Disclaimer:

This presentation may contain examples of code and names of companies or persons. The code is given for presentation purposes and has not been tested by IBM and/or Skip Marchesani. Therefore IBM and/or Skip Marchesani does not guarantee the reliability, serviceability, or function of the code and the code is provided "AS IS". IBM AND/OR SKIP MARCHESANI EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, INCLUDING BUT NOT LIMITED TO, THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND MERCHANTABILITY. Any names appearing in this presentation are designed to be fictitious and IBM and Skip Marchesani makes no representations as to the accuracy of the names or data presented in accordance therewith.

Reproduction:

This presentation is the property of Skip Marchesani and System i Developer, LLC. Permission is granted to make a limited number of copies of this material for non-commercial purposes, providing this page is included with all copies. Express written permission is required for making copies for other purposes.

System i, iSeries, AS/400, i5/OS, OS/400, DB2/400, DB2 UDB are registered trademarks of the IBM Corporation
Overview

An Important Goal for this Presentation

- An important goal for this presentation is to show you that SQL is just another programming language. The more you work and play with it, the more you realize the power of SQL and what it can do for you as an application development or database manipulation tool. With a little thought and creativity you will find you can use SQL for things that at first glance you did not think possible.

- This presentation is based on the SQL function available in V5R4 of OS/400, assumes you have a basic understanding of relational database concepts and SQL, and that you are familiar with using the SELECT, INSERT, UPDATE, and DELETE statements.

Sample Tables Used in Examples

- Employee Table - EMP

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>CLS</th>
<th>SEX</th>
<th>DPT</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>5</td>
<td>M</td>
<td>911</td>
<td>7,000</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>2</td>
<td>M</td>
<td>901</td>
<td>6,000</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>5</td>
<td>M</td>
<td>977</td>
<td>3,200</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>4</td>
<td>M</td>
<td>977</td>
<td>6,500</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>3</td>
<td>F</td>
<td>911</td>
<td>7,500</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>2</td>
<td>M</td>
<td>990</td>
<td>6,500</td>
</tr>
</tbody>
</table>

- Department Table - DEP

<table>
<thead>
<tr>
<th>DPT</th>
<th>DNM</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>Accounts</td>
</tr>
<tr>
<td>977</td>
<td>Manufact</td>
</tr>
<tr>
<td>911</td>
<td>Sales</td>
</tr>
<tr>
<td>990</td>
<td>Spares</td>
</tr>
</tbody>
</table>
Overview

- **Part 1**
  - SQL Functions for Data manipulation and Analysis
  - Summarizing Data with the SELECT Statement
  - Other Interesting Stuff
- **Part 2**
  - Working with Edit Characters
  - Subselect - Basic to Advanced
  - Identifying Potential Duplicate Rows
  - Searching by Sound
- **Part 3**
  - Embedding SQL in RPG (and Cobol) Programs
- **Part 4**
  - Stored Procedures
  - SQL Procedure Language
- **Part 5**
  - SQL Triggers and Other Trigger Enhancements in V5
Overview Part 1

- COUNT, SUM, and AVG for data analysis
- DEC to format numeric columns
- AS to name a derived column
- SUBSTR and CONCAT for character columns
- CAST, CHAR, and DIGITS to change data type
- Summarizing Data with SELECT
- Other Interesting Stuff
- Summary
- V5R3 and V5R4 SQL Information Sources

COUNT, SUM, and AVG for Data Analysis
Counting with SQL

Types of Counting with SQL

- **Total number of rows in a table**
  - How many rows are in the table

- **Subset of rows in a table**
  - Based on selection criterion in WHERE clause

- **Distinct or unique occurrences of rows**
  - How many different (distinct or unique) values exist

- **Groups of rows**
  - Summarization of data using GROUP BY and HAVING clauses

COUNT

Two Types of COUNT

- **Column Function**

- **COUNT**
  - Result set is a large integer with precision (length) of 10
  - Max result set size is 10 digits with a limit of 2,147,483,647

- **COUNT BIG**
  - Max result set size is 32 digits
  - 9,999,999,999,999,999,999,999,999,999,999,999
  - aka COUNT REALLY REALLY BIG
COUNT...

