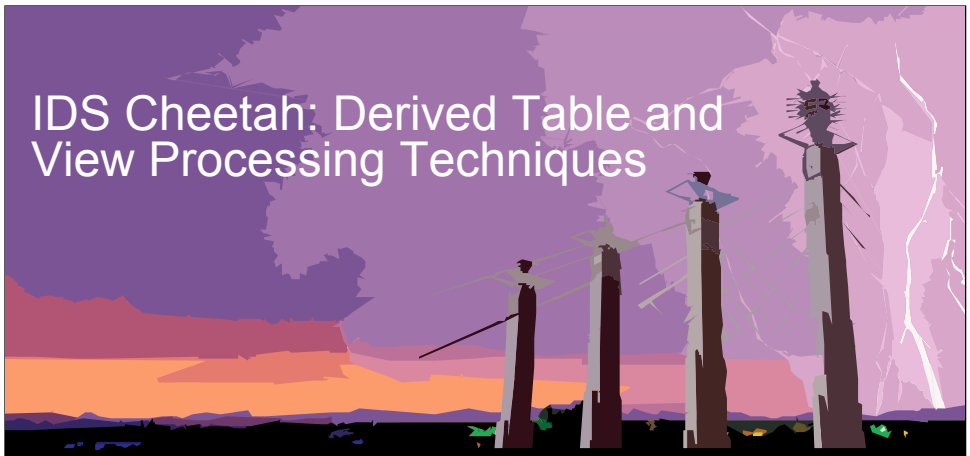


IDS Cheetah: Derived Table and View Processing Techniques

Ajaykumar Gupte
IBM

B07
Day, April 29, 2008 • 09:30 a.m. – 10:30 a.m.

2008 IIUG Informix Conference

www.iiug.org

Agenda

- Overview
- Derived table scope
- Iterator (table) function
- Derived table & views
- View folding techniques
- Tips

Overview

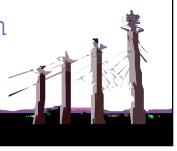
- Legacy syntax for collection derived table
  - TABLE(MULTISET(<full select >))
- Non SQL Standard
- Limitations – serial/serial8 & blob data types, duplicate column names, rowid
- Porting applications
- Tricky syntax with IDS multiset datatype -multiset(<element type> not null)

```
SELECT * FROM table(multiset ( SELECT
  customer_num FROM customer) );
```

Overview

- SQL Standard syntax
- 'AS' & <table-name> clause optional
- Dynamic processing – no information stored in catalog tables
- View processing rules are applicable
- Scope
  - can be used where full select is allowed, projection list, nested cases, views, triggers, stored procedures
  - Table references

```
SELECT * FROM ( SELECT customer_num
  FROM customer);
```



Derived Table Scope

- Simple case

```
SELECT stock_num, manu_code FROM
  (SELECT * FROM items WHERE stock_num = 5);
```

- AS clause

```
SELECT stock_num, manu_code FROM
  (SELECT * FROM items WHERE quantity > 3) AS vtab
    WHERE vtab.stock_num = 5;
```

- Derived columns

```
SELECT vc1, vc2 FROM
  (SELECT stock_num, manu_code FROM items
    WHERE quantity > 3) AS vtab(vc1,vc2)
  WHERE vtab.vc1 = 5;
```

Scope

- Multiple tables

```
SELECT * FROM
  (SELECT stock.stock_num FROM stock, stock_discount
    WHERE stock.stock_num = stock_discount.stock_num)
    AS dtab(common_num);
```

- Multiple tables with duplicate column names

```
SELECT * FROM
 (SELECT stock.stock_num , stock_discount.stock_num
  FROM stock, stock_discount
   WHERE stock.stock_num != stock_discount.stock_num);
```

- Output

```
 stock_num stock_num_1
     1          201
     1          201
 ----------------------------------
```

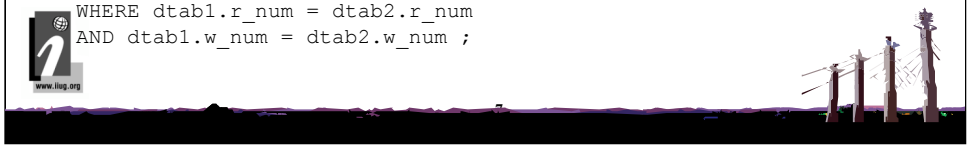
Scope

- Skip M - first N inside derived table

```
SELECT * FROM
  (SELECT skip 5 first 10 stock.stock_num, stock_discount.stock_num
    FROM stock, stock_discount
   WHERE stock.stock_num != stock_discount.stock_num)
 AS dtab(stock_num,dis_stock_num);
```

- Multiple derived tables

```
SELECT dtab1.r_num, dtab1.w_num, location, discount
  FROM (SELECT r.customer_num, w.customer_num,w.customer_loc
          FROM retail_customer r, whlsale_customer w
         WHERE w.customer_loc = r.customer_loc)
       AS dtab1(r_num, w_num, location),
      (SELECT r.customer_num, w.customer_num, w.cust_discount
         FROM retail_customer r, whlsale_customer w
        WHERE w.cust_discount = r.cust_discount)
       AS dtab2(r_num, w_num, discount)
 WHERE dtab1.r_num = dtab2.r_num
 AND dtab1.w_num = dtab2.w_num ;
```



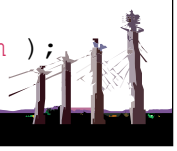
Scope

- ORDER BY case

```
SELECT * FROM
  (SELECT s.stock_num,sd.stock_num
    FROM stock d, stock_discount sd
   WHERE s.stock_num != sd.stock_num
   ORDER BY 1);
```

