# Advanced SQL Set Processing

Rob Bestgen
Db2 for i Consultant
bestgen@us.ibm.com

# Thinking in Sets

- A carefully crafted SELECT statement is basically a contract between you and the database

- You are precisely describing the inputs and the contents and format of the result set

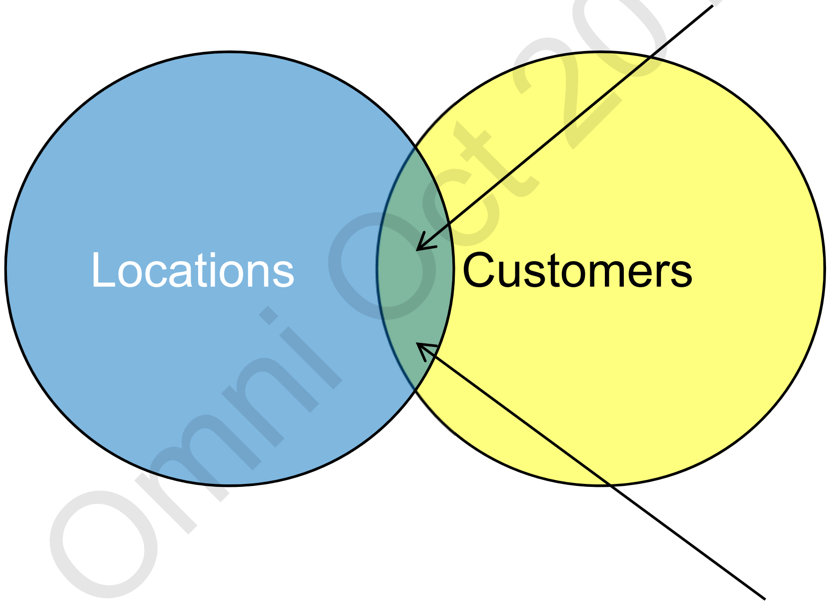
- It is up to the database to choose the most efficient way of providing your result set

- Traditional languages using Record Level Access (RLA) are very row based in their approach