COUNT Syntax

- **COUNT(\(*\)**
  - Includes rows with null values in the count

- **COUNT(column_name) or COUNT(ALL column_name)**
  - Omits rows with null values from the count

- **COUNT(DISTINCT column_name)**
  - Omits rows with duplicate or null values from the count

Note: If database tables are not defined with null capable columns, COUNT(*) and COUNT(column_name) or COUNT(ALL column_name) will return the same result set.

COUNT Examples

```
SELECT COUNT( *) FROM emp
```

```
COUNT(*)
6
```

```
SELECT COUNT( *) FROM emp
WHERE dpt = 977
```

```
COUNT(*)
2
```
COUNT...

COUNT Examples...

```
SELECT COUNT(DISTINCT class) FROM emp

COUNT(*)
4
```

```
SELECT dpt, COUNT(*) FROM emp
GROUP BY dpt
ORDER BY dpt

<table>
<thead>
<tr>
<th>DPT</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>1</td>
</tr>
<tr>
<td>911</td>
<td>2</td>
</tr>
<tr>
<td>977</td>
<td>2</td>
</tr>
<tr>
<td>990</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Summing or Adding with SQL

**SUM Column Function**

- Result set is the sum of a set of numbers with 1 to many numbers in the set
  - Precision (length) is 31 with a scale (number of decimal positions) the same as the column being summed

- Data type of column must be numeric
  - Decimal - packed
  - Numeric - zoned
  - Smallint
  - Integer
  - Bigint
  - Float
  - Real
  - Double
SUM

SUM Syntax

- **SUM(column_name)** or **SUM(ALL column_name)**
  - Sum the value found in the column for all rows selected
- **SUM(DISTINCT column_name)**
  - Duplicate values are not included in the sum

SUM Examples

```
SELECT SUM(sal) FROM emp
SUM(sal)
36,700

SELECT SUM(sal) FROM emp
  WHERE dpt = 911
SUM(sal)
14,500
```
Averaging Data with SQL

AVG Column Function

- **Result set is the average of a set of numbers with 1 to many numbers in the set**
  - Precision (length) is 31 with a scale (number of decimal positions) equal to 31 minus defined column length
    - Translation: Lots of decimal positions

- **Data type of column must be numeric**
  - Decimal - packed
  - Numeric - zoned
  - Smallint
  - Integer
  - Bigint
  - Float
  - Real
  - Double

AVG

AVG Syntax

- **AVG(column_name) or AVG(ALL column_name)**
  - Average the value found in the column for all rows selected

- **AVG(DISTINCT column_name)**
  - Duplicate values are not included in the average
AVG...

AVG Examples

SELECT AVG(sal) FROM emp

| AVG(sal) | 6,116.66666666666666666666666666 |

SELECT AVG(sal) FROM emp WHERE dpt = 911

| AVG(sal) | 7,250.00000000000000000000000000 |

• Ugly to look at!

DEC to Format Numeric Results

Copyright 2008 System i Developer, LLC
DEC to Format Numeric Results

DEC or DECIMAL Scalar Column Function

- Result set is the representation of a number in decimal format (1234.56)
  - Precision (length) and scale (number of decimal positions) are defined as part of the DEC function

- Data type of column must be numeric
  - Decimal - packed
  - Numeric - zoned
  - Smallint
  - Integer
  - Bigint
  - Float
  - Real
  - Double

NEWLY DISCOVERED V5R3 SECRET!

- If data type not numeric - implicit cast of datatype to numeric
- The DECIMAL function returns a decimal representation of a
  - Number
  - Character or graphic string representation of a
    - Decimal number
    - Integer
    - Floating-point number

- Many V5R3 scalar functions will implicitly cast to correct data type when possible
DEC

DEC or DECIMAL Syntax

- DEC(expression, precision, scale, ['decimal-character'])

AVG Examples

SELECT AVG(sal) FROM emp

| AVG(sal) | 6,116.6666666666666666666666 |

SELECT AVG(sal) FROM emp
WHERE dpt = 911

| AVG(sal) | 7,250.0000000000000000000000 |

- Ugly to look at!
DEC...

DEC or DECIMAL Examples

```
SELECT DEC(AVG(sal),7,2) FROM EMP
```

6,116.66

```
SELECT DECIMAL(AVG(sal),7,2) FROM emp
WHERE dpt = 911
```

7,250.00

To Truncate or Round?

What is the Correct Answer?

