Session Survey

- Sign in to the Online Session Guide (www.common.org/sessions)
- Go to your personal schedule.
- Click on the session that you attended.
- Click on the Feedback Survey button located above the abstract.

Completing session surveys helps us plan future programming and provides feedback used in speaker awards.

Thank you for your participation.
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?

Question: who are the customers in China?

Count what’s here

Return what’s here
Data Centric Overview – Leveraging the Database

HLL PGM or Interface

Results

DB2

T1

T2

T3

T4
Programming in Sets
A working, procedural based program

DECLARE CURSOR cursor1 FOR
SELECT cust_id, prod_id, quantity, amount
FROM orders
WHERE transaction_date = '2018.05.01';

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
DECLARE CURSOR cursor1 FOR
SELECT cust_id, prod_id, quantity, amount FROM orders
WHERE transaction_date = '2019.05.01';

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;

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.prodid
WHERE o.transaction_date = '2019.05.01'

Much better!!
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 orders2015) 
  
  EXCEPT 
  
  (SELECT cusnum FROM orders2016)

- All (distinct) rows that exist in both t1 & t2
  
  (SELECT cusnum FROM orders2015) 
  
  INTERSECT 
  
  (SELECT cusnum FROM orders2016)
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
Subselect dependence

Subselects can be independent or dependent

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

```sql
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

```
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) 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!

```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 MAX(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 (customer_name, total_sales) AS
  (SELECT customer_name, SUM(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
CTEs: Recursive (Hierarchical) SQL

Perform a **Recursive Query** with CTEs!

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

```sql
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

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

SELECT LEVEL, name
FROM emp
START WITH name = 'Carfino'
CONNECT BY mgrid = PRIOR empid
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 the days of the month

```sql
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

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

```sql
SELECT *
FROM active_employee d1
WHERE d1.empid = ?

SELECT count(*)
FROM active_employee d1
```
GROUPING SETS
And conditional aggregation
Grouping Sets

- Many BI applications and OLAP tools involve multi-dimensional aggregate views of transaction data
  - Users need to view results at multiple levels
  - Users need to view result data from different perspective

  **EXAMPLE:**
  ```sql
  SELECT country, region, store, product, SUM(sales)
  FROM trans
  GROUP BY country, region, store, product
  
  - Result does not have dimensions
  
- **Grouping Sets** allow data to be grouped in multiple ways with a single SQL request
  - ROLLUP
  - CUBE
  - GROUPING SETS

- Operators can be mixed together

- Use the GROUPING function to determine which group is being represented
**Grouping Sets - ROLLUP**

**ROLLUP** is an extension to the GROUP BY clause that produces a hierarchy set containing sub-total rows in addition to the “regular” grouped rows

- Does multiple ‘GROUP BY’s in one statement

**Example:**

```sql
SELECT country, region, SUM(sales)
FROM trans
GROUP BY ROLLUP (country, region)
```

- **ROLLUP**(Country, Region) will result in the data being summarized at the following levels
  - (Country, Region)
  - (Country)
  - ( ) << represents Grand Total
ROLLUP Output Example

SELECT country, region, SUM(sales)
FROM trans
GROUP BY ROLLUP (country, region)

<table>
<thead>
<tr>
<th>Country</th>
<th>Region</th>
<th>Sum(Sales)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canada</td>
<td>-</td>
<td>100,000</td>
</tr>
<tr>
<td>Canada</td>
<td>NW</td>
<td>100,000</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>-</td>
<td>3,250,000</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>NE</td>
<td>450,000</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>NW</td>
<td>940,000</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>SE</td>
<td>550,000</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>SW</td>
<td>1,310,000</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>3,350,000</td>
</tr>
</tbody>
</table>
Grouping Sets - CUBE

**CUBE** is an extension to the GROUP BY clause that produces all dimensions of a group definition.

**Example:**

```sql
SELECT country, region, SUM(sales)
FROM trans
GROUP BY CUBE (country, region)
```

- **CUBE(Country, Region)** will result in the data being summarized at the following levels:
  - (Country, Region)
  - (Country)
  - (Region)
  - ( ) **<<** represents Grand Total
GROUPING SETS is an extension to the GROUP BY clause that produces the dimensions specified in the set

- Useful if you don’t want all dimensions produced by ROLLUP or CUBE

**Example:**

```sql
SELECT country, region, store, SUM(sales)
FROM trans
GROUP BY GROUPING SETS((country, region), (country, store))
```

- GROUPING SETS((country, region), (country, store)) will result in the data being summarized at the following levels
  - (Country, Region)
  - (Country, Store)
OLAP Functions
What are OLAP Specifications?