- Derived table inside view

```
CREATE VIEW stock_view (stock_num,
  dis_stock_num) AS SELECT * FROM
    (SELECT s.stock_num, sd.stock_num
       FROM stock s, stock_discount sd
      WHERE s.stock_num != sd.stock_num );
```



Scope

- New & old syntax

```
SELECT item_tab.item_order, item_tab.item_id,
   order_tab.ord_date
 FROM table(multiset( SELECT item_num,
   order_num FROM items ))
    AS item_tab(item_id,item_order),
    (SELECT order_date, order_num FROM orders)
      AS order_tab(ord_date,ord_num)
WHERE item_tab.item_order = order_tab.ord_num;
```

- Derived table in projection list

```
SELECT (SELECT count(*) FROM
 (SELECT * FROM stock, stock_discount
  WHERE stock.stock_num = stock_discount.stock_num)
  )
 FROM items;
```

Scope

- UNION/UNION ALL node

```
SELECT number FROM
 (SELECT order_num FROM orders
   WHERE order_date > '10/10/1998'
  UNION
  SELECT order_num FROM items
   WHERE item_subtotal BETWEEN 200 AND 300)
 AS union_tab(number);

SELECT stocknum, catalog_num FROM
 (SELECT stock_num FROM stock
   WHERE unit_price BETWEEN 200 AND 300
 UNION ALL
  SELECT stock_num FROM items
   WHERE item_subtotal BETWEEN 200 AND 300)
AS unionall_tab(stocknum), catalog
WHERE unionall_tab.stocknum = catalog.stock_num;
```

Scope - Rowid

```
SELECT rowid FROM (SELECT item_num FROM items)
  AS itemtab(item_num);

SELECT drowid FROM
 (SELECT rowid, item_num FROM items)
   AS itemtab(drowid, item_num);

SELECT rowid FROM
  (SELECT item_num, unit_price
    FROM items, stock
    WHERE items.stock_num = stock.stock_num)
 AS item_stock(item_num, unit_price);
```

-- 205: Cannot use ROWID for views with union, aggregates, group by, multiple tables, or derived expressions.

```
SELECT item_rowid FROM
  (SELECT items.rowid, stock.rowid, item_num,
     unit_price FROM items, stock
    WHERE items.stock_num = stock.stock_num)
 AS item_stock(item_rowid, stock_rowid, item_num,
  unit_price);
```

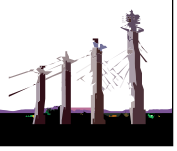
Scope

- SERIAL/SERIAL8 and BLOB data types

```
CREATE TABLE blobtab(
  c1 serial, c2 text, c3 byte,
  c4 blob, c5 clob);

INSERT INTO blobtab
  VALUES (1, NULL, NULL, NULL, NULL);

SELECT * FROM (SELECT * FROM blobtab);
```

- Output

```
c1  1
c2
c3  <BYTE value>
c4  <SBlob Data>
c5

1 row(s) retrieved.
```

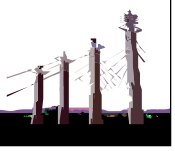


Scope – Nested Cases

```
SELECT number FROM
  (SELECT * FROM
    (SELECT item_num FROM items)
     AS item_in(num_in) )
    AS item_tab(number);
```

- Derived table in projection list

```
SELECT ( SELECT count(*) FROM
  (SELECT * FROM stock,
    (SELECT *
     FROM stock_discount d, items I
     WHERE d.manu_code MATCHES i.manu_code)
    AS discount_tab
   WHERE stock.stock_num = discount_tab.stock_num)
 ) FROM items;
```



Scope – Subquery

```
SELECT * FROM
  (SELECT stock_num FROM stock_discount
     WHERE unit_discount > 0.30) AS dtab(stknum)
 WHERE stknum > ANY
  (SELECT stock_num FROM items
    WHERE stock_num = stknum
      AND item_subtotal > 150) ;

SELECT stknum FROM
  (SELECT stock_num FROM stock_discount
     WHERE unit_discount > 0.30) AS dtab(stknum)
 WHERE stknum > 500
   OR EXISTS (SELECT stock_num FROM items
        WHERE stock_num = stknum
          AND item_subtotal > 150) ;
```

Scope – ANSI Joins

```
SELECT * FROM
  (SELECT c.stock_num, c.catalog_num
    FROM catalog c ,stock s
   WHERE c.stock_num = s.stock_num
     AND c.manu_code NOT MATCHES 'ANZ')
   AS vtab1(c1,c2)
  LEFT JOIN (SELECT stock_num FROM items
    WHERE item_subtotal > 100) AS vtab2(vc1)
 ON vtab1.c1 = vtab2.vc1
 ORDER BY c1;
```

```
SELECT * FROM
  (SELECT c.stock_num, c.catalog_num
    FROM catalog c RIGHT JOIN stock s
     ON c.stock_num = s.stock_num
    AND c.manu_code NOT MATCHES 'ANZ')
    AS vtab1(c1,c2)
   LEFT JOIN (SELECT stock_num FROM items
      WHERE item_subtotal > 100) AS vtab2(vc1)
  ON vtab1.c1 = vtab2.vc1
ORDER BY c1;
```

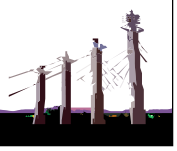
Scope – ANSI Joins

```
SELECT * FROM
 (SELECT c.stock_num, c.catalog_num
   FROM catalog c RIGHT JOIN stock s
     ON c.stock_num = s.stock_num
    AND c.manu_code NOT MATCHES 'ANZ')
    AS vtab1(c1,c2) FULL JOIN
   (SELECT stock_num FROM items
    WHERE item_subtotal > 100) AS vtab2(vc1)
   ON vtab1.c1 = vtab2.vc1 ORDER BY c1;

SELECT * FROM
 (SELECT c.stock_num, c.catalog_num
    FROM catalog c , stock s
   WHERE c.stock_num = s.stock_num
     AND c.manu_code NOT MATCHES 'ANZ') AS
  vtab1(c1,c2), OUTER (SELECT stock_num FROM items
      WHERE item_subtotal > 100) AS vtab2(vc1)
 WHERE vtab1.c1 = vtab2.vc1 ORDER BY c1;
```

