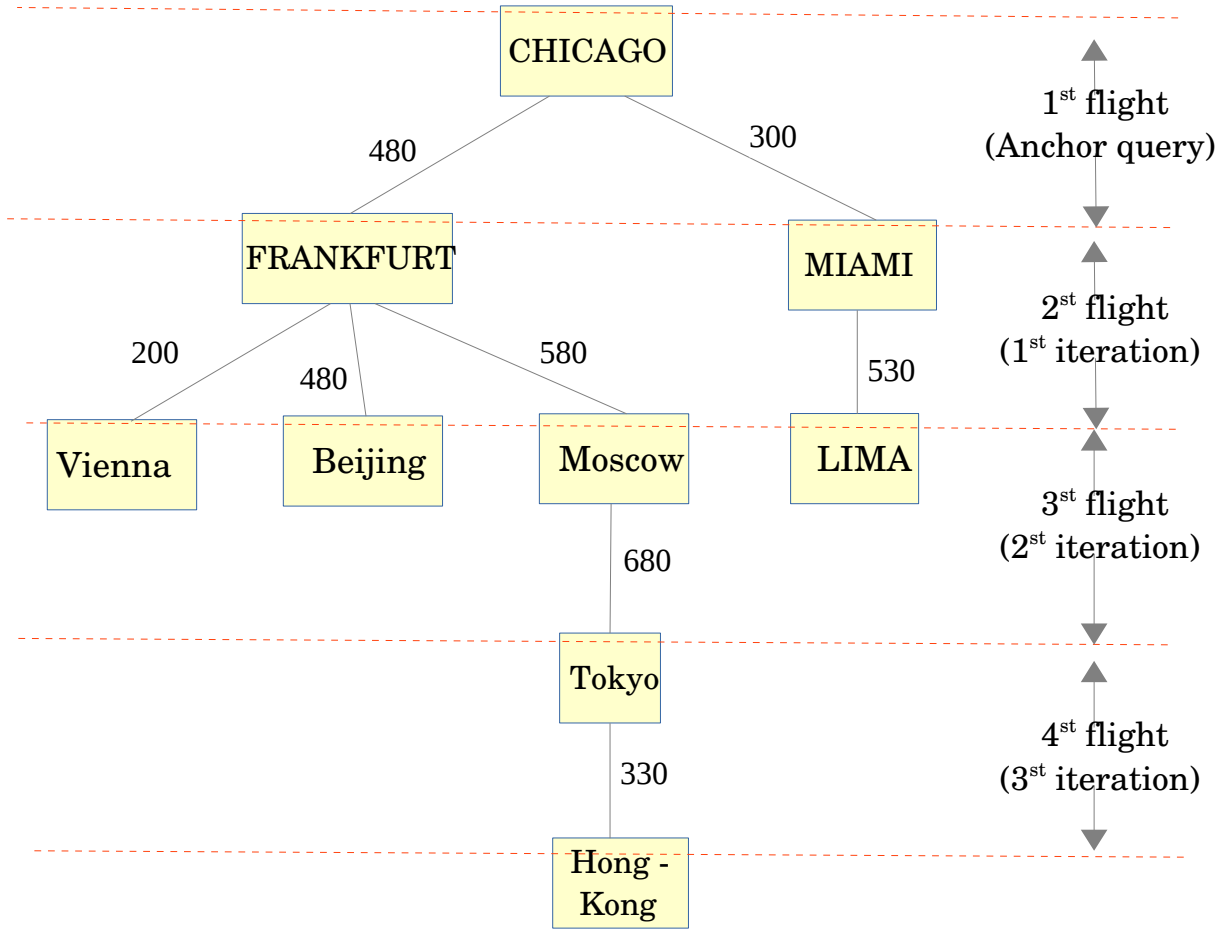
As shown , the UNDER column is loaded with 'Bala' when anchor query is executed, from then use the same column value till end of the iterative query.

