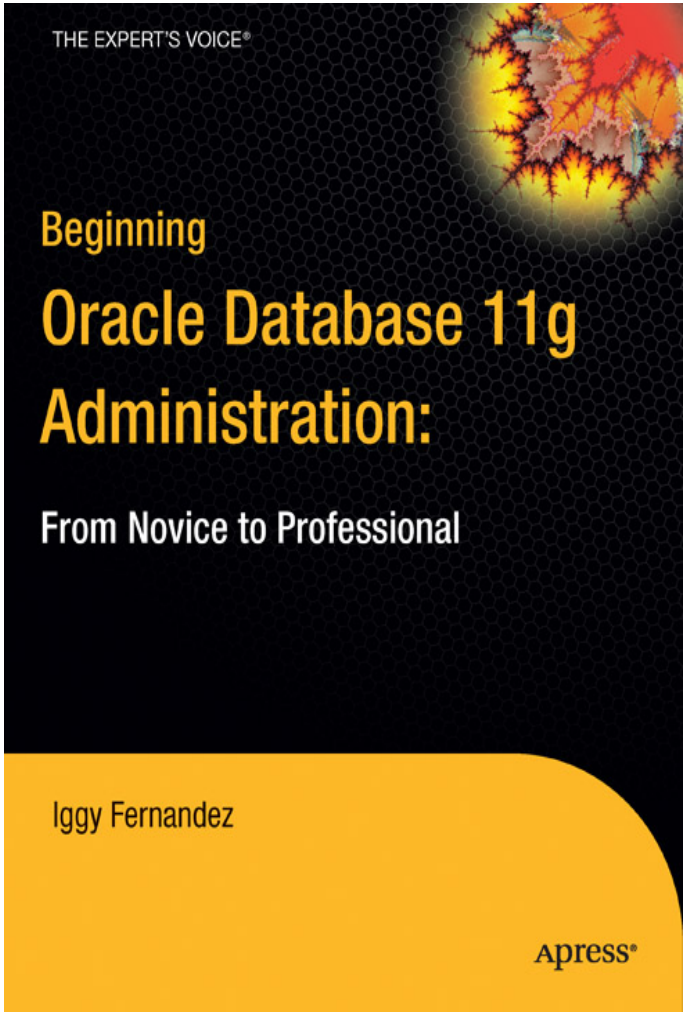
COLLABORATE 10
Technology and Applications Forum for the Oracle Community