- SQL works best when you think in terms of sets

# SQL – Working with Sets

**Question: number of customers in China?**

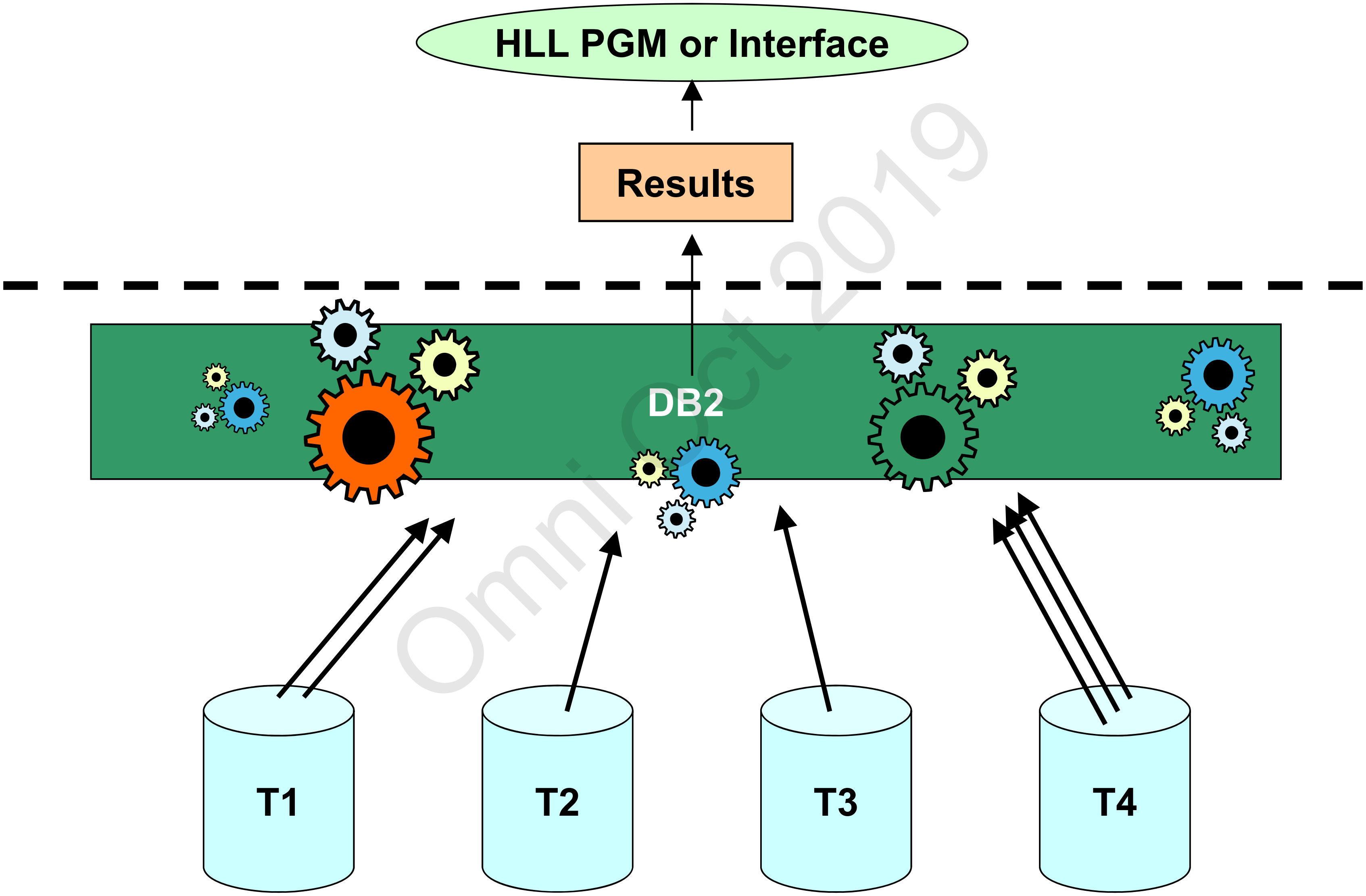


Count what’s here

Return what’s here

**Question: who are the customers in China?**

# Data Centric Overview – Leveraging the Database



# Programming in Sets

# A working, procedural based program

```
DECLARE CURSOR cursor1 FOR
SELECT cust_id, prod_id, quantity, amount
FROM orders
WHERE transaction_date = :v_date;

OPEN cursor1;

DO
  FETCH cursor1
    INTO :v_custid, :v_prodid, :v_qty, :v_amt;

  SELECT cust_name, cust_address
    INTO :v_name, :v_address
   FROM customers
   WHERE custid= :v_custid;

  SELECT prod_name INTO :v_prodname
   FROM products
   WHERE prodid= :v_prodid;

  INSERT INTO daily_thank_you_log VALUES
    ( :v_name, :v_address, :v_prodname, :v_qty, :v_amt);

UNTIL ( no more data );

CLOSE cursor1;
```

**What are we trying to do??**

**Express it in 'business' terms**

**Generate a list of customer orders for a given day so we can send them 'thank you' emails**

# Our ‘program’ revisited

```
DECLARE CURSOR cursor1 FOR
SELECT cust_id, prod_id, quantity, amount
FROM orders
WHERE transaction_date = :v_date;

OPEN cursor1;

DO
  FETCH cursor1
    INTO :v_custid, :v_prodid, :v_qty, :v_amt;

  SELECT cust_name, cust_address
    INTO :v_name, :v_address
   FROM customers
   WHERE custid= :v_custid;

  SELECT prod_name INTO :v_prodname
   FROM products
   WHERE prodid= :v_prodid;

  INSERT INTO daily_thank_you_log VALUES
    ( :v_name, :v_address, :v_prodname, :v_qty, :v_amt);

UNTIL ( no more data );

CLOSE cursor1;
```

Much better!!

```
INSERT INTO daily_thank_you_log
SELECT c.cust_name, c.cust_address,
          p.prod_name,
          o.quantity, o.amount
FROM orders o
   INNER JOIN customers c
      ON c.custid= o.cust_id
    INNER JOIN products p
      ON p.prodid = o.prod_id
WHERE o.transaction_date = :v_date
```

# Sets and SQL

## SQL is commonly used in the single SELECT form

- SELECT … FROM… WHERE

## And it is very powerful

- Can do join, filtering, projection….