Under	Employee	Manager	Level
Bala	Chandhu	Bala	2
Bala	Chaitanya	Bala	2
Bala	Divya	Chandhu	3
Bala	Dinesh	Chandhu	3
Bala	Devi	Chaitanya	3
Bala	Dash	Chaitanya	3
Bala	Eswar	Dash	4
Bala	Edwin	Dash	4

```
WITH EMP_CTE (UNDER,EMP_NAME,MGR_NAME,LEVEL)
AS
(
   SELECT MANAGER,EMPLOYEE,MANAGER,2
   FROM EMP
   WHERE MANAGER = 'BALA'
   UNION ALL
   SELECT B.UNDER, A.EMPLOYEE, A.MANAGER, B.LEVEL+1
   FROM EMP A , EMP_CTE B
   WHERE A.MANAGER = B.EMP_NAME
)
SELECT UNDER, EMP_NAME, MGR_NAME, LEVEL
FROM EMP_CTE
```

Example:2

Consider the flights table shown on right hand side. Now the requirement is obtain the list of places one can go from CHICAGO with a single or multiple flights with counts and what is the cost of the journey. The expected result is shown in the below hierarchical model.



FLIGHTS

DEPARTURE	ARRIVAL	Price
NEW YORK	PARIS	400
CHICAGO	MIAMI	300
NEW YORK	LONDON	350
LONDON	ATHENS	340
ATHENS	NICOSIA	280
PARIS	MADRID	380
PARIS	CAIRO	480
CHICAGO	FRANKFURT	480
FRANKFURT	MOSCOW	580
FRANKFURT	BEIJING	480
MOSCOW	TOKYO	680
FRANKFURT	VIENNA	200
PARIS	ROME	340
MIAMI	LIMA	530
NEW YORK	LOS ANGELES	330
LOS ANGELES	TOKYO	530
TOKYO	HONG KONG	330
WASHINGTON	TORONTO	250

```
WITH DEST (ORIGIN,DEPARTURE,ARRIVAL,COUNT,PRICE)
AS
(
     SELECT DEPARTURE,DEPARTURE,ARRIVAL,1,PRICE
     FROM FLIGHTS
     WHERE DEPARTURE = 'CHICAGO'
     UNION ALL
     SELECT B.ORIGIN,A.DEPARTURE,A.ARRIVAL,
          B.COUNT + 1,B.PRICE+A.PRICE
     FROM FLIGHTS A, DEST B
     WHERE A.DEPARTURE = B.ARRIVAL
)
SELECT ORIGIN, DEPARTURE, ARRIVAL, COUNT,PRICE
FROM DEST
```

Result set

ORIGIN	DEPARTURE	ARRIVAL	COUNT	PRICE
CHICAGO	CHICAGO	MIAMI	1	300
CHICAGO	CHICAGO	FRANKFURT	1	480
CHICAGO	MIAMI	LIMA	2	830
CHICAGO	FRANKFURT	BEIJING	2	960
CHICAGO	FRANKFURT	MOSCOW	2	1060
CHICAGO	FRANKFURT	VIENNA	2	680
CHICAGO	MOSCOW	TOKYO	3	1740
CHICAGO	TOKYO	HONG KONG	4	2070

Example 3 : Including flights table let us consider another table TRAINS. Now both put together, what are the different places one can go from CHICAGO using flights or trains or both. Query and result set shown in the following slides.

TRAINS

DEPARTURE	ARRIVAL	PRICE
CHICAGO	WASHINGTON	90
MADRID	BARCELONA	60
WASHINGTON	BOSTON	50

```
WITH DESTINATIONS (DEPARTURE, ARRIVAL, LINKS, FLIGHTS, TRAINS, COST)
AS
(SELECT DEPARTURE, ARRIVAL, 0, 1, 0, PRICE
               FROM  FLIGHTS
               WHERE DEPARTURE = 'CHICAGO'
   UNION ALL
   SELECT DEPARTURE, ARRIVAL, 0, 0, 1, PRICE
            FROM TRAINS
            WHERE DEPARTURE = 'CHICAGO'
   UNION ALL
   SELECT R.DEPARTURE, A.ARRIVAL, R.LINKS + 1 , R.FLIGHTS + 1,R.TRAINS, R.COST + A.PRICE
               FROM  FLIGHTS A ,DESTINATIONS R
               WHERE A.DEPARTURE=R.ARRIVAL
   UNION ALL
   SELECT R.DEPARTURE, A.ARRIVAL, R.LINKS + 1 , R.FLIGHTS, R.TRAINS + 1, R.COST + A.PRICE
                FROM TRAINS A ,DESTINATIONS R
                WHERE A.DEPARTURE=R.ARRIVAL
)
SELECT DEPARTURE, ARRIVAL, LINKS, FLIGHTS, TRAINS, COST
FROM DESTINATIONS
```