```
SELECT DEC(AVG(sal),7,2) FROM emp
```

6,116.66

6,116.66 or 6,116.67
Rounding a Result

ROUND Scalar Column Function

- Returns a numeric value rounded to some number of places to the right or left of the decimal point

- ROUND(expression, decimal-position)
  - Positive number - round to right of decimal point
  - Negative number - round to left of decimal point

SELECT DEC(ROUND(AVG(sal),2),7,2)
FROM emp

DEC
6,116.67
What's the Difference?

Column Headings Reflect the Function

```
SELECT  DEC(AVG(sal),7,2)  FROM  EMP

| DEC | 6,116.66 |
```

```
SELECT  DECIMAL(AVG(sal),7,2)  FROM  emp
WHERE  dpt = 911

| DEC | 7,250.00 |
```

- Easy to fix

---

AS to Name a Derived Column
AS to Name a Derived Column

AS Operator

- **Used to**
  - Rename an existing column
  - Name a derived column

- **Considerations**
  - Name cannot be qualified
  - Name does not have to be unique

AS Syntax

- **AS column_name**
DEC or DECIMAL Examples

SELECT DEC(AVG(sal),7,2) FROM EMP

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC</td>
<td></td>
</tr>
<tr>
<td>6,116.67</td>
<td></td>
</tr>
</tbody>
</table>

SELECT DECIMAL(AVG(sal),7,2) FROM emp
WHERE dpt = 911

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC</td>
<td></td>
</tr>
<tr>
<td>7,250.00</td>
<td></td>
</tr>
</tbody>
</table>

AS Examples

SELECT DEC(AVG(sal),7,2) AS avgsal FROM EMP

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AVGSAL</td>
<td></td>
</tr>
<tr>
<td>6,116.67</td>
<td></td>
</tr>
</tbody>
</table>

SELECT DECIMAL(AVG(sal),7,2) AS avg911sal FROM emp
WHERE dpt = 911

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG911SAL</td>
<td></td>
</tr>
<tr>
<td>7,250.00</td>
<td></td>
</tr>
</tbody>
</table>
AS...

AS Can Be Implied

SELECT DEC(AVG(sal),7,2) avgsal
FROM EMP

| AVG(SAL) | 6,116.67 |

SELECT DECIMAL(AVG(sal),7,2) avg911sal
FROM emp
WHERE dpt = 911

| AVG911SAL | 7,250.00 |

SUBSTR and CONCAT for Character Columns
SUBSTR for a Character Column

SUBSTR or SUBSTRING Scalar Function

- Result is a subset or substring or a character column
  - Start position and length of resulting string is defined in the function

- Data type must be character
  - Fixed or variable length - including large object (BLOB)
  - Single or double byte

- V5R3 implicit cast

SUBSTR...

SUBSTR or SUBSTRING Syntax

- SUBSTR(string_expression, start_position, length)
- SUBSTRING(string_expression FROM start_position FOR length)
**SUBSTR...**

**SUBSTR or SUBSTRING Examples**

```
SELECT dpt, SUBSTR(dnm,1,4) AS sname
FROM dep
```

<table>
<thead>
<tr>
<th>DPT</th>
<th>SNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>Acco</td>
</tr>
<tr>
<td>977</td>
<td>Manu</td>
</tr>
<tr>
<td>911</td>
<td>Sale</td>
</tr>
<tr>
<td>990</td>
<td>Spar</td>
</tr>
</tbody>
</table>

**SUBSTR and V5R3 Implicit Cast**