Scope – ANSI Joins - Error Case

- 19820: Informix OUTER JOIN and ANSI JOIN in the same query block.

```
SELECT * FROM
 (SELECT c.stock_num, c.catalog_num FROM catalog c
   LEFT JOIN stock s
    ON c.stock_num = s.stock_num
   AND c.manu_code NOT MATCHES 'ANZ')
  AS vtab1(c1, c2),
 OUTER (SELECT stock_num FROM items
     WHERE item_subtotal > 100)
    AS vtab2(vc1)
 WHERE vtab1.c1 = vtab2.vc1
 ORDER BY c1;
```



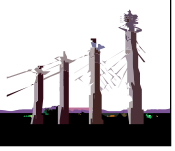
Scope – ANSI Joins - Error Case

- 19820: Informix OUTER JOIN and ANSI JOIN in the same query block.

```
SELECT * FROM
 (SELECT c.stock_num, c.catalog_num
    FROM catalog c, stock s
   WHERE c.stock_num = s.stock_num
     AND c.manu_code not matches 'ANZ')
   AS vtab1(c1,c2),
  OUTER (SELECT stock_num
     FROM (SELECT c.stock_num, c.catalog_num
       FROM catalog c
       LEFT JOIN stock s
       ON c.stock_num = s.stock_num)
        intab(col1,col2), items
    WHERE item_subtotal > 100) AS vtab2(vc1)
WHERE vtab1.c1 = vtab2.vc1
ORDER BY c1;
```

Remote Cases

```
SELECT * FROM (SELECT s.stock_num
  FROM stores@rem_server:stock s, stock_discount sd
 WHERE s.stock_num = sd.stock_num) AS dtab(common_num);

CREATE VIEW stock_view(stock_num) AS
 SELECT * FROM (SELECT s.stock_num FROM
  stores@rem_server:stock s, stock_discount sd
 WHERE s.stock_num = sd.stock_num) AS dtab(common_num);

SELECT * FROM (SELECT c.stock_num, c.catalog_num
      FROM catalog c ,stock s
     WHERE c.stock_num = s.stock_num
       AND c.manu_code NOT MATCHES 'ANZ') AS
   vtab1(c1,c2)
    LEFT JOIN (SELECT i.stock_num
      FROM stores@rem_server:items I
     WHERE i.item_subtotal > 100) AS vtab2(vc1)
    ON vtab1.c1 = vtab2.vc1
ORDER BY c1;
```



QUERY:
------
select * from (select s.stock_num from stores@rem_server:stock s, stock_discount sd where
s.stock_num = sd.stock_num) as dtab(common_num)

Estimated Cost: 9
Estimated # of Rows Returned: 14

1) informix.sd: SEQUENTIAL SCAN

2) informix.s: REMOTE PATH

Remote SQL Request:
select x0.stock_num from stores:"informix".stock x0 where
(x0.stock_num = ?)

NESTED LOOP JOIN

QUERY:
------
select * from stock_view

Estimated Cost: 9
Estimated # of Rows Returned: 14

1) informix.x2: SEQUENTIAL SCAN

2) informix.x1: REMOTE PATH

Remote SQL Request:
select x0.stock_num from stores:"informix".stock x0 where
(x0.stock_num = ?)

NESTED LOOP JOIN

Iterator (Table) Function

- SPL : 'with resume' clause
- C function : 'with (iterator)' clause
- Legacy syntax

```
SELECT * FROM table ( function
  ret_resume(1));
```

- SQL Standard syntax

```
SELECT * FROM table (ret_resume(1));
SELECT * FROM table (ret_resume(1)) AS dtab
  (vcol);
```

- Scope – Similar to derived table

An *iterator function* is a user-defined function that returns to its calling SQL statement several times, each time returning a value. The database server gathers these returned values together in an *active set*. To access a value in the active set, you must obtain it from a database cursor. Therefore, an iterator function is a *cursor function* because it must be associated with a cursor when it is executed.

```
CREATE PROCEDURE ret_resume(num int)
RETURNING integer;
DEFINE retval integer;
 FOREACH SELECT (item_subtotal * num) INTO retval FROM items
      WHERE item_subtotal < 25
  RETURN retval WITH RESUME;
END FOREACH;
END PROCEDURE;
```

```
-- Example - C function using table iterator

CREATE FUNCTION lvargen(arg integer) RETURNS lvarchar WITH (iterator)
EXTERNAL NAME '$USERFUNCDIR/lvar.udr(lvargen))' LANGUAGE C;
```