- On-line Analytical Processing functions providing the ability to return ranking, row numbering, aggregates and more as part of a SQL query result
  - Also referred to as window functions

- Can be specified as part of the select-list or ORDER BY clause

- OLAP function categories:
  - Ordered OLAP specifications
  - Numbering specifications
  - Aggregation specifications
  - Super groups
OLAP example with ROW_NUMBER

- ROW_NUMBER can be used to assign a number to query results

```
SELECT ROW_NUMBER() OVER(ORDER BY workdept, lastname) AS Nbr, lastname, salary
FROM employee
ORDER BY workdept, lastname
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>LASTNAME</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HAAS</td>
<td>52750.00</td>
</tr>
<tr>
<td>2</td>
<td>HEMMINGER</td>
<td>46500.00</td>
</tr>
<tr>
<td>3</td>
<td>LUCCHESI</td>
<td>46500.00</td>
</tr>
<tr>
<td>4</td>
<td>O’CONNELL</td>
<td>29250.00</td>
</tr>
<tr>
<td>5</td>
<td>ORLANDO</td>
<td>29250.00</td>
</tr>
</tbody>
</table>
OLAP Function Syntax Overview

- **All of the OLAP functions allow the window partition clause and the window order clause**
  - The window **order** clause is required in most cases while the window **partition** clause is optional

- **OVER** specifies the definition of the window over the result set

- The **window-partition-clause** defines the boundaries between the partitions within the window

- **window-order-clause** defines the sort order of the rows within a partition
  - Note: this does not define the ordering of the query’s final result set!
RANK and DENSE_RANK

- RANK and DENSE_RANK allow for highlighting data attribute – independent of the result set sorting

```sql
SELECT empno, lastname, salary+bonus AS TOTAL_SALARY,
    RANK() OVER (ORDER BY salary+bonus DESC) AS Salary_Rank
FROM employee
WHERE salary + bonus > 30000
ORDER BY lastname
```

Dense_Rank() Output

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>LASTNAME</th>
<th>TOTAL_SALARY</th>
<th>SALARY_RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>000050</td>
<td>Geyer</td>
<td>40975.00</td>
<td>5</td>
</tr>
<tr>
<td>000010</td>
<td>Haas</td>
<td>53750.00</td>
<td>1</td>
</tr>
<tr>
<td>200010</td>
<td>Hemminger</td>
<td>47500.00</td>
<td>2</td>
</tr>
<tr>
<td>000090</td>
<td>Henderson</td>
<td>30350.00</td>
<td>11</td>
</tr>
<tr>
<td>200220</td>
<td>John</td>
<td>30440.00</td>
<td>9</td>
</tr>
<tr>
<td>000030</td>
<td>Kwan</td>
<td>39050.00</td>
<td>6</td>
</tr>
<tr>
<td>000110</td>
<td>Lucchesi</td>
<td>47400.00</td>
<td>3</td>
</tr>
<tr>
<td>000220</td>
<td>Lutz</td>
<td>30440.00</td>
<td>9</td>
</tr>
<tr>
<td>000070</td>
<td>Pulaski</td>
<td>36870.00</td>
<td>7</td>
</tr>
<tr>
<td>000060</td>
<td>Stern</td>
<td>32750.00</td>
<td>8</td>
</tr>
<tr>
<td>000020</td>
<td>Thompson</td>
<td>42050.00</td>
<td>4</td>
</tr>
</tbody>
</table>
Aggregate functions (V7R3)

- Aggregate functions can have the windowing clause
  - SUM, COUNT, MIN, MAX, AVG...
  - Adding the OVER windowing clause is what makes these functions OLAP functions