```
SELECT dpt, SUBSTR(dpt,2,2) AS sdpt
FROM dep
```

<table>
<thead>
<tr>
<th>DPT</th>
<th>SDPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>01</td>
</tr>
<tr>
<td>977</td>
<td>77</td>
</tr>
<tr>
<td>911</td>
<td>11</td>
</tr>
<tr>
<td>990</td>
<td>90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DPT</th>
<th>DNM</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>Accounts</td>
</tr>
<tr>
<td>977</td>
<td>Manufact</td>
</tr>
<tr>
<td>911</td>
<td>Sales</td>
</tr>
<tr>
<td>990</td>
<td>Spares</td>
</tr>
</tbody>
</table>
CONCAT for a Character Column

CONCAT Scalar Function

- Combines or concatenates two string expressions together
- Result is a single string consisting of the first string expression, followed by the second string expression
- Data type must be character
  - Fixed or variable length - including large object (BLOB)
  - Single or double byte
- V5R3 Implicit Cast

CONCAT Syntax - Industry Standard

- CONCAT(string_expression_01, string_expression_02)

CONCAT Syntax - Non Standard

- string_01 || string_02 || string_03
  - || = double pipe (upper case backward slash)
- string_01 CONCAT string_02 CONCAT string_03
CONCAT...

CONCAT Examples - Simple CONCAT

```
SELECT  nbr,  nam,
        CONCAT(sex,  nam)  AS  sexnam
FROM  emp
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>SEXNAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>MEd</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>MHeikki</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>MJohn</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>MMike</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>FMarcela</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>MFrank</td>
</tr>
</tbody>
</table>

CONCAT...

CONCAT Examples - CONCAT with SUBSTR

```
SELECT  dpt,
        CONCAT(SUBSTR(dnm,1,4),  SUBSTR(dnm,1,4))  AS  double
FROM  dep
```

<table>
<thead>
<tr>
<th>DPT</th>
<th>DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>AccoAcco</td>
</tr>
<tr>
<td>977</td>
<td>ManuManu</td>
</tr>
<tr>
<td>911</td>
<td>SaleSale</td>
</tr>
<tr>
<td>990</td>
<td>SparSpar</td>
</tr>
</tbody>
</table>
CONCAT Examples - CONCAT with a Literal

```sql
SELECT dpt,
    CONCAT(SUBSTR(dnm,1,4), 'Stuff')
AS stuff
FROM dep
```

<table>
<thead>
<tr>
<th>DPT</th>
<th>STUFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>AccoStuff</td>
</tr>
<tr>
<td>977</td>
<td>ManuStuff</td>
</tr>
<tr>
<td>911</td>
<td>SaleStuff</td>
</tr>
<tr>
<td>990</td>
<td>SparStuff</td>
</tr>
</tbody>
</table>

CONCAT Examples - Concat with 3 Strings

```sql
SELECT dpt,
    CONCAT(CONCAT(SUBSTR(dnm,1,4), '-'), 'Stuff')
AS hyphen
FROM dep
```

<table>
<thead>
<tr>
<th>DPT</th>
<th>HYPHEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>Acco-Stuff</td>
</tr>
<tr>
<td>977</td>
<td>Manu-Stuff</td>
</tr>
<tr>
<td>911</td>
<td>Sale-Stuff</td>
</tr>
<tr>
<td>990</td>
<td>Spar-Stuff</td>
</tr>
</tbody>
</table>
## CONCAT Examples - Concat with 3 Strings

```sql
SELECT dpt,
       SUBSTR(dnm,1,4) || '-' || 'Stuff'
       AS hyphen
FROM dep
```

<table>
<thead>
<tr>
<th>DPT</th>
<th>HYPHEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>Acco-Stuff</td>
</tr>
<tr>
<td>977</td>
<td>Manu-Stuff</td>
</tr>
<tr>
<td>911</td>
<td>Sale-Stuff</td>
</tr>
<tr>
<td>990</td>
<td>Spar-Stuff</td>
</tr>
</tbody>
</table>
CONCAT...

CONCAT and V5R3 Implicit Cast

```
SELECT dpt,
    SUBSTR(dnm,1,4) CONCAT '-' CONCAT dpt
AS hyphen
FROM dep
```

<table>
<thead>
<tr>
<th>DPT</th>
<th>HYPHEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>Acco-901</td>
</tr>
<tr>
<td>977</td>
<td>Manu-977</td>
</tr>
<tr>
<td>911</td>
<td>Sale-911</td>
</tr>
<tr>
<td>990</td>
<td>Spar-990</td>
</tr>
</tbody>
</table>

SUBSTR & CONCAT

SUBSTR & CONCAT with Numeric Columns

- Can be used with numeric columns but data type must be changed from numeric to character
- V5R3 Implicit Cast
CAST, CHAR, and DIGITS

to Change Data Type

CAST to Change Data Type

CAST Scaler Function

- Converts the existing data type of the CAST expression to the specified data type
  - Numeric to character
  - Character to numeric
  - Numeric to numeric
  - Character to character
  - See SQL Reference for supported CAST from/to data types