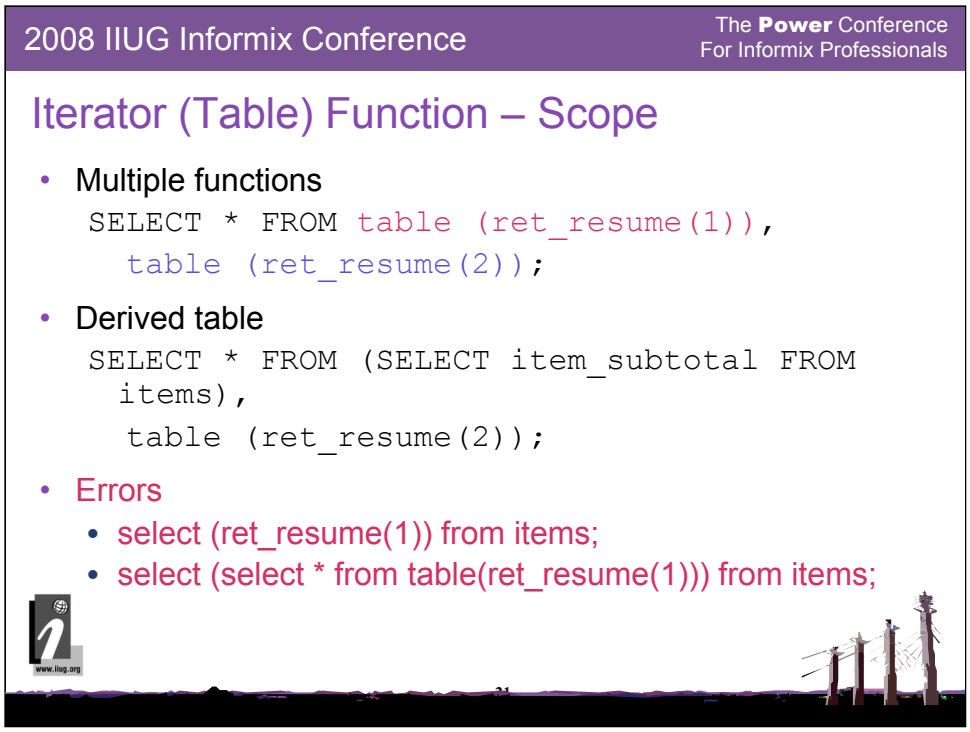


Table function can be used at all locations where table references and <full select> statements are allowed

(Scope is similar to the derived table scope as specified)

Scope

can be used where full select is allowed, projection list, nested cases, views, triggers, stored procedures

Table references

Iterator (Table) Function – Scope

- ANSI Joins

```
SELECT * FROM table ( ret_resume(1)) tab1(c1)
  LEFT JOIN table (ret_resume(2)) tab2(c2)
   ON tab1.c1 = tab2.c2;
CREATE VIEW iter1(vc1, vc2) AS
 SELECT * FROM
  (SELECT item_subtotal, dcol
    FROM items, table (ret_resume(2)) AS
  dtab(dcol));
```

- Remote case

```
SELECT count(*) FROM table
  (db1@$MACHINE2:ret_resume(1));
```

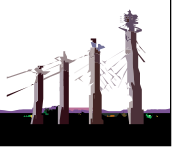


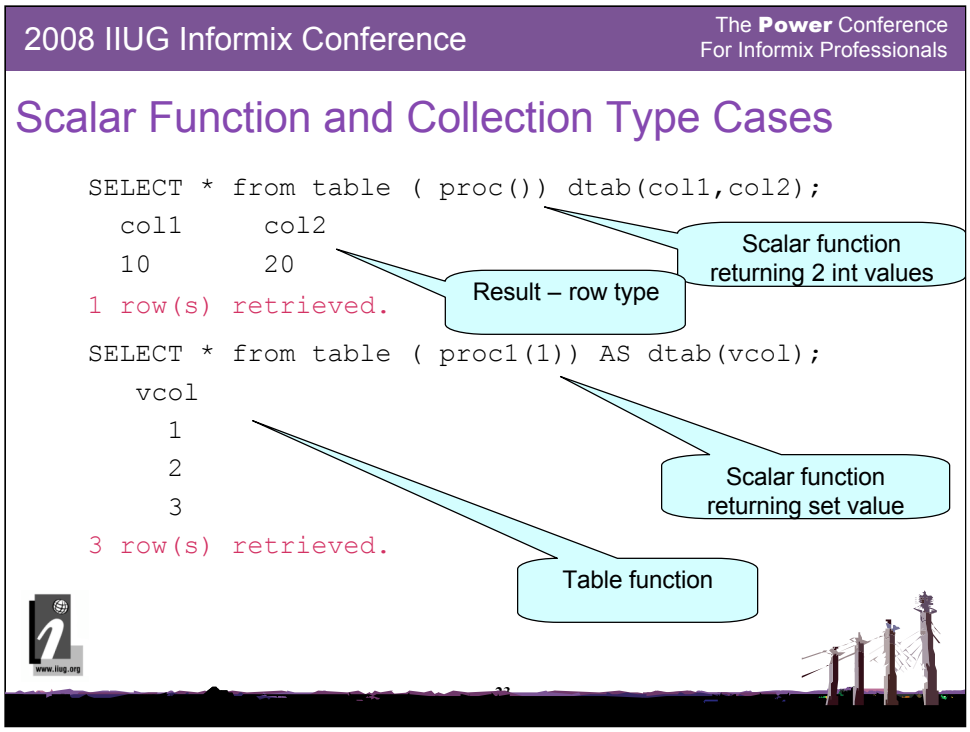
Table function can be used at all locations where table references and <full select> statements are allowed

(Scope is similar to the derived table scope as specified)

Scope

can be used where full select is allowed, projection list, nested cases, views, triggers, stored procedures

Table references




```
CREATE PROCEDURE proc()
RETURNING integer, integer;
DEFINE a1 int;
DEFINE a2 int;
LET a1 = 10;
LET a2 = 20;
RETURN a1, a2;
END PROCEDURE;

CREATE PROCEDURE proc1(num int)
RETURNING set(int not null);
DEFINE s1 set (int not null);
LET s1 = set{1,2,3};
RETURN s1;
END PROCEDURE;
```

Collection Subquery

```
CREATE TABLE t1 ( a set (int not null));
INSERT INTO t1 VALUES (set{1,2,3});
SELECT * FROM (SELECT a from t1);

a  SET{1          ,2          ,3          }

1 row(s) retrieved.

SELECT * FROM table ((SELECT a FROM t1))
    AS vtab(vc);

         vc

          1
          2
          3

3 row(s) retrieved.
```

Derived Table

Collection Subquery

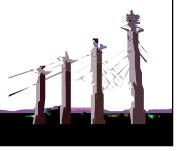
Collection Subquery

```
CREATE TABLE t1 ( a set (int not null), b char(5));
INSERT INTO t1 VALUES (set{1,2,3}, "test");

SELECT * FROM table ((select b from t1));
```

9610: A collection data type must be supplied within this context.

```
SELECT * FROM table ((SELECT a, a FROM t1));
```

574: A subquery has returned not exactly one column.