```sql
SELECT empno, workdept, lastname, salary+bonus AS TOTAL_SALARY, AVG(salary+bonus) OVER (PARTITION BY workdept) AS Avg_for_dept
FROM employee
ORDER BY salary+bonus DESC
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>WORKDEPT</th>
<th>LASTNAME</th>
<th>TOTAL_SALARY</th>
<th>AVG_FOR_DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>200010</td>
<td>A00</td>
<td>HEMMINGER</td>
<td>47500</td>
<td>31140</td>
</tr>
<tr>
<td>000110</td>
<td>A00</td>
<td>LUCHESSEI</td>
<td>47400</td>
<td>31140</td>
</tr>
<tr>
<td>000020</td>
<td>B01</td>
<td>THOMPSON</td>
<td>42050</td>
<td>42050</td>
</tr>
<tr>
<td>000050</td>
<td>E01</td>
<td>GEYER</td>
<td>40975</td>
<td>40975</td>
</tr>
<tr>
<td>000030</td>
<td>C01</td>
<td>KWAN</td>
<td>39050</td>
<td>30347</td>
</tr>
<tr>
<td>000070</td>
<td>D21</td>
<td>PULASKI</td>
<td>36870</td>
<td>26168</td>
</tr>
<tr>
<td>000060</td>
<td>D11</td>
<td>STERN</td>
<td>32750</td>
<td>25647</td>
</tr>
<tr>
<td>000220</td>
<td>D11</td>
<td>LUTZ</td>
<td>30440</td>
<td>25647</td>
</tr>
<tr>
<td>200220</td>
<td>D11</td>
<td>JOHN</td>
<td>30440</td>
<td>25647</td>
</tr>
<tr>
<td>000090</td>
<td>E11</td>
<td>HENDERSON</td>
<td>30350</td>
<td>21434</td>
</tr>
<tr>
<td>000120</td>
<td>A00</td>
<td>O'CONNELL</td>
<td>29850</td>
<td>31140</td>
</tr>
<tr>
<td>200120</td>
<td>A00</td>
<td>ORLANDO</td>
<td>29850</td>
<td>31140</td>
</tr>
<tr>
<td>000240</td>
<td>D21</td>
<td>MARINO</td>
<td>29360</td>
<td>26168</td>
</tr>
<tr>
<td>200240</td>
<td>D21</td>
<td>MONTEVERDE</td>
<td>29360</td>
<td>26168</td>
</tr>
<tr>
<td>000140</td>
<td>C01</td>
<td>NICHOLLS</td>
<td>29020</td>
<td>30347</td>
</tr>
<tr>
<td>200140</td>
<td>C01</td>
<td>NATZ</td>
<td>29020</td>
<td>30347</td>
</tr>
<tr>
<td>000200</td>
<td>D11</td>
<td>BROWN</td>
<td>28340</td>
<td>25647</td>
</tr>
<tr>
<td>000270</td>
<td>D21</td>
<td>PEREZ</td>
<td>27880</td>
<td>26168</td>
</tr>
</tbody>
</table>
Limiting Result Sets
LIMIT and OFFSET

- LIMIT and OFFSET support is popular, but not standard SQL

- This style of data access is most useful for those cases where you only need a subset (page) of rows

- The offset-clause is only allowed as part of the outer fullselect of a DECLARE CURSOR statement or a prepared select-statement
# LIMIT and OFFSET

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Action</th>
<th>Alternative Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIMIT x</td>
<td>Return the first x rows</td>
<td>FETCH FIRST x ROWS ONLY</td>
</tr>
<tr>
<td>LIMIT x OFFSET y</td>
<td>Skip the first y rows and return the next x rows</td>
<td>OFFSET y ROWS FETCH FIRST x ROWS ONLY</td>
</tr>
<tr>
<td>LIMIT y, x</td>
<td>Skip the first y rows and return the next x rows</td>
<td>OFFSET y ROWS FETCH FIRST x ROWS ONLY</td>
</tr>
</tbody>
</table>
LIMIT and OFFSET in action