- When converting numeric to character
  - Leading zeros are truncated
  - Decimal point included and digits to the right are part of result string
  - Digits are left justified
CHAR to Change Data Type

**CHAR Scaler Function**

- Converts the existing data type of the CHAR expression to character data type
  - Numeric to character
  - Character to character
  - See SQL Reference for supported CHAR data types

- **When converting numeric to character**
  - Leading zeros are truncated
  - Decimal point included and digits to the right are part of result string
  - Digits are left justified

DIGITS to Change Data Type

**DIGITS Scaler Function**

- Only converts numeric to character data type

- **When converting numeric to character**
  - Leading zeros are NOT truncated
  - Decimal point ignored and digits to the right are part of result string
  - Digits are left justified
CAST, CHAR, and DIGITS

CAST Syntax

- \texttt{CAST(expression AS data\_type)}

CHAR Syntax

- \texttt{CHAR(expression [,attribute])}

DIGITS Syntax

- \texttt{DIGITS(numeric\_expression)}

CAST, CHAR, and DIGITS Example

\begin{verbatim}
SELECT nbr, nam,
       SUBSTR(CAST(dpt AS CHAR(3)),2,2) AS chardpt
FROM emp

SELECT nbr, nam,
       SUBSTR(CHAR(dpt)),2,2) AS chardpt
FROM emp

SELECT nbr, nam,
       SUBSTR(DIGITS(dpt),2,2) AS chardpt
FROM emp
\end{verbatim}
## CAST, CHAR, and DIGITS...

### CAST, CHAR, and DIGITS Example

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>CHARDPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>11</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>01</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>77</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>77</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>11</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>90</td>
</tr>
</tbody>
</table>

## CAST and DIGITS...

### CAST and DIGITS Example

```sql
SELECT * FROM empd
ORDER BY nbr
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>CLS</th>
<th>SEX</th>
<th>DPT</th>
<th>SAL</th>
<th>EDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>5</td>
<td>M</td>
<td>911</td>
<td>7,000</td>
<td>11,152,000</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>2</td>
<td>M</td>
<td>901</td>
<td>6,000</td>
<td>03,122,002</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>5</td>
<td>M</td>
<td>977</td>
<td>3,200</td>
<td>05,152,001</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>4</td>
<td>M</td>
<td>977</td>
<td>6,500</td>
<td>10,152,000</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>3</td>
<td>F</td>
<td>911</td>
<td>7,500</td>
<td>12,012,001</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>2</td>
<td>M</td>
<td>990</td>
<td>6,500</td>
<td>09,152,003</td>
</tr>
</tbody>
</table>

**Note:** Neither Interactive SQL nor RUN SQL Scripts will show leading zeros in EDATE

Copyright 2008 System i Developer, LLC
CAST and DIGITS...

CAST and DIGITS Example

```
SELECT * FROM empd
ORDER BY edate
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>CLS</th>
<th>SEX</th>
<th>DPT</th>
<th>SAL</th>
<th>EDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Heikki</td>
<td>2</td>
<td>M</td>
<td>901</td>
<td>6,000</td>
<td>03,012,002</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>5</td>
<td>M</td>
<td>977</td>
<td>3,200</td>
<td>05,152,001</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>2</td>
<td>M</td>
<td>990</td>
<td>6,500</td>
<td>09,152,003</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>4</td>
<td>M</td>
<td>977</td>
<td>6,500</td>
<td>10,152,000</td>
</tr>
<tr>
<td>10</td>
<td>Ed</td>
<td>5</td>
<td>M</td>
<td>911</td>
<td>7,000</td>
<td>11,152,000</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>3</td>
<td>F</td>
<td>911</td>
<td>7,500</td>
<td>12,012,001</td>
</tr>
</tbody>
</table>

Note: Neither Interactive SQL nor RUN SQL Scripts will show leading zeros in EDATE