## But SQL becomes even more powerful when combining more than one SELECT

- Can leverage more set thinking!

# Set Operators

Use **Set operators** to combine results from multiple subselects

- **UNION** – combine into a distinct result set
- **UNION ALL** – append result sets
- **INTERSECT** – return only distinct rows found in both result sets
- **EXCEPT** – return distinct rows from first subselect not found in second subselect

Examples:

- Return all (distinct) rows that are in t1, but not t2

```
(SELECT cusnum FROM orders2018)
 EXCEPT
(SELECT cusnum FROM orders2019)
```

- All (distinct) rows that exist in both t1 & t2

```
(SELECT cusnum FROM orders2018)
 INTERSECT
(SELECT cusnum FROM orders2019)
```

# Subselects

**Subselect**, as the name implies, is a:

1. SELECT statement
2. within ('sub') an SQL statement

Subselects are the underpinning for many advanced SQL techniques

Strong suggestion:

Qualify your column references!

# Subselect dependence

Subselects can be **independent** or **dependent**

- Independent – aka non-Correlated
  - Subselect (along with any of its inner components) is autonomous
  - Example:

```
SELECT e.last_name FROM employee e
WHERE  deptnum IN
    (SELECT l.deptno FROM location l WHERE l.name = ‘Indy’)
```

- Dependent – aka Correlated
  - Dependent on outer row for evaluation because of a reference
  - Example:

```
SELECT last_name FROM employee x
WHERE x.salary >
    (SELECT AVG(y.salary) FROM employee y
            WHERE x.deptnum = y.deptnum )
```

## Subquery example:

Return the details of the latest order for each of my customers

```
SELECT C.CUSTNAME, O.ORDERDATE, I.ITEMNAME, O.QUANTITY
FROM ORDERS O
  INNER JOIN CUSTOMER C ON O.CUSTNO = C.CUSTNO
  INNER JOIN ITEMS I ON O.ITEMNUM = I.ITEMNUM
WHERE O.ORDERDATE =
               (SELECT MAX(O2.ORDERDATE)
                FROM ORDERS O2
                WHERE O.CUSTNO = O2.CUSTNO)
```

## Row subquery

BTW, you can compare more than a single column with an IN subquery:

```
SELECT contact_name, contact_phone FROM contact o
 WHERE (o.contact_state, o.contact_id) IN (
      SELECT c.state, c.custid FROM customer c)
```

# Derived Tables

# Common Table Expressions

# Views

# But First, some VALUES

## VALUES –

- A table-less result set. A way to produce an answer set out of thin air
- You've probably used it in INSERT statements

  ```
  INSERT INTO mytab VALUES(1,2,3)
  ```

- But it can also be used as a source of data in most any query

  ```
  SELECT * FROM TABLE(VALUES(1,2,3)) X(C1, C2, C3)
  ```

- Including multiple rows

  ```
  SELECT * FROM TABLE(VALUES(1,2,3),(4,5,6)) X(C1, C2, C3)
  ```

It can be a very handy tool in the toolbox

# Derived Tables

**Derived Tables** are subselects embedded in a FROM clause that produce a set of rows

- A virtual table

```
SELECT e.name as mgrname, d1.deptno as dept,
       d1.empcount as numemployees
FROM employees e inner join
   (SELECT deptno, COUNT(*) as empcount
    FROM employee GROUP BY deptno) d1
ON e.deptno = d1.deptno
```

