Expressions in Query: An In-Depth Exploration Into Function Statements

Session #23398 (S1)  
March 11, 2007  
(8:30 AM - 11:30 AM)  

Alliance 2007 Conference  
Orlando, Florida
Presenters

Uriel Hernandez
Information Technology Applications Specialist
Central Washington University

Tim McGuire
Information Technology Applications Specialist
Central Washington University
Overview

• Review in detail both common function statements and complex expressions.

• Explore the many possibilities of using function statements to provide greater flexibility, functionality and power to your queries.

• Discover creative ways to overcome many of the limitations of the PS Query Tool for improved reporting use.
Agenda Topics

• Exploring Functions
• Methodology for Finding Solutions
• Break
• Exploring Further
• Hands-On Problem Solving
• Resources
• Q & A
CWU Trivia

Main Campus

• Located in Ellensburg, WA
• 7,754 full-time students
• 8,225 in attendance

Off-site Centers

• Six satellite centers
  • 3 each in Eastern and Western, WA
• 1,246 full-time students
• 1,734 in attendance
PeopleSoft HRSA at CWU

• Version: PeopleSoft 8.0 SP1
• PeopleTools: 8.22.12
• Database: Oracle 10g
• Live Date: September 2004
• Self-Service Name: Safari
• Currently upgrading to HCMCS 8.9
• Projected Go-Live: October 2007
Ground Rules

• Interactive - Participation Required
• Understanding of Query Tool
• Correct Joins and Criteria Needed
• Not Table Specific
• System Independent
• Just Options
• Ideas Welcome
SQL FUNCTIONS*

*Command Set for Oracle 9i/10g Databases
What are functions?

Functions are special types of command words in the SQL command set, usually one-word commands which manipulate data items and return a single value which can be determined by various input parameters.
Function Groups

There are two groups of functions:

- *Deterministic*
- *Nondeterministic*

So, what does that really mean!?!?
Functions Groups (continued)

Deterministic functions always return the same result

• *When having specific set of input values with the same state of database*

Nondeterministic functions may return different results

• *Even with specific set of input values and same state of database*
Function Statements - Review

Function Statements are predefined system commands expressed with their operational parameters.
Function Statements - Example

JUMP

How High? = 24 inches
How Long? = 120 seconds
Who? = Mickey

JUMP(‘Mickey’, 24, 120)
Questions?
Function Categories

Oracle identifies six different categories of functions:

- Aggregate
- Single-row
- Analytic
- Object Reference
- Model Function
- User Defined

Our focus will be on Aggregate, Single-row, and Analytic functions.
AGGREGATE FUNCTIONS
Aggregate Functions

Operate against a collection of values, but return a single, summarizing value.

- **AVG** - computes the average of values in a column or an expression
- **SUM** - computes the sum (both AVG and SUM work with numeric values and ignore NULL values)
- **COUNT** - counts all rows defined in an expression
- **MIN** - identifies the minimum value in a column by the expression
- **MAX** - finds the maximum value in a column by the expression
Aggregate and Single-row Functions

The number of values an aggregate function processes may vary, depending on the number of rows queried from the table.

This unique process makes aggregate functions different from single-row functions, which require a fixed number and fixed type of parameters.
Aggregate and Single-row Functions

Aggregate and Single-row functions complement each other. They both can be used in the following:

- SELECT statement (in the select list)
- WHERE clauses
- HAVING clauses
SINGLE-ROW FUNCTIONS
Single-row Functions

• Operate on a single value and then return a single value.

• They can be used wherever an expression is valid.

• They can be divided into different logical categories.
Single-row Functions - Types

The different types of Single-row functions are:

- Numeric
- String/Character
- Conversion
- Date and Time
- Advanced
Questions?
Numeric Functions - **CEIL/FLOOR**

Numeric: performs operations on numeric values and returns numeric values, accurate to 38 decimal points

- **CEIL** - returns the smallest integer value that is greater than or equal to a number
  - `ceil(number)`
    - `ceil(21.3) = 22`
    - `ceil(15.9) = 16`
    - `ceil(-8.9) = -8`

- **FLOOR** - returns the largest integer value that is equal to or less than a number
  - `floor(number)`
    - `floor(21.6) = 21`
    - `floor(15.9) = 15`
    - `floor(-8.9) = -9`
Numeric Functions - MOD/REMAINDER