CAST Truncates Leading Zeros

```
SELECT nbr, nam, edate,
    CONCAT(SUBSTR(CAST(edate AS CHAR(8)),5,4),
    SUBSTR(CAST(edate AS CHAR(8)),1,4))
    AS hire_date
FROM empd  ORDER BY hire_date
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>EDATE</th>
<th>HIRE_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>John</td>
<td>5152001</td>
<td>001 5152</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>3012002</td>
<td>002 3012</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>9152003</td>
<td>003 9152</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>10152000</td>
<td>20001015</td>
</tr>
<tr>
<td>10</td>
<td>Ed</td>
<td>11152000</td>
<td>20001115</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>12012001</td>
<td>20011201</td>
</tr>
</tbody>
</table>

CAST turned 05152001 into 5152001_
CAST and DIGITS...

DIGITS Preserves Leading Zeros

```
SELECT nbr, nam, edate,
    CONCAT(SUBSTR(DIGITS(edate),5,4),
            SUBSTR(DIGITS(edate),1,4)) AS hire_date
FROM empd
ORDER BY hire_date
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>EDATE</th>
<th>HIRE_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>Mike</td>
<td>10152000</td>
<td>20001015</td>
</tr>
<tr>
<td>10</td>
<td>Ed</td>
<td>11152000</td>
<td>20001115</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>5152001</td>
<td>20010515</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>12012001</td>
<td>20011201</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>3012002</td>
<td>20020301</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>9152003</td>
<td>20030915</td>
</tr>
</tbody>
</table>
Summarizing Data with SELECT

The SELECT statement is an *Either Or* Proposition

- **Either** return detail rows in the result set
- **Or** return summarized data in the result set
- A single SQL statement has no capability to do
  - Detail
  - Detail
  - Level break
  - Detail
  - Level break

- **Query Manager CAN do above**
  - SQL based query product

**SELECT Statement Clauses**

- SELECT . . . . . (columns, *, or expressions)
- FROM . . . . . (tables or views)
- WHERE . . . . . (row selection criteria)
- GROUP BY . . . (row summarization criteria)
- HAVING . . . . (GROUP BY selection criteria)
- ORDER BY . . . (column ordering criteria)
Summarizing Data with SELECT...

GROUP BY Clause

- Defines grouping or summary criteria for SELECT
- Format

```
SELECT  fldA,  fldE,  FUNCTION(fldG),  FUNCTION(fldH)
FROM    table-name
WHERE   selection-criteria
GROUP BY  fldE,  fldA
ORDER BY  fldA,  fldE
```

- If a column referenced in the column list of the SELECT statement is not operated on by a function, that column must be referenced as part of the grouping criteria in the GROUP BY clause

Summarizing Data with SELECT...

GROUP BY Clause...

- For each department list the total salary and total number of employees

```
SELECT  dpt,  SUM(sal),  COUNT(*)
FROM    emp
GROUP BY  dpt
ORDER BY  dpt
```

<table>
<thead>
<tr>
<th>DPT</th>
<th>SUM(SAL)</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>6,000</td>
<td>1</td>
</tr>
<tr>
<td>911</td>
<td>14,500</td>
<td>2</td>
</tr>
<tr>
<td>977</td>
<td>9,700</td>
<td>2</td>
</tr>
<tr>
<td>990</td>
<td>6,500</td>
<td>1</td>
</tr>
</tbody>
</table>
Summarizing Data with SELECT...

HAVING Clause...

- Defines GROUP BY selection criteria
- Format

```
SELECT fldA, fldE, FUNCTION(fldG), FUNCTION(fldH)
FROM   table-name
WHERE selection-criteria
GROUP BY fldE, fldA
HAVING group-by-selection-criteria
ORDER BY fldA, fldE
```

For those departments that have more than one employee, list the total salary and total number of employees

```
SELECT dpt, SUM(sal) AS saltot,
     COUNT( * ) AS emptot
FROM   emp
GROUP BY dpt
HAVING COUNT( * ) > 1
ORDER BY dpt
```

<table>
<thead>
<tr>
<th>DPT</th>
<th>SALTOT</th>
<th>EMPTOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>911</td>
<td>14,500</td>
<td>2</td>
</tr>
<tr>
<td>977</td>
<td>9,700</td>
<td>2</td>
</tr>
</tbody>
</table>
Some Other Interesting SQL Scalar Functions

- **CASE**
  - Allows selection and substitution of a value based on a condition

- **RRN**
  - Returns the relative record number of a row in a table

- **HEX**
  - Returns the hexadecimal representation of a value

- **RTRIM** (Right Trim)
  - Removes blanks or hexadecimal zeros from the end, or the trailing or right side, of a string expression

- **LENGTH**
  - Returns the length of a number or character value
Other Interesting Stuff...