Connect,  
SELECT…OFFSET 0 LIMIT 5  
Fetch 5 rows, Close,  
Disconnect

Connect,  
SELECT…OFFSET 5 LIMIT 5  
Fetch 5 rows, Close,  
Disconnect

Connect,  
SELECT…OFFSET 10 LIMIT 5  
Fetch 5 rows, Close,  
Disconnect

<table>
<thead>
<tr>
<th>Row Number</th>
<th>Ordering Data</th>
<th>Other Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Abcd</td>
<td>1234</td>
</tr>
<tr>
<td>2</td>
<td>Abdc</td>
<td>3214</td>
</tr>
<tr>
<td>3</td>
<td>Acbd</td>
<td>4131</td>
</tr>
<tr>
<td>4</td>
<td>Acdb</td>
<td>2143</td>
</tr>
<tr>
<td>5</td>
<td>Bacd</td>
<td>1243</td>
</tr>
<tr>
<td>6</td>
<td>Bacd</td>
<td>2341</td>
</tr>
<tr>
<td>7</td>
<td>Bcad</td>
<td>4213</td>
</tr>
<tr>
<td>8</td>
<td>Bcda</td>
<td>3142</td>
</tr>
<tr>
<td>9</td>
<td>Bdac</td>
<td>1423</td>
</tr>
<tr>
<td>10</td>
<td>Bdca</td>
<td>2431</td>
</tr>
<tr>
<td>11</td>
<td>Bdca</td>
<td>3412</td>
</tr>
<tr>
<td>12</td>
<td>Cadb</td>
<td>1324</td>
</tr>
<tr>
<td>13</td>
<td>Cbad</td>
<td>4321</td>
</tr>
</tbody>
</table>
Catalogs and Services
aka Don’t Reinvent the Wheel
SQL Catalogs and SQL Based Services

There are three general categories of catalogs/services:

- **DB2 for i Catalogs**
  - Contain metadata for DB2 for i objects
    - Includes objects created using both SQL and non-SQL interfaces

- **DB2 for i Services**
  - Includes services used to manage DB2 for i functionality

- **IBM i Services**
  - Provide SQL interfaces to perform functions that were traditionally done using CL commands and/or API interfaces

[http://ibm.biz/Db2foriServices](http://ibm.biz/Db2foriServices)
IBM® Db2® for i Services

Health Center Procedures
- QSYS2.HEALTH_ACTIVITY
- QSYS2.HEALTH_DATABASE_OVERVIEW
- QSYS2.HEALTH_DESIGN_LIMITS
- QSYS2.HEALTH_ENVIRONMENTAL_LIMITS
- QSYS2.HEALTH_SIZE_LIMITS
- QSYS2.RESET_ENVIRONMENTAL_LIMITS

Utility Procedures
- QSYS2.CANCEL_SQL
- QSYS2.DUMP_SQL_CURSORS
- QSYS2.EXTRACT_STATEMENTS
- QSYS2.FIND_AND_CANCEL_QSYSRVR_SQL
- QSYS2.FIND_QSYSRVR_JOBS
- QSYS2.GENERATE_SQL
- QSYS2.RESTART_IDENTITY
- SYSTOOLS.CHECK_SYS CST
- SYSTOOLS.CHECK_SYSROUTINE

Application Services
- QSYS2.DELIMIT_NAME – UDF
- QSYS2.OVERRIDE_QAQINI – PROCEDURE
- QSYS2.OVERRIDE_TABLE – PROCEDURE
- QSYS2.PARSE_STATEMENT – UDTF
- SYSPROC.WLM_SET_CLIENT_INFO – PROCEDURE

Performance Services
- QSYS2.CONDENSEINDEXADVICE – VIEW
- QSYS2.DATABASE_MONITOR_INFO – VIEW
- QSYS2.RESET_TABLE_INDEX_STATISTICS – PROCEDURE
- QSYS2.SYSIXADV – TABLE
- SYSTOOLS.ACT_ON_INDEX_ADVICE – PROCEDURE
- SYSTOOLS.HARVEST_INDEX_ADVICE – PROCEDURE
- SYSTOOLS.REMOVE_INDEXES – PROCEDURE