• MOD - returns the remainder of m divided by n (and returns m if n is 0)
  – mod(m,n)
    • Two functions for the price of one (uses FLOOR functionality)
    • Second function applied when dealing with decimals
    • See REMAINDER

• REMAINDER - returns the remainder of m divided by n
  – remainder(m,n)
    • New 10g function
    • Two functions for the price of one (uses ROUND functionality)
    • remainder(16,3) = 1
    • remainder(16,6) = 4
    • remainder(16,0) = 16
    • remainder(-16,3) = -1
Numeric Functions - *ROUND/TRUNC*

- **ROUND** - returns a number rounded to a certain number of decimal points
  - `round(number,[decimal places])`
    - *number* is the number to round and *decimal_places* is the number of places rounded to (if omitted, default is 0)
    - `round(123.456) = 123`
    - `round(123.456,1) = 123.5`
    - `round(123.456,2) = 123.46`

- **TRUNC** - returns a number truncated to a certain number of decimal points
  - `trunc(number,[decimal places])`
    - *number* is the number to round and *decimal_places* is the number of places rounded to (if omitted, default is 0)
    - `trunc(123.456,1) = 123.4`
    - `trunc(123.456,-1) = 120`
Questions?
String Functions - **CONCAT**

String (also referred to as Character): perform operations on a string (char/varchar) input value and return a string or numeric value

- **CONCAT** - appends two or more literal expressions, column values or variables together into one string
  - `(string1 || string2 || string_n)` or `concat(string1,string2)`
  - `A.FIRST_NAME || A.LAST_NAME = MickeyMouse`
  - `A.FIRST_NAME || ' ' || A.LAST_NAME = Mickey Mouse`
  - `'NAME:'||A.FIRST_NAME || CASE WHEN LENGTH(A.MIDDLE_NAME) = 1 AND A.MIDDLE_NAME <> ' ' THEN ' ' || A.MIDDLE_NAME || ' ' WHEN LENGTH(A.MIDDLE_NAME) > 1 THEN ' ' || A.MIDDLE_NAME ELSE '' END || ' ' || A.LAST_NAME || CASE WHEN A.NAME_SUFFIX <> ' ' AND A.NAME_SUFFIX NOT LIKE 'I_' THEN A.NAME_SUFFIX || '.' ELSE A.NAME_SUFFIX END`
String Functions - INITCAP/INSTR

- INITCAP - converts a string to initial capital letters
  - initcap(string1)
    - initcap('mickey mouse') = Mickey Mouse
    - initcap('MI NNI E MOUSE') = Minnie Mouse

- INSTR - returns the location of a substring in a string
  - instr(string1,string2,[start_position],[nth_appearance])
    - string1 is the string to search and string2 is the substring to search for in string1
    - start_position is the position in string1 where the search begins (if omitted, default is 1 - first position in string) and nth_appearance is the nth appearance of string2 (if omitted, default is 1)
    - instr('Mickey','c') = 3 (first occurrence of the letter c, as in “C you real soon…”)
    - instr('Mickey Mousey','y',1,2) = 13 (second occurrence of the letter Y, as in, “Y, because we like you…”)
String Functions -  **LOWER/REPLACE**

- **LOWER** - converts a string to all lowercase characters
  - `lower(string1)`
    - Similar to `initcap` but focusing on the entire string
    - `lower('Mickey Mouse') = mickey mouse`
    - `lower('MINNIE MOUSE') = minnie mouse`

- **REPLACE** - replaces a sequence of characters in a string with another set of characters
  - `replace(string1,string_to_replace,[replacement_string])`
    - `string1` is the string being affected and `string_to_replace` is the string which will be searched for in `string1`
    - `replacement_string` is optional (if omitted, the replace function removes all occurrences of `string_to_replace` and returns the resulting string)
    - `replace('Mickey the Rat','Rat','Mouse') = Mickey the Mouse`
String Functions - *SOUNDEX/XLAT*

- **SOUNDEX** - returns a string containing the phonetic representation (the way it sounds) of the string
  - `soundex(string1)`
    - Allows for the comparison of words that are spelled differently, but sound alike in English
    - `soundex('Jon') = John, Jon, Jean-Pierre, Jonny, Johnnie`
    - `soundex(A.FIRST_NAME) = soundex('John')`