Collection Subquery

```
CREATE TABLE t1 ( a set (int not null));
INSERT INTO t1 VALUES (set{1,2,3});
INSERT INTO t1 VALUES (set{1,2,3});

SELECT * FROM table ((SELECT a FROM t1))
  AS vtab(vc);
```

284: A subquery has returned not exactly one row.

Derived Table (DT) and Views

- Derived Table

```
SELECT * FROM
   (SELECT stock_num FROM items)
   AS dtab(num);
```

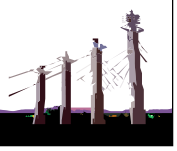
- View

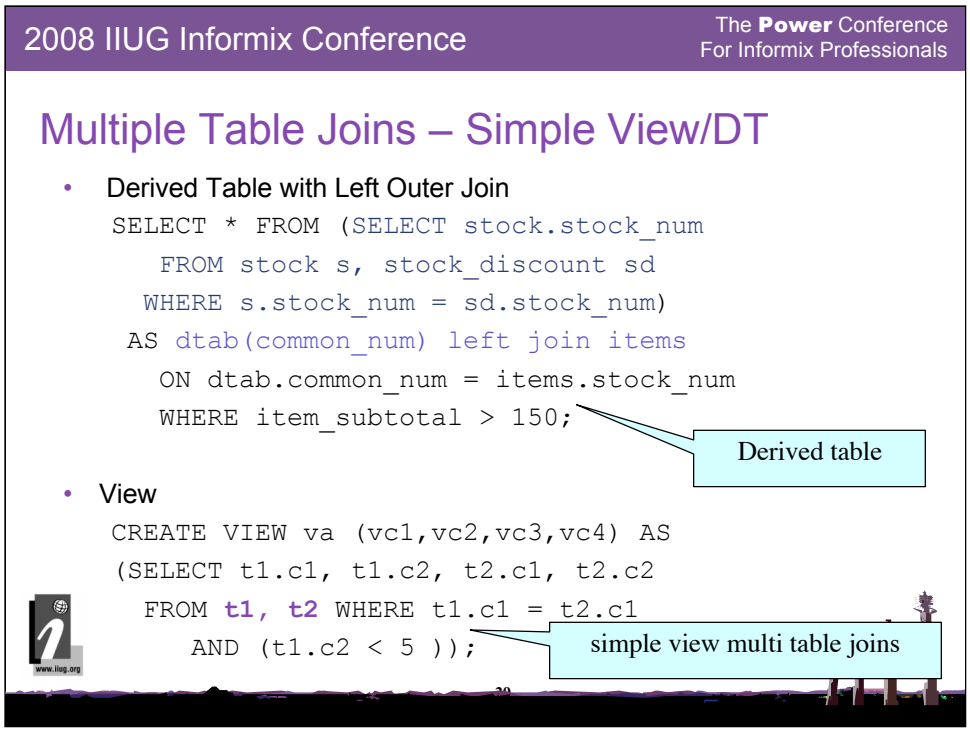
```
CREATE VIEW v1(num) AS
    SELECT stock_num FROM items;
SELECT * FROM v1;
```

- View processing rules are applied
- Temp table rules
  - Aggregate, order by, group by
  - Ansi joins inside Derived table

View Folding Techniques

- Applicable to Derived table
- UNION ALL & multiple table join with ANSI joins
- Terminology : simple view , complex view , view folding , view materialization (temp table)
- Fold union all and multiple table join views / DT
- IFX_FOLDVIEW onconfig parameter : 1 or 0
- Query rewrite cases
  - Multiple table joins – simple view / DT
    - Informix join main query
    - ANSI join main query





QUERY:
------
select * from (select stock.stock_num from stock, stock_discount where
stock.stock_num = stock_discount.stock_num) as dtab(common_num) left join
items on dtab.common_num = items.stock_num where item_subtotal > 150

Estimated Cost: 13
Estimated # of Rows Returned: 10

1) informix.stock_discount: SEQUENTIAL SCAN

2) informix.items: INDEX PATH

Filters: informix.items.item_subtotal > $150.00

(1) Index Keys: stock_num manu_code unit (Serial, fragments: ALL)
Lower Index Filter: informix.items.stock_num = informix.stock_discount.s
tock_num
NESTED LOOP JOIN
3) informix.stock: INDEX PATH

(1) Index Keys: stock_num manu_code unit (Key-Only) (Serial, fragments: A
LL)
Lower Index Filter: informix.stock.stock_num = informix.stock_discount.s
tock_num
NESTED LOOP JOIN

Multiple Table Joins – Simple View/DT

```
SELECT va.vc1, t3.c1 FROM va
  LEFT JOIN t3 ON va.vc1 = t3.c1
```

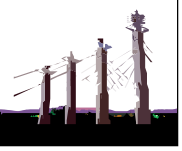
1) (Temp Table For View): SEQUENTIAL SCAN

2) informix.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: **(Temp Table For View).vc1** = informix.t3.c1

old plan

ON-Filters:(Temp Table For View).vc1 = informix.t3.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)