Result set:

```
DEPARTURE   ARRIVAL          Links   FLIGHTS    TRAINS      COST
---------+---------+---------+---------+---------+---------+---------+-
CHICAGO     MIAMI              0         1         0       300
CHICAGO     FRANKFURT          0         1         0       480
CHICAGO     WASHINGTON         0         0         1        90
CHICAGO     LIMA               1         2         0       830
CHICAGO     BEIJING            1         2         0       960
CHICAGO     MOSCOW             1         2         0      1060
CHICAGO     VIENNA             1         2         0       680
CHICAGO     TORONTO            1         1         1       340
CHICAGO     TOKYO              2         3         0      1740
CHICAGO     HONG KONG          3         4         0      2070
CHICAGO     BOSTON             1         0         2       140
DSNE610I NUMBER OF ROWS DISPLAYED IS 11
DSNE612I DATA FOR COLUMN HEADER DEPARTURE COLUMN NUMBER 1 WAS TRUNCATED
DSNE612I DATA FOR COLUMN HEADER ARRIVAL COLUMN NUMBER 2 WAS TRUNCATED
DSNE611I COLUMN HEADER CONNECTIONS FOR COLUMN NUMBER 3 WAS TRUNCATED
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+-
```

Non-hierarchical example for recursive SQL

The recursive SQL is not only for hierarchical data model, rather can be used for other applications which purely depends upon programmer's logical ability. Consider the following query.

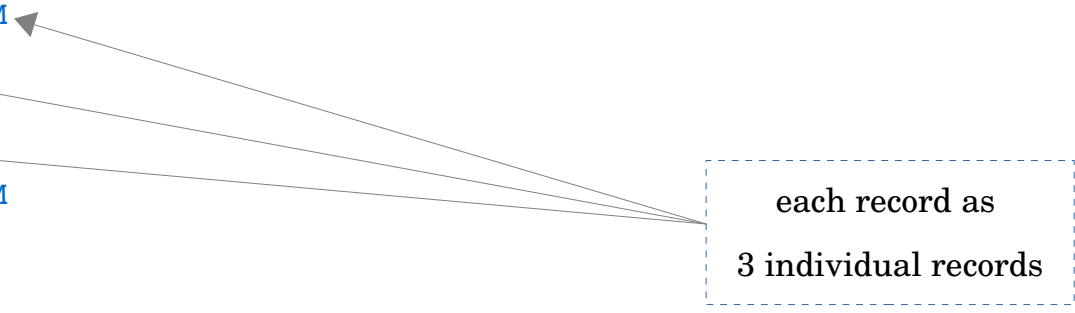
```
SELECT   'SELECT COUNT(*) FROM '|| STRIP(CREATOR)||'.'||STRIP(NAME)||' WITH UR;'
FROM SYSIBM.SYSTABLES
WHERE CREATOR = 'VXB5B1' ;
```

Result:

```
SELECT COUNT(*) FROM VXB5B1.ACC_LVL WITH UR;
SELECT COUNT(*) FROM VXB5B1.EMP_DATA WITH UR;
SELECT COUNT(*) FROM VXB5B1.EMP_DATA_AUX WITH UR;
```

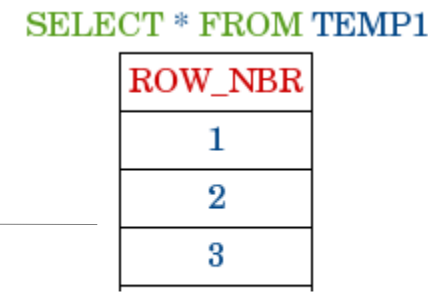
Suppose, due to some record length issues we need each record as 3 individual records (3 separate lines) as shown in below.

```
SELECT COUNT(*) FROM
VXB5B1.ACC_LVL
WITH UR;
SELECT COUNT(*) FROM
VXB5B1.EMP_DATA
WITH UR;
SELECT COUNT(*) FROM
VXB5B1.EMP_DATA_AUX
WITH UR;
```