Plan Cache Procedures
- QSYS2.CHANGE_PLAN_CACHE_SIZE
- QSYS2.CLEAR_PLAN_CACHE
- QSYS2.DUMP_PLAN_CACHE
- QSYS2.DUMP_PLAN_CACHE_PROPERTIES
- QSYS2.DUMP_PLAN_CACHE_TOPN
- QSYS2.DUMP_SNAP_SHOT_PROPERTIES
- QSYS2.END_ALL_PLAN_CACHE_EVENT_MONITORS
- QSYS2.END_PLAN_CACHE_EVENT_MONITOR
- QSYS2.IMPORT_PC_EVENT_MONITOR
- QSYS2.IMPORT_PC_SNAPSHOT
- QSYS2.REMOVE_PC_EVENT_MONITOR
- QSYS2.REMOVE_PC_SNAPSHOT
- QSYS2.REMOVE_PERFORMANCE_MONITOR
- QSYS2.START_PLAN_CACHE_EVENT_MONITOR

IBM® i Services

Security Services
- QSYS2.AUTHORITY_COLLECTION - VIEW
- QSYS2.AUTHORIZATION_LIST_INFO - VIEW
- QSYS2.AUTORIZATION_LIST_USER_INFO - VIEW
- QSYS2.DRDA_AUTHENTICATION_ENTRY_INFO - VIEW
- QSYS2.FUNCTION_INFO - VIEW
- QSYS2.FUNCTION_USAGE - VIEW
- QSYS2.GROUP_PROFILE_ENTRIES - VIEW
- QSYS2.JOURNAL_INFO - VIEW
- QSYS2.SQL_CHECK_AUTHORITY - UDF
- QSYS2.USER_INFO - VIEW
- SYSPROC.SET_COLUMN_ATTRIBUTE - PROCEDURE

Communication Services
- QSYS2.NETSTAT_INFO - VIEW
- QSYS2.NETSTAT_INTERFACE_INFO - VIEW
- QSYS2.NETSTAT_JOB_INFO - VIEW
- QSYS2.NETSTAT_ROUTE_INFO - VIEW
- QSYS2.SERVER_SBS_ROUTING - VIEW
- QSYS2.SET_SERVER_SBS_ROUTING - PROCEDURE
- QSYS2.TCPP遂_IP_INFO - VIEW
- SYSSIBMADM.ENV_SYS_INFO - VIEW

Product Services
- QSYS2.LICENSE_INFO - VIEW
- SYSTOOLS.LICENSE_EXPIRATION_CHECK - PROCEDURE

Application Services
- QSYS2.ENVIRONMENT_VARIABLE_INFO - VIEW
- QSYS2.QCMDEXEC - PROCEDURE
- QSYS2.SERVICES_INFO - TABLE
- QSYS2.SET_PASE_SHELL_INFO - PROCEDURE
- QSYS2.STACK_INFO - UDTF

Storage Services
- QSYS2.ASP_INFO - VIEW
- QSYS2.ASP_VARY_INFO - VIEW
- QSYS2.MEDIA_LIBRARY_INFO - VIEW
- QSYS2.SYSDISKSTAT - VIEW
- QSYS2.SYSTEMSTG - VIEW
- QSYS2.USER_STORAGE - VIEW

Journal Services
- QSYS2.DISPLAY_JOURNAL - UDTF
- QSYS2.JOURNAL_INFO - VIEW
- QSYS2.USER_INFO - VIEW
- SYSPROC.SET_COLUMN_ATTRIBUTE - PROCEDURE

System Health Services
- QSYS2.SYLIMITS - VIEW
- QSYS2.SYSLIMTBL - TABLE

Message Handling Services
- QSYS2.HISTORY_LOG_INFO - UDTF
- QSYS2.JOBLOG_INFO - UDTF
- QSYS2.MESSAGE_QUEUE_INFO - VIEW
- QSYS2.REPLY_LIST_INFO - VIEW