New V5R3 Scalar Functions

- **INSERT**
  - Inserts a number or character string into an existing string and optionally overlays specific positions in the existing string with the insert string

- **REPLACE**
  - Replaces all occurrences of a search string in a number or character string with a replacement string

CASE for Substitution of a Value

Two forms of CASE Syntax

- **Simple WHEN clause**
  ```sql
  CASE  expression
    WHEN  expression  THEN  result-expression
    WHEN  expression  THEN  result-expression
    ...    
    ELSE  result-expression
  END CASE
  ```

- **Searched WHEN clause**
  ```sql
  CASE
    WHEN  search-condition  THEN  result-expression
    WHEN  search-condition  THEN  result-expression
    ...    
    ELSE  result-expression
  END CASE
  ```
CASE...

Two forms of Case...

- **Simple WHEN clause**

  ```sql
  SELECT nbr, nam,
  CASE dpt
      WHEN 901 THEN 'Accounts'
      WHEN 911 THEN 'Sales'
      WHEN 977 THEN 'Manufact'
      WHEN 990 THEN 'Spares'
      ELSE 'No Name'
  END CASE
  FROM emp ORDER BY nbr
  ```

- **Searched WHEN clause**

  ```sql
  SELECT nbr, nam,
  CASE
      WHEN dpt = 901 THEN 'Accounts'
      WHEN dpt = 911 THEN 'Sales'
      WHEN dpt = 977 THEN 'Manufact'
      WHEN dpt = 990 THEN 'Spares'
      ELSE 'No Name'
  END CASE
  FROM emp ORDER BY nbr
  ```
CASE...

Two forms of Case...

- Results Set for both CASE Examples

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>CASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>Sales</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>Accounts</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>Manufact</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>Manufact</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>Sales</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>Spares</td>
</tr>
</tbody>
</table>

RRN to Find Relative Record Number

RRN Scalar Function

- Result set is the relative record number of a row
  - Length is 15 with zero decimal positions

RRN Syntax

- RRN(table_name)
**RRN**

**RRN Example**

```sql
SELECT nbr, nam, RRN(ax) AS rec
FROM emp ax ORDER BY nbr
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>REC</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>5</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>3</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>6</td>
</tr>
</tbody>
</table>

**HEX to Find Hexadecimal Value**

**HEX Scalar Function**

- Result set is the hexadecimal representation of a value
  - Length is twice the defined length with a max length of approx 32K

**HEX Syntax**

- `HEX(expression)`
HEX Example

```
SELECT nbr, nam, HEX(nam) AS hex_nam
FROM emp ORDER BY nbr
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>HEX_NAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>C5844040404040404040</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>C8858992928940404040</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>D1968895404040404040</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>D4899285404040404040</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>D4819983859381404040</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>C6998195924040404040</td>
</tr>
</tbody>
</table>

RTRIM to Remove Trailing Blanks

RTRIM Scalar Function

- Result set is the RTRIM expression with trailing blanks or hexadecimal zeros removed
  - Length of the result set is the length of the expression minus the number of blanks or hex zeros removed

RTRIM Syntax

- RTRIM(expression)
RTRIM

RTRIM Example

```
SELECT nbr, nam, RTRIM(nam) AS tnam
FROM emp ORDER BY nbr
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>TNAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>Ed</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>Heikki</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>John</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>Mike</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>Marcela</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>Frank</td>
</tr>
</tbody>
</table>

• What's the difference between NAM and TNAM?

LENGTH to Determine the Length of a Value

LENGTH Scalar Function

• Result set is the length of the string in the expression including any blanks or zeros
  ▶ Result set length is 10 digits with zero decimals

LENGTH Syntax

• LENGTH(expression)
LENGTH

LENGTH Example

```
SELECT nbr, nam, LENGTH(nam) AS len1
FROM emp ORDER BY nbr
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>LEN1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>10</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>10</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>10</td>
</tr>
</tbody>
</table>

• Why is the length for each NAM string = 10?