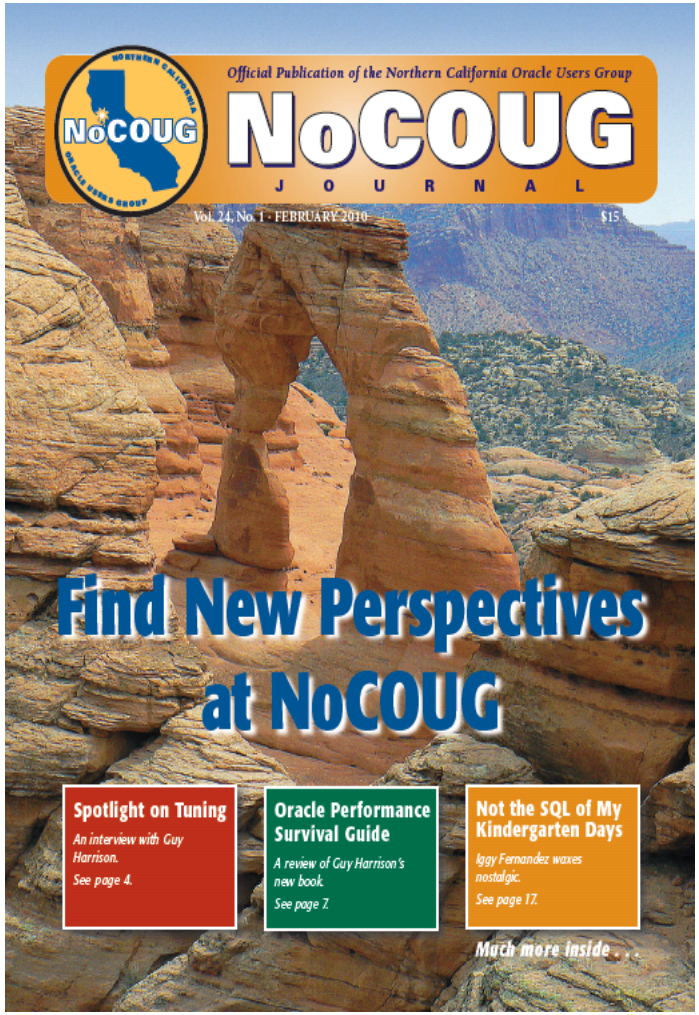
IOUG
independent oracle users group

Recursive Common Table Expressions in Oracle Database 11*g* Release 2

Iggy Fernandez
Database Specialists
Session #303

DatabaseSpecialists


THE EXPERT'S VOICE®
Beginning
Oracle Database 11g
Administration:
From Novice to Professional
Iggy Fernandez
Apress®


Official Publication of the Northern California Oracle Users Group
NoCOUG
JOURNAL
Vol. 24, No. 1 · FEBRUARY 2010
$15
Find New Perspectives at NoCOUG
Spotlight on Tuning
An interview with Guy Harrison.
See page 4.
Oracle Performance Survival Guide
A review of Guy Harrison's new book.
See page 7.
Not the SQL of My Kindergarten Days
Iggy Fernandez waxes nostalgic.
See page 17.
Much more inside . . .

CTE Recap
Inline Views

```
SELECT *
FROM    (SELECT *
         FROM    Suppliers
         MINUS
         SELECT *
         FROM    (SELECT SupplierName
                  FROM    (SELECT *
                           FROM    (SELECT *
                                    FROM    Suppliers
                                    Parts)
                           MINUS
                           SELECT *
                           FROM    (SELECT SupplierName,
                                    PartName
                                    FROM    Quotes))));
```

CTE Recap
All Supplier Part Pairs

```
WITH

AllSupplierPartPairs AS
(
  SELECT *
  FROM Suppliers, Parts
),
```

CTE Recap
Valid Supplier Part Pairs

```
ValidSupplierPartPairs AS
(
  SELECT SupplierName, PartName
  FROM Quotes
),
```

CTE Recap
Invalid Supplier Part Pairs

```
InvalidSupplierPartPairs AS
(
  SELECT *
  FROM AllSupplierPartPairs
  MINUS
  SELECT *
  FROM ValidSupplierPartPairs
),
```

CTE Recap
Suppliers Who Don't Supply All Parts

```
SuppliersWhoDontSupplyAllParts AS
(
  SELECT SupplierName
  FROM InvalidSupplierPartPairs
),
```

CTE Recap
Suppliers Who Supply All Parts

```
SuppliersWhoSupplyAllParts AS
(
  SELECT *
  FROM Suppliers
  MINUS
  SELECT *
  FROM SuppliersWhoDontSupplyAllParts
)
```

CTE Recap
Suppliers Who Supply All Parts

```
SELECT *
FROM SuppliersWhoSupplyAllParts;
```

Recursive CTE Algorithm

1. Split the CTE expression into anchor and recursive members.
2. Run the anchor member(s) creating the first invocation or base result set (T0).
3. Run the recursive member(s) with Ti as an input and Ti+1 as an output.
4. Repeat step 3 until an empty set is returned.
5. Return the result set. This is a UNION ALL of T0 to Tn.