## Derived Tables…

### A Derived Table can be **laterally correlated***

- Its results are dependent on a table to the ‘left’
- Must use the LATERAL keyword
- Good way to ‘pivot’ multiple columns into rows

```
CREATE TABLE HOME_LOAN_APPS
(NAME VARCHAR(128),
 APP_NBR INT,
 LOC CHAR(5),
STRCTR CHAR(5),
AGE CHAR(5));
```

```
SELECT A.NAME, A.APP_NBR,
          L.PROPERTY_ASPECT , L.SCORE
FROM HOME_LOAN_APPS A CROSS JOIN
   LATERAL
   (SELECT
     PROPERTY_ASPECT, SCORE
     FROM TABLE
      (VALUES
        (‘Location', A.LOC),
        (‘Structures' , A.STRCTR),
        (‘Age Of Buildings', A.AGE)
      ) E(PROPERTY_ASPECT,SCORE)
   ) L
```

* Can be used with inner, left outer joins, left exception, and cross joins

# Common Table Expressions (CTEs)

## **Common Table Expressions** (CTEs) produce a result set

- Virtual temporary table – avoid physical work tables
- Can be referenced multiple times
- Divides a report into logical steps
- Can be used to perform Recursive SQL!

```
WITH staff (deptno, empcount) AS
(SELECT deptno, COUNT(*) FROM employee
 WHERE division = :div_var GROUP BY deptno)
SELECT deptno, empcount FROM staff
WHERE empcount >
        (SELECT AVG(empcount) FROM staff)
```

# CTEs – Thinking in Sets …

- What if you want a list of customers who were in the “top 10” for two consecutive years?  Think in sets …

```
WITH top10_2017 (customer_name, total_sales) AS
       (SELECT customer_name, SUM(sales) FROM sales
                           WHERE year=2017
                           GROUP BY customer_name
                           ORDER BY SUM(sales) DESC
                           FETCH FIRST 10 ROWS ONLY) ,
          top10_2018 AS
       (SELECT customer_name, SUM(sales) total_sales FROM sales
                           WHERE year=2018
                           GROUP BY customer_name
                           ORDER BY SUM(sales) DESC
                           FETCH FIRST 10 ROWS ONLY)

 SELECT y1.customer_name,
             y1.total_sales AS sales2017, y2.total_sales AS sales2018
             FROM top10_2017 y1 INNER JOIN top10_2018 y2
              ON y1.customer_name = y2.customer_name
```

Which column naming approach is better?

# CTEs: **Recursive (Hierarchical)** SQL

Perform a **Recursive Query** with CTEs!

- Useful for navigating tables where rows are inherently related to other rows in same table
  - Bill of Materials, Organizational Hierarchies, etc…

```
WITH emp_list (level, empid, name) AS
      (SELECT 1, empid, name  FROM emp
        WHERE name = 'Carfino‘
      UNION ALL
        SELECT o.level + 1, next_layer.empid, next_layer.name
          FROM emp as next_layer, emp_list o
           WHERE o.empid = next_layer.mgrid )
SELECT level, name FROM emp_list
```