- **TRANSLATE** - converts a string from one character set to another
  - `translate(string1,string_to_replace,[replacement_string])`
    - `string1` is the string being affected and `string_to_replace` is the string which will be searched for in `string1`
    - All characters in the `string_to_replace` will be replaced with the corresponding character in the `replacement_string`
    - Similar to REPLACE, except TRANSLATE provides single-char, one-to-one substitution instead of string substitutions
    - `translate('Foggy','Fgg','Gof') = Goofy`
String Functions - TRIM/UPPER

• TRIM - removes leading characters, trailing characters or both from a character string
  - trim([leading| trailing| both[trim_character]]string1)
    • leading removes *trim_string* from front of string1
    • trailing removes *trim_string* from end of string1
    • both removes *trim_string* from front and end of string1
    • trim(leading ‘$’ from ‘$123.45’) = 123.45
    • trim(trailing ‘.’ from ‘Mr.’) = Mr
    • trim(both ‘.’ from ‘Mr. Jones Jr.’) = Mr. Jones Jr

• UPPER - converts a string to all uppercase characters
  - upper(string1)
    • upper(‘Mickey Mouse’) = MICKEY MOUSE
    • upper(‘minnie mouse’) = MINNIE MOUSE
String Functions - \textit{LENGTH}

- \textbf{LENGTH} - returns the number of characters in a string or field.
  - \texttt{LENGTH(char)}
    - It returns a Number.
    - It counts all characters including trailing blanks.

- \texttt{LENGTH(‘Mickey Mouse’) = 12}
- \texttt{LENGTH(A.EMPLID) = 8}
String Functions - **SUBSTR**

- **SUBSTR** - extracts a portion of a string or field.
  - `SUBSTR(char, position [, substring_length ])
    - `position` is the Starting Position.
    - If position is 0, then it is treated as 1.
    - If position is positive, then the count starts from the beginning.
    - If position is negative, then it starts from the end and counts backward.
    - `substring_length` is the number of characters to extract
  - `SUBSTR('ABCDEFG', 3, 4) = CDEF`
  - `SUBSTR('ABCDEFG', -5, 3) = CDE`
Questions?
Conversion Functions

Conversion: Change or convert values from one data type to another (character to numeric, numeric to character, character to date or date to character)

Note: There are two things you should notice regarding the differences between numeric data types and character string types:

1. Arithmetic expressions and functions can be used on numeric values.
2. Numeric values are right-justified, whereas character string data types are left-justified in the output result.
Conversion Functions (continued)

• **TO_CHAR** - converts a number or date to a string
  - `to_char(value,[format_mask])`
  - *value* is either a number or date that will be converted to a string
  - *format_mask* is the format used to convert the value to a string
  - `to_char(1234.567, '9999.9') = 1234.5`
  - `to_char(1234.567, '9,999.99') = 1,234.56`
  - `to_char(1234.56, '$9,999.00') = $1,234.56`
  - `to_char(23, '000099') = 000023`
  - `to_char(sysdate, 'yyyy/mm/dd') = 2007/03/11`
  - `to_char(sysdate, 'Month DD, YYYY') = March 11, 2007`

• **TO_DATE** - converts a string to a date
  - `to_date(string1,[format_mask])`
  - *string1* is the string that will be converted to a date
  - *format_mask* is the format that will be used to convert string1 to a date
  - `to_date('39152','MMDDYY') = 03/11/07`
Questions?
Date and Time Functions

Date and Time: Perform operations on a date and time input values and return string, numeric, or date and time values

- **SYSDATE** - returns the current system date and time on your local database
  - `sysdate`
    - Let's use March 11, 2007 (03-11-07)
    - `to_char(sysdate - 30, 'MM-DD-YY') = 02-09-07`

- **ADD_MONTHS** - returns a date plus $n$ months
  - `add_months(date1,n)`
    - `add_months('11-Mar-07',3) = 11-Jun-07`
    - `add_months('11-Mar-07',-3) = 11-Dec-06`
Date and Time Functions - *MONTHS_BETWEEN*

- **MONTHS_BETWEEN** - returns number of months between two dates.