Number Generator Old Style

```
SELECT level AS n
FROM dual
CONNECT BY level <= 100;
```

Number Generator New Style

```
WITH numbers(n) AS
(
  -- Anchor member
  SELECT 1 FROM dual

  UNION ALL

  -- Recursive member
  SELECT n + 1 FROM numbers WHERE n < 100
)
SELECT * FROM numbers;
```

Traditional Hierarchical Queries Managers and Employees

```
SELECT
  LPAD (' ', 4 * (LEVEL - 1)) || first_name || ' ' ||
last_name AS name
FROM employees
START WITH manager_id IS NULL
CONNECT BY manager_id = PRIOR employee_id;
```

Traditional Hierarchical Queries Managers and Employees

```
Name
--------------------------------------------------
Steven King
    Neena Kochhar
        Nancy Greenberg
            Daniel Faviet
            John Chen
            Ismael Sciarra
            Jose Manuel Urman
            Luis Popp
        Jennifer Whalen
        Susan Mavris
        Hermann Baer
        Shelley Higgins
            William Gietz
```

Traditional Hierarchical Queries Managers and Employees

```
WITH

RCTE (employee_id, first_name, last_name, lvl) AS
(

SELECT
  employee_id,
  first_name,
  last_name,
  1 AS lvl
FROM
  employees
WHERE manager_id IS NULL
```

Traditional Hierarchical Queries Managers and Employees

```
UNION ALL

SELECT
  e.employee_id,
  e.first_name,
  e.last_name,
  lvl + 1 AS lvl
FROM
  RCTE INNER JOIN employees e
  ON (RCTE.employee_id = e.manager_id)
)
-- SEARCH DEPTH FIRST BY employee_id ASC SET seq#
```

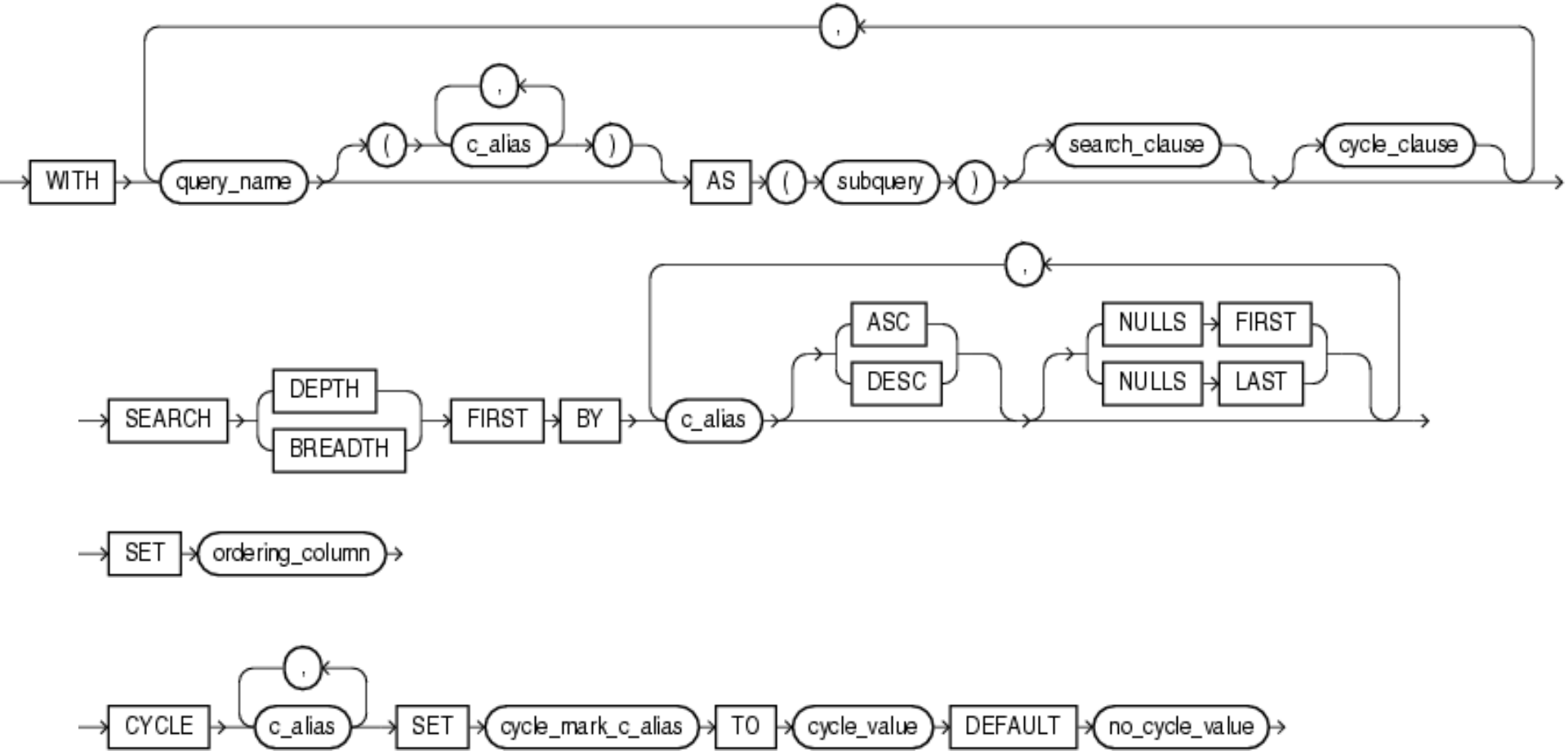
Traditional Hierarchical Queries Managers and Employees

```
SELECT LPAD (' ', 4 * (lvl - 1)) || first_name || ' ' ||
last_name AS name
FROM RCTE
--ORDER BY seq#;
```