Combining RTRIM and LENGTH

How Many Non Blank Characters are in Each Name?

```
SELECT nbr, nam, LENGTH(RTRIM(nam)) AS len2
FROM emp ORDER BY nbr
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>LEN2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>7</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>5</td>
</tr>
</tbody>
</table>
INSERT to Overlay Positions In a String

**INSERT Scalar Function - New for V5R3**

- Result set is the source string with the insert string placed into the source string at the specified start position, and optionally specified positions in the source string overlaid with the insert string
  - Result set length is length of the source string plus length of the insert string minus the number of positions overlaid (if any)
  - Result set length cannot exceed the maximum length of the data type
  - Data type of the source string and insert string must be compatible

**INSERT Syntax**

- `INSERT(source_string, start, length, insert_string)`

---

**INSERT Example with No Overlay**

```sql
SELECT nbr, nam, INSERT(nam, 3, 0, 'xy') AS insrt1
FROM emp
ORDER BY nbr
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>INSRT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>Edxy</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>Hexyikki</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>Joxyhn</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>Mixyke</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>Maxyrcela</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>Frxyank</td>
</tr>
</tbody>
</table>
**INSERT...**

**INSERT Example Overlaying Positions 3 and 4**

```sql
SELECT nbr, nam, INSERT(nam, 3, 2, 'xy') AS insrt2
FROM emp ORDER BY nbr
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>INSRT2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>Edxy</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>Hexyki</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>Joxy</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>Mixy</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>Maxyela</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>Frxyk</td>
</tr>
</tbody>
</table>

- Easy way to update part of a character string!

---

**REPLACE - Based on a Search String**

**REPLACE Scalar Function - New for V5R3**

- Result set is the source string with all occurrences of the search string replaced with the replace string
  - Result set length is length of the source string plus length of the replace string minus the length of the search string
  - Result set length cannot exceed the maximum length of the data type
  - Data type of the source string and replace string must be compatible
  - If no match with the search string, the source string is returned unchanged as the result set

**REPLACE Syntax**

- REPLACE(source_string, search_string, replace_string)
REPLACE

REPLACE Example

```sql
SELECT nbr, nam, REPLACE(nam, 'ik', 'qqq') AS replc1
FROM emp ORDER BY nbr
```

<table>
<thead>
<tr>
<th>NBR</th>
<th>NAM</th>
<th>REPLC1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Ed</td>
<td>Ed</td>
</tr>
<tr>
<td>20</td>
<td>Heikki</td>
<td>Heqqki</td>
</tr>
<tr>
<td>30</td>
<td>John</td>
<td>John</td>
</tr>
<tr>
<td>40</td>
<td>Mike</td>
<td>Mqqqe</td>
</tr>
<tr>
<td>50</td>
<td>Marcela</td>
<td>Marcela</td>
</tr>
<tr>
<td>60</td>
<td>Frank</td>
<td>Frank</td>
</tr>
</tbody>
</table>

- Easy way to update part of a character string!

Summary

I Didn't Know You Could Do that with SQL!

- COUNT, SUM, and AVG for data analysis
- DEC to format numeric columns
- AS to name a derived column
- SUBSTR and CONCAT for character columns
- CAST, CHAR, and DIGITS to change data type
- Summarizing Data with SELECT
- CASE
- HEX and RRN
- RTRIM and LENGTH
- INSERT and REPLACE - New for V5R3
Summary...

I Didn't Know You Could Do that with SQL...

- From this presentation you should have a better understanding of SQL as a programming language.

- The more you work and play with it, the more you realize the power of SQL and what it can do for you as an application development or database manipulation tool.

- With a little thought and creativity you will find you can use SQL for things that at first glance you did not think possible.

V5R3 and V5R4 SQL Information Sources

- iSeries Information Center Publications - Web or CD
  - SQL Reference
  - SQL Programming
  - Embedded SQL Programming
  - Query Manager Use
  - SQL Messages and Codes

- To access Info Center on the Web
    - In left scroll bar
      - Click on iSeries Information Center ...
      - Click on Database
      - Click on Printable PDFs
      - Use right scroll bar to scroll down to above SQL publication

- DB2 UDB for iSeries on the Web