Multiple Table Joins – Simple View/DT

1) informix.t1: SEQUENTIAL SCAN

Filters: informix.t1.c2 < 5

New plan with fold view

2) informix.t2: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: informix.t1.c1 = informix.t2.c1
NESTED LOOP JOIN

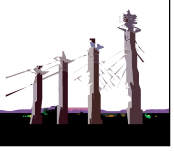
3) informix.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: informix.t1.c1 = informix.t3.c1

ON-Filters:informix.t1.c1 = informix.t3.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)

Simple View/DT Folding Cases

```
SELECT va.vc1, t3.c1 FROM va RIGHT JOIN t3
   ON va.vc1 = t3.c1;

SELECT va.vc1, t3.c1 FROM va FULL JOIN t3
   ON va.vc1 = t3.c1;

SELECT t1.c2, va.vc4, t3.c2 FROM t1
  RIGHT JOIN (va LEFT JOIN t3
          ON va.vc3 = t3.c1)
    ON (t1.c1 = va.vc3
    AND va.vc3 < 2)
  WHERE t1.c1 = 1;

SELECT t3.c1 FROM (t3 RIGHT OUTER JOIN
    (t2 RIGHT OUTER JOIN va ON t2.c1=vc1)
    ON t3.c1=t2.c1);
```



CREATE VIEW va (vc1, vc2 ,vc3, vc4) AS
(SELECT t1.c1, t1.c2, t2.c1, t2.c2 FROM t1, t2
  WHERE t1.c1 = t2.c1
  AND (t1.c2 < 5));

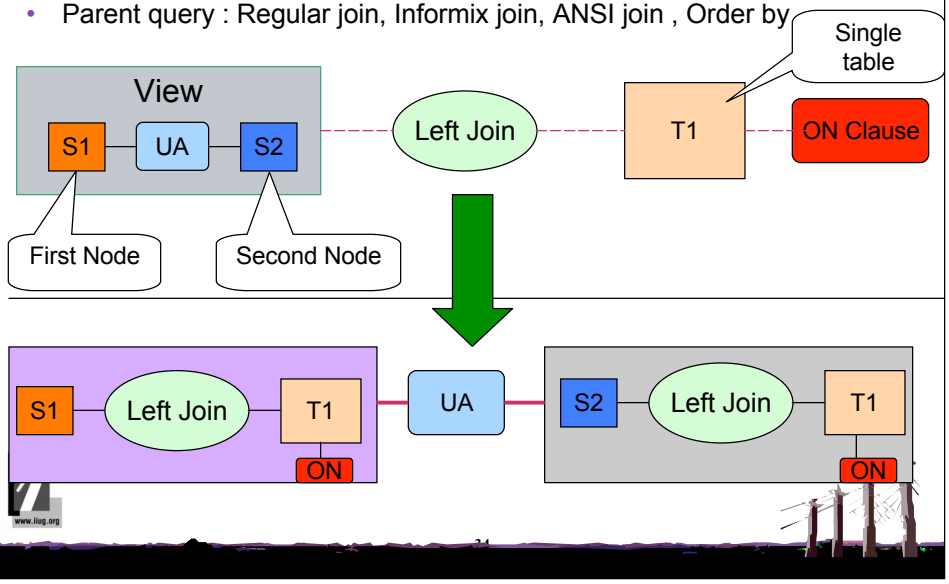
Simple View/DT Folding Cases

```
SELECT t3.c1 FROM (t3 LEFT OUTER JOIN
       (t2 LEFT OUTER JOIN va ON t2.c1=vc1)
   ON t3.c1=t2.c1);

SELECT vc1, t1.c2 FROM t1 LEFT OUTER JOIN va
    ON t1.c1=va.vc3
  WHERE t1.c2 < 3
UNION
SELECT vc1, t3.c2 FROM t3 LEFT OUTER JOIN va
    ON t3.c1=va.vc3
  WHERE t3.c2 < 2;
```

CREATE VIEW va (vc1, vc2 ,vc3, vc4) AS
(SELECT t1.c1, t1.c2, t2.c1, t2.c2 FROM t1, t2
  WHERE t1.c1 = t2.c1
  AND (t1.c2 < 5));

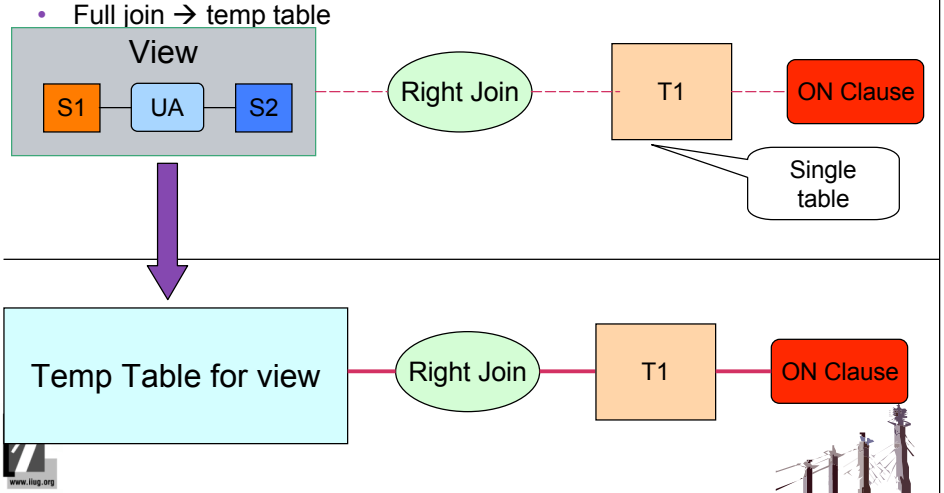
UNION ALL View/DT Query Rewrite

- Parent query : Regular join, Informix join, ANSI join , Order by



UNION ALL View/DT – Cases

- Right join (view/DT on inner (subservient side)) → temp table
- Full join → temp table