To get this desired result we can use following recursive query.

```
WITH TEMP1(ROW_NBR) AS
(
   SELECT 1
   FROM SYSIBM.SYSDUMMY1
   UNION ALL
   SELECT ROW_NBR + 1
   FROM TEMP1
   WHERE ROW_NBR < 3
)
 SELECT CASE ROW_NBR
        WHEN 1 THEN CHAR('SELECT COUNT(*) FROM',80)
        WHEN 2 THEN CHAR(STRIP(CREATOR)||'.'||STRIP(NAME),80)
        WHEN 3 THEN CHAR('WITH UR',80)
        END
FROM SYSIBM.SYSTABLES,TEMP1
WHERE CREATOR = 'VXB5B1'
ORDER BY CREATOR,NAME,ROW_NBR
With UR;
```



- ✔ Using CTE we created a temporary table called TEMP1 which has 4 numeric numbers 1-3. the result is inner join between TEMP1 and SYSTABLES. One record is split into 3 records using CASE expression.
- ✔ Here ORDER BY clause is very important since these individual records may be any order. One of the common undesired result shown on the left hand side (If ORDER BY clause is not specified then query is ordering records based on only ROW_NBR).

```
Undesired result
SELECT COUNT(*) FROM
SELECT COUNT(*) FROM
SELECT COUNT(*) FROM
VXB5B1.ACC_LVL
VXB5B1.EMP_DATA
VXB5B1.EMP_DATA_AUX
WITH UR;
WITH UR;
WITH UR;
```