Traditional Hierarchical Queries Breadth First Search

```
Steven King
    Michael Hartstein
    Neena Kochhar
    Lex De Haan
    Den Raphaely
    Matthew Weiss
    Adam Fripp
    Payam Kaufling
    Shanta Vollman
    Kevin Mourgos
    John Russell
    Karen Partners
    Alberto Errazuriz
    Gerald Cambrault
    Eleni Zlotkey
        Pat Fay
        Jennifer Whalen
```

Railroad Diagram



Restrictions

The recursive member cannot contain any of the following elements:

The DISTINCT keyword or a GROUP BY clause

The model_clause

An aggregate function. However, analytic functions are permitted in the select list.

Subqueries that refer to the recursive member.

Outer joins that refer to recursive member as the right table.

Other Goodies

- SYS_CONNECT_BY_PATH
- CONNECT_BY_ROOT
- CONNECT_BY_CYCLE
- CONNECT_BY_ISLEAF
- ORDER SIBLINGS BY

Coupon Clipping

Given a list of products and a list of discount coupons, we needed to find the minimum price for all the products based on certain rules. Here are those rules:

A maximum of ten coupons can be applied on the same product.

The discount price can not be less than 70% of the original price.

The total amount of the discount can not exceed 30$.

Coupon Clipping

Id	Name	Price	Discounted Price	Discount Amount	Discount Rate	Coupon Names
1	PROD 1	100.00	73.00	27.00	27.00	CP 1 : -15$ + CP 4 : -12$
2	PROD 2	220.00	193.00	27.00	12.27	CP 1 : -15$ + CP 4 : -12$
3	PROD 3	15.00	13.50	1.50	10.00	CP 3 : -10%
4	PROD 4	70.00	49.50	20.50	29.29	CP 1 : -15$ + CP 3 : -10%
5	PROD 5	150.00	120.00	30.00	20.00	CP 3 : -10% + CP 1 : -15$