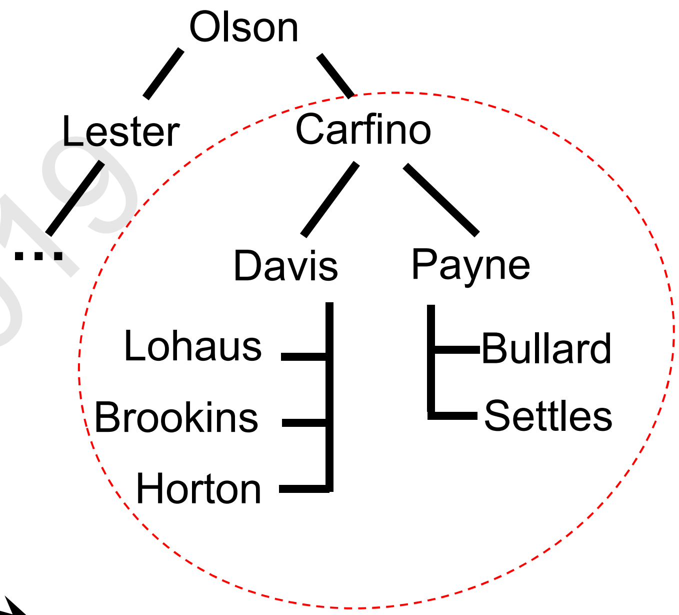
1 - Initializing the query

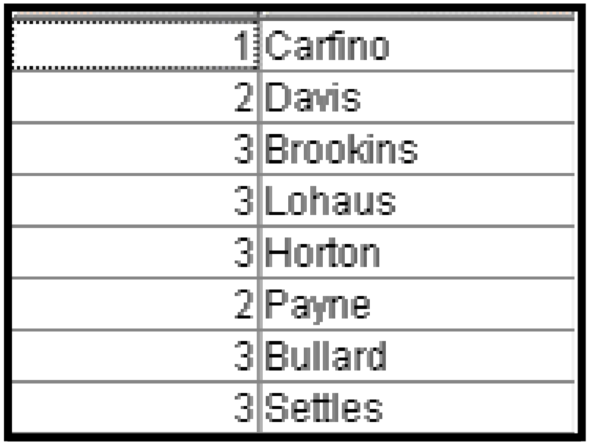
2 – Recursive reference
to the next level

3 – Start the query &
return final results

# Recursive SQL simple case alternative
# - CONNECT BY

```
SELECT LEVEL, name
  FROM emp
 START WITH name = 'Carfino'
 CONNECT BY mgrid = PRIOR empid
```





| | |
|---|---|
| 1 | Carfino |
| 2 | Davis |
| 3 | Brookins |
| 3 | Lohaus |
| 3 | Horton |
| 2 | Payne |
| 3 | Bullard |
| 3 | Settles |

```
WITH emp_list (level, empid, name) AS
      SELECT 1, empid, name  FROM emp
         WHERE name = 'Carfino‘
       UNION ALL
      SELECT o.level+1, next_layer.empid, next_layer.name
         FROM emp as next_layer, emp_list o
         WHERE o.empid = next_layer.mgrid )
 SELECT level, name FROM emp_list
```

## CTEs: **Recursive** SQL – Hierarchical SQL

# RCTE vs. CONNECT BY. Which is better?

Both have advantages:

- RCTE – More complex definitions allowed
- CONNECT BY – more options to control circular loops and depth

Use the one that ‘speaks’ to you.

# CTEs: **Recursive** SQL

How about generating sales info for each day of this month

```
WITH month_days (d, DayOfMonth) AS
   (  VALUES(CURRENT DATE – (DAY(CURRENT DATE) – 1 ) DAYS, 1)
     UNION ALL
       SELECT d+1 DAYS, DAY(d+1 DAYS) FROM month_days
       WHERE MONTH(d+1 DAYS) = MONTH(CURRENT DATE)
  )
SELECT s.order_date, sum(s.sales) as totsales
FROM sales s INNER JOIN month_days m
ON s.order_date = m.d
```

# Logical Separation Using Views

# Remember to Use SQL Views

An SQL view provides many advantages

- Encapsulate common ‘patterns’ in queries into a single location
- SQL views provide a way to **logically** separate the application from the physical database layout
- Views are performance neutral. They neither hurt (nor help) performance. The optimizer merges the view definition with the query

```
CREATE VIEW active_employee AS
(SELECT d1.* FROM employee d1
 WHERE d1.deptno IN
 (SELECT p.deptnum
  FROM projects p
  where status=‘active’))
```

```
.
SELECT *
FROM active_employee d1
WHERE d1.empid = ?
.
.
.
SELECT count(*)
FROM active_employee d1
```

*Thank you!*