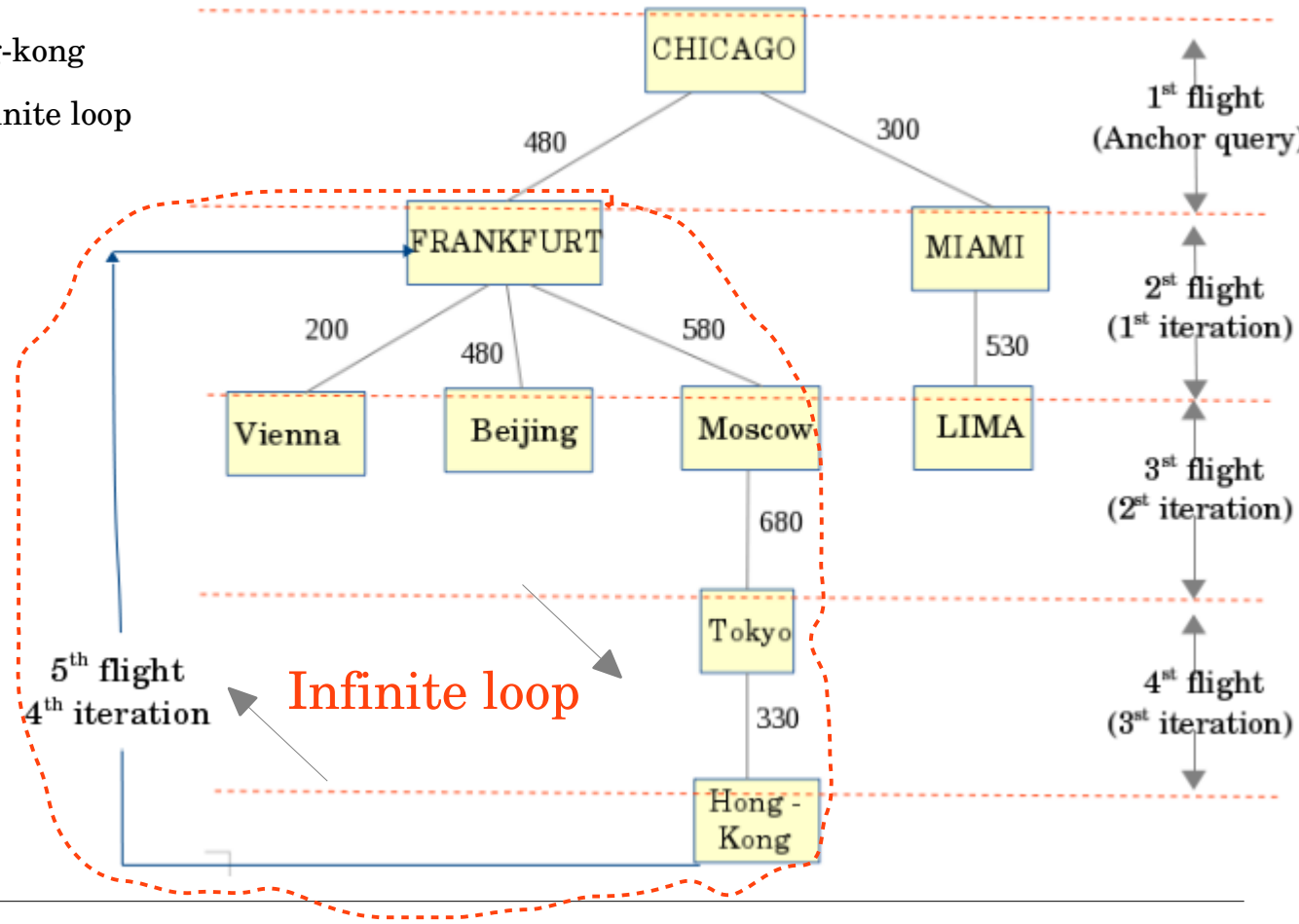
6. Infinite loop

Most of the recursive SQL's will give warning SQLCODE 347 since iterations may not end, it may go to infinite

```
DSNT404I SQLCODE = 347, WARNING:  THE RECURSIVE COMMON TABLE EXPRESSION
         DESTINATIONS MAY CONTAIN AN INFINITE LOOP
DSNT418I SQLSTATE   = 01605 SQLSTATE RETURN CODE
```

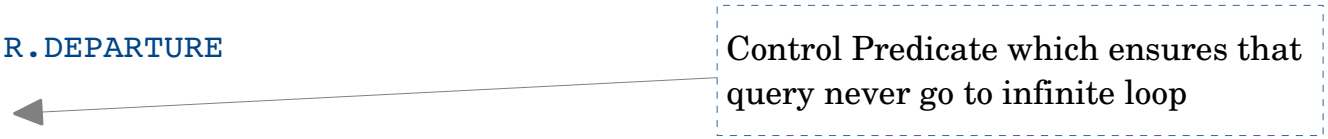
For example if the flight table has a record which tells there is a flight from Hong-kong to FRANKFURT then it leads to a infinite loop as shown in figure.

But for hierarchical data query never goes to infinite loop. The data shown in diagram is network data since the Hong-kong has a child node called FRANKFURT which is great grant parent node also.



Then how to control dept of recursive. For suppose if we are sure the query will not have iterations more then 20 then following is the way to control dept of recursive.

```
WITH DEST (ORIGIN,DEPARTURE,ARRIVAL,F_COUNT,PRICE)
AS
(
  SELECT DEPARTURE,DEPARTURE,ARRIVAL,1,PRICE
         FROM FLIGHTS A
         WHERE A.DEPARTURE = 'CHICAGO'
  UNION ALL
  SELECT R.ORIGIN,B.DEPARTURE,B.ARRIVAL,R.F_COUNT + 1,
         R.PRICE+B.PRICE
         FROM DEST R, FLIGHTS B
         WHERE B.ARRIVAL = R.DEPARTURE
         AND F_COUNT <= 20
)
SELECT ORIGIN, DEPARTURE, ARRIVAL, F_COUNT,PRICE
FROM DEST
```



Since above query never gets into infinite loop, a SQLCODE 100 will be encountered. Otherwise without the control predicate F_COUNT <=20 , a SQLCODE 347 will be encountered . This control predicate is also used for debugging the query. The control predicate can be used appropriately to limit the number of iterations as part of the query processing and also helps to understand what result is retrieved at which iteration.

Bibliography

https://www.ibm.com/support/knowledgecenter/search/

Efforts Never Die