UNION ALL View – ANSI Left Join

```
CREATE VIEW v0 (vc1,vc2,vc3,vc4) AS (
  SELECT c1, c2, c3, c3-1 FROM t1 WHERE c1 < 5
 UNION ALL
  SELECT c1, c2, c3, c3-1 FROM t2 WHERE c1 > 5);

SELECT v0.vc1, t3.c1 FROM v0 LEFT JOIN t3
  ON v0.vc1 = t3.c1;
```

1) informix.t1: SEQUENTIAL SCAN

Filters: informix.t1.c1 < 5

New plan with fold view

2) informix.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: informix.t1.c1 = informix.t3.c1

ON-Filters:informix.t1.c1 = informix.t3.c1
NESTED LOOP JOIN(LEFT OUTER JOIN

UNION ALL View – ANSI Left Join

Union Query:
---

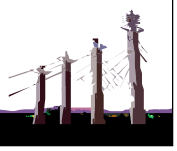
1) informix.t2: SEQUENTIAL SCAN

Filters: informix.t2.c1 > 5

2) informix.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: informix.t2.c1 = informix.t3.c1

ON-Filters:informix.t2.c1 = informix.t3.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)



UNION ALL View – ANSI Right Join

```
SELECT t3.c1, v0.vc1 FROM t3 RIGHT JOIN v0
   ON v0.vc1 = t3.c1;
```

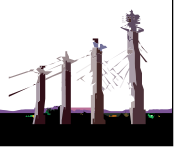
1) informix.t1: SEQUENTIAL SCAN

Filters: informix.t1.c1 < 5

2) informix.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: informix.t1.c1 = informix.t3.c1

ON-Filters:informix.t1.c1 = informix.t3.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)



UNION ALL View – ANSI Right Join

Union Query:
---

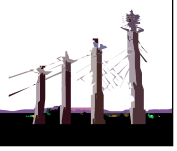
1) informix.t2: SEQUENTIAL SCAN

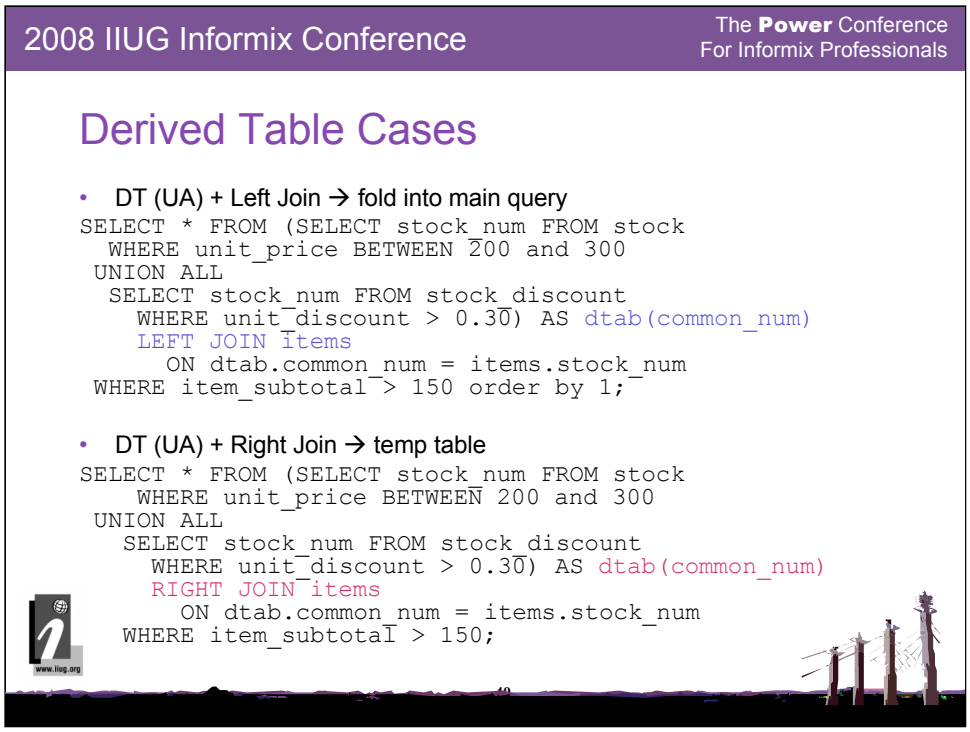
Filters: informix.t2.c1 > 5

2) informix.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: informix.t2.c1 = informix.t3.c1

ON-Filters:informix.t2.c1 = informix.t3.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)





QUERY:
------
select * from (select stock_num from stock where unit_price between 200 and 300
union all
select stock_num from stock_discount where unit_discount > 0.30
) as dtab(common_num) left join
items on dtab.common_num = items.stock_num where item_subtotal > 150
order by 1

Estimated Cost: 21
Estimated # of Rows Returned: 9
Temporary Files Required For: Order By

1) informix.stock: SEQUENTIAL SCAN

Filters: (informix.stock.unit_price <= $300.00 AND informix.stock.unit_p
rice >= $200.00)