PTF Services
- QSYS2.GROUP_PTF_INFO - VIEW
- QSYS2.PTF_INFO - VIEW
- SYSTOOLS.GROUP_PTF_CURRENCY - VIEW
- SYSTOOLS.GROUP_PTF_DETAIL - VIEW

Java Services
- QSYS2.JVM_INFO - VIEW
- QSYS2.SET_JVM - PROCEDURE

Spool Services
- QSYS2.OUTPUT_QUEUE_ENTRIES - VIEW
- QSYS2.OUTPUT_QUEUE_ENTRIES - UDTF
- QSYS2.OUTPUT_QUEUE_INFO - VIEW

Librarian Services
- QSYS2.LIBRARY_LIST_INFO - VIEW
- QSYS2.OBJECT_STATISTICS - UDTF

Work Management Services
- QSYS2.ACTIVE_JOB_INFO - UDTF
- QSYS2.GET_JOB_INFO - UDTF
- QSYS2.JOB_INFO - UDTF
- QSYS2.JOBQUEUE_INFO - VIEW
- QSYS2.MEMORY_POOL - UDTF
- QSYS2.QUEUE_INFO - VIEW
- QSYS2.OBJECT_LOCK_INFO - VIEW
- QSYS2.RECORD_LOCK_INFO - VIEW
- QSYS2.SCHEDULED_JOB_INFO - VIEW
- QSYS2.SYSTEM_STATUS - UDTF
- QSYS2.SYSTEM_STATUS_INFO - VIEW
- QSYS2.SYSTEM_VALUE_INFO - VIEW

http://ibm.com/Db2foriServices
Publications
and
Additional Help
• Get the SQL Reference!

• Perfect place for programmers to get ‘how to’ information

http://ibm.biz/DB2fori_SQLreference
• Need help using the newest DB2 for i technologies?
• Are you getting the most out DB2 for i?

IBM DB2 for i Consulting and Services

✓ Database & Query modernization
✓ DB2 Web Query
✓ Database design, features and functions
✓ DB2 SQL performance analysis and tuning
✓ Data warehousing and Business Intelligence
✓ DB2 for i education and training

Contact:   Doug Mack       mackd@us.ibm.com
IBM Systems and Technology Group
Rochester, MN USA
IBM i on developerWorks

IBM i technology information wiki

- Wiki URL: www.ibm.com/developerworks/ibmi/techupdates
- DB2 URL: www.ibm.com/developerworks/ibmi/techupdates/db2

- Centralized location for communicating DB2 for i enhancements made in between major IBM i releases
- Organized by subject matter and category of enhancement
- Notify me of Page Changes option
- comment or recommend specific pages

- Info Center remains the site for technical documentation, however, sometimes the information will appear on dW first
Thank you!
Special notices

This document was developed for IBM offerings in the United States as of the date of publication. IBM may not make these offerings available in other countries, and the information is subject to change without notice. Consult your local IBM business contact for information on the IBM offerings available in your area.

Information in this document concerning non-IBM products was obtained from the suppliers of these products or other public sources. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. Send license inquiries, in writing, to IBM Director of Licensing, IBM Corporation, New Castle Drive, Armonk, NY 10504-1785 USA.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

The information contained in this document has not been submitted to any formal IBM test and is provided "AS IS" with no warranties or guarantees either expressed or implied.

All examples cited or described in this document are presented as illustrations of the manner in which some IBM products can be used and the results that may be achieved. Actual environmental costs and performance characteristics will vary depending on individual client configurations and conditions.

IBM Global Financing offerings are provided through IBM Credit Corporation in the United States and other IBM subsidiaries and divisions worldwide to qualified commercial and government clients. Rates are based on a client's credit rating, financing terms, offering type, equipment type and options, and may vary by country. Other restrictions may apply. Rates and offerings are subject to change, extension or withdrawal without notice.

IBM is not responsible for printing errors in this document that result in pricing or information inaccuracies.

All prices shown are IBM's United States suggested list prices and are subject to change without notice; reseller prices may vary.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Some measurements quoted in this document may have been estimated through extrapolation. Users of this document should verify the applicable data for their specific environment.

Revised September 26, 2006