Coupon Clipping

```
CREATE TABLE products (ID INTEGER PRIMARY KEY, Name
VARCHAR2(20),Price NUMBER);

INSERT INTO products VALUES (1,'PROD 1',100);
INSERT INTO products VALUES (2,'PROD 2',220);
INSERT INTO products VALUES (3,'PROD 3',15);
INSERT INTO products VALUES (4,'PROD 4',70);
INSERT INTO products VALUES (5,'PROD 5',150);

CREATE TABLE coupons (ID INTEGER PRIMARY KEY, Name VARCHAR2(20),
Value INTEGER, IsPercent CHAR(1));

INSERT INTO coupons VALUES (1,'CP 1 : -15$',15,'N');
INSERT INTO coupons VALUES (2,'CP 2 : -5$',5,'N');
INSERT INTO coupons VALUES (3,'CP 3 : -10%',10,'Y');
INSERT INTO coupons VALUES (4,'CP 4 : -12$',12,'N');
```

Coupon Clipping

```
WITH RCTE(ID, Name, Price, DiscountedPrice, DiscountAmount,
DiscountRate, CouponNames, CouponCount, CouponID) AS
(
SELECT
  ID,
  Name,
  Price,
  Price AS DiscountedPrice,
  0 AS DiscountAmount,
  0 AS DiscountRate,
  CAST(' ' AS VARCHAR2(1024)) AS CouponNames,
  0 AS CouponCount,
  -1 AS CouponId
FROM
  products
```

Coupon Clipping

```
UNION ALL


SELECT
  RCTE.ID, RCTE.Name, RCTE.Price,
  DECODE(C.IsPercent, 'N', RCTE.DiscountedPrice - C.Value,
RCTE.DiscountedPrice - (RCTE.DiscountedPrice / 100 * C.Value))
DiscountedPrice,
  RCTE.Price - DiscountedPrice AS DiscountAmount,
  (RCTE.Price - DiscountedPrice) / RCTE.Price * 100 AS
DiscountRate,
  DECODE(RCTE.CouponNames, ' ', C.Name, RCTE.CouponNames || ' + '
|| C.Name) AS CouponNames,
  RCTE.CouponCount + 1 AS CouponCount,
  C.ID AS CouponID
FROM RCTE, coupons C
WHERE
  instr(RCTE.couponnames, c.Name) = 0 AND CouponCount < 2 AND
DiscountAmount <= 30 AND DiscountRate <= 30
),
```

Coupon Clipping

```
SortedPrices AS
(
  SELECT
    RCTE.*,
    ROW_NUMBER() OVER (PARTITION BY ID ORDER BY DiscountedPrice)
AS RowNumber
  FROM RCTE
)

SELECT
  ID, Name, Price,
  DiscountedPrice, DiscountAmount, DiscountRate,
  CouponNames
FROM SortedPrices
WHERE RowNumber = 1
ORDER BY ID;
```

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Sudoku!

```
WITH RecursiveCTE(PartiallySolvedSudoku, BlankCell) AS
(
  SELECT
    cast(rpad('&&SudokuPuzzle', 81) AS VARCHAR2(81)) AS SudokuPuzzle,
    instr(rpad('&&SudokuPuzzle', 81), ' ', 1) AS FirstBlankCell
  FROM dual
  UNION ALL
  SELECT
    cast(substr(RecursiveCTE.PartiallySolvedSudoku, 1, BlankCell - 1) ||
to_char(Candidates.N) || substr(RecursiveCTE.PartiallySolvedSudoku, BlankCell + 1)
AS VARCHAR2(81)) AS PartiallySolvedSudoku,
    instr(RecursiveCTE.PartiallySolvedSudoku, ' ', RecursiveCTE.BlankCell + 1) AS
NextBlankCell
  FROM RecursiveCTE, Candidates
  WHERE
  -- Check the contents of the row containing the blank cell
  -- Check the contents of the column containing the blank cell
  -- Check the contents of the 3x3 grid containing the blank cell
  AND BlankCell > 0
)
SELECT PartiallySolvedSudoku "Partially Solved Sudoku" FROM RecursiveCTE;
```

Thanks For Listening

iggy_fernandez@hotmail.com
http://iggyfernandez.wordpress.com

Please submit evaluation forms