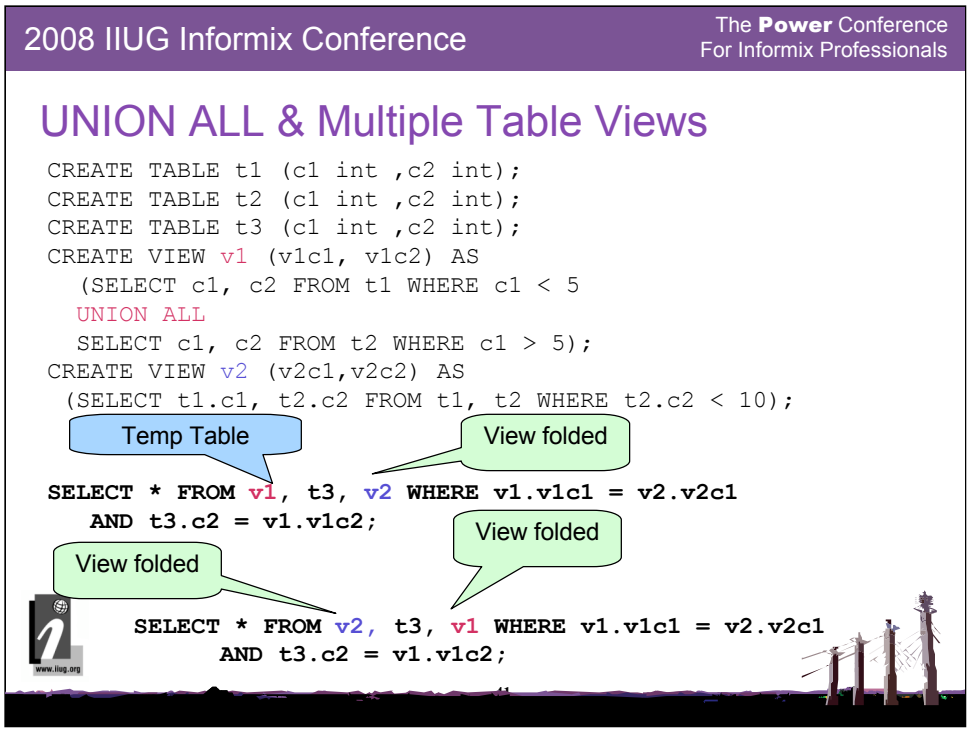
2) informix.items: INDEX PATH

Filters: informix.items.item_subtotal > $150.00

(1) Index Keys: stock_num manu_code unit (Serial, fragments: ALL)
Lower Index Filter: informix.stock.stock_num = informix.items.stock_num
NESTED LOOP JOIN

Union Query:

Temporary Files Required For: Order By



CASE 1 : **select * from v1, t3, v2 where v1.v1c1 = v2.v2c1 and t3.c2 = v1.v1c2;**

QUERY:

---

create view "informix".v1 (v1c1,v1c2) as select x0.c1 ,x0.c2 from "informix".t1 x0 where (x0.c1 < 5) union all select x1.c1 ,x1.c2 from "informix".t2 x1 where (x1.c1 > 5);

Estimated Cost: 4
Estimated # of Rows Returned: 2

1) informix.t1: SEQUENTIAL SCAN

Filters: informix.t1.c1 < 5

Union Query:

---

1) informix.t2: SEQUENTIAL SCAN

Filters: informix.t2.c1 > 5

QUERY:

---

select * from v1, t3, v2 where v1.v1c1 = v2.v2c1 and t3.c2 = v1.v1c2

Estimated Cost: 16
Estimated # of Rows Returned: 2

1) informix.t2: SEQUENTIAL SCAN

Filters: informix.t2.c2 < 10

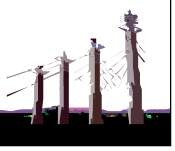
2) informix.t1: SEQUENTIAL SCAN

UNION ALL & Multiple Table Views

```
CREATE VIEW v1 (vc2) AS
  SELECT t3.c2 FROM t3 WHERE t3.c2 > 10
 UNION ALL
  SELECT t4.c2 FROM t3, t4 WHERE t4.c2 < t3.c2 ;

CREATE VIEW v2 (vc2) AS
  SELECT t4.c2 FROM t3, t4 WHERE t4.c2 < t3.c2 ;

SELECT v1.vc2 FROM v1 LEFT JOIN t1
 ON v1.vc2 = t1.c1
UNION ALL
SELECT v1.vc2 FROM v1 LEFT JOIN t2
 ON v1.vc2 = t2.c1 ;

SELECT v2.vc2 FROM v2 LEFT JOIN t1
 ON v2.vc2 = t1.c1
UNION ALL
SELECT v2.vc2 FROM v2 LEFT JOIN t2
 ON v2.vc2 = t2.c1 ;
```

Temp Table

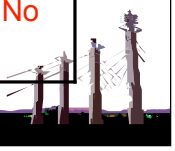
View folded



UNION ALL View - Summary

D : view has dominant (outer table) role in main query e.g. V1 left join T1
S : view has subservient (inner table) role in main query e.g. T1 left join V1
Main query LOJ : main query has left outer join
Main query ROJ : main query has right outer join
Main query FOJ : main query has full outer join

Query type	Main query LOJ		Main query ROJ		Main query FOJ	
View type	D	S	D	S	D	S
Simple	Yes	Yes	Yes	Yes	Yes	Yes
Union all	Yes	No	Yes	No	No	No



Tips

- Avoid legacy syntax
- Compare with view
- Sqexplain
- Order by & nested derived table cases
- ANSI joins inside derived table (or view)
- UNION vs. UNION ALL
- Text for view, trigger & SPL
- Temp table creation with union all view
  - View has aggregate, group by, order by, union, distinct , outer joins (ANSI or Informix)
  - Parent query has union , union all , multiple views

References

- IDS 11.10 Info Center - http://publib.boulder.ibm.com/infocenter/idshelp/v111/index.jsp
- Informix product family - http://www.ibm.com/software/data/informix

Session ####

IDS Cheetah: Derived Table and View Processing Techniques

Ajaykumar Gupte
IBM
gupte@us.ibm.com