  - **MONTHS_BETWEEN**(date1, date2)

If today’s date = March 05, 2007 then

**MONTHS_BETWEEN**(‘12-MAR-09’, SYSDATE)

= 24.203837365
Date and Time Functions - NEXT_DAY

- NEXT_DAY - returns the date of the first weekday named that is later than the date specified.
  - NEXT_DAY(date, char)

<table>
<thead>
<tr>
<th>Career</th>
<th>Term</th>
<th>Short Desc</th>
<th>Term Begin Date</th>
<th>The Next Tuesday</th>
</tr>
</thead>
<tbody>
<tr>
<td>UGRD</td>
<td>1071</td>
<td>Win 2007</td>
<td>01/03/2007</td>
<td>01/09/2007</td>
</tr>
<tr>
<td>UGRD</td>
<td>1073</td>
<td>Spr 2007</td>
<td>03/27/2007</td>
<td>04/03/2007</td>
</tr>
<tr>
<td>UGRD</td>
<td>1076</td>
<td>Sum 2007</td>
<td>06/18/2007</td>
<td>06/19/2007</td>
</tr>
<tr>
<td>UGRD</td>
<td>1083</td>
<td>Spr 2008</td>
<td>03/25/2008</td>
<td>04/01/2008</td>
</tr>
<tr>
<td>UGRD</td>
<td>1086</td>
<td>Sum 2008</td>
<td>06/25/2008</td>
<td>07/01/2008</td>
</tr>
<tr>
<td>UGRD</td>
<td>1089</td>
<td>Fall 2008</td>
<td>09/25/2008</td>
<td>09/30/2008</td>
</tr>
</tbody>
</table>

TO_CHAR(NEXT_DAY(TO_DATE((A.TERM_BEGIN_DT), 'YYYY-MM-DD'),'TUESDAY'), 'YYYY-MM-DD')
Questions?
Advanced Functions

Only for the brave and adventurous PeopleSoft query writers; functions to stimulate your creative/analytical mind:

• GREATEST / LEAST
• NVL / NVL2
• ROWNUM
• COALESCE
• DECODE
• CASE
Advanced Functions - GREATEST/LEAST

• **GREATEST** - returns the greatest from a list of one or more expressions.
  
  - GREATEST(expr [, expr ]...)

• **LEAST** - returns the least from a list of expressions.
  
  - LEAST(expr [, expr ]...)
Advanced Functions - *NVL/NVL2*

- **NVL** - allows substitution of a value when a null value is encountered.
  - `NVL(string1, replace_with)`
    - `string1` is the string to be tested for a null value and `replace_with` is the value returned if `string1` is null.
    - `NVL(course_gpa,'Grade Pending')`
      - if `course_gpa` is null then Grade Pending is returned otherwise `course_gpa` value is returned.

- **NVL2** - allows the substitution of a value when a null value is encountered, as well as when a non-null value is encountered.
  - `NVL2(string1, value_if_not_null, value_if_null)`
    - `string1` is the string to be tested for a null value.
    - `value_if_not_null` is the value returned if `string1` is not null and `value_if_null` is the value returned if `string1` is null.
    - `NVL2(FERPA,'Do Not Disclose','Disclose')`

*(NVL2 extends the functionality of NVL by letting you determine the value returned based on whether something is null or not null.)*
Advanced Functions - *ROWNUM* (p1)

- **ROWNUM** - assigns a number indicating the order in which each row is returned by a query.

<table>
<thead>
<tr>
<th>ROWNUM</th>
<th>ID</th>
<th>Career</th>
<th>Career Nbr</th>
<th>Term</th>
<th>Prim Prog</th>
<th>Take Prgrs</th>
<th>GPA</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22393</td>
<td>UGRD</td>
<td>0</td>
<td>1069</td>
<td>UG</td>
<td>14.000</td>
<td>2.964</td>
<td>2.964</td>
</tr>
<tr>
<td>2</td>
<td>22519</td>
<td>UGRD</td>
<td>1</td>
<td>1069</td>
<td>UG</td>
<td>15.000</td>
<td>3.100</td>
<td>3.100</td>
</tr>
<tr>
<td>3</td>
<td>22647</td>
<td>UGRD</td>
<td>0</td>
<td>1069</td>
<td>INTL</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>4</td>
<td>22752</td>
<td>UGRD</td>
<td>0</td>
<td>1069</td>
<td>NM</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>5</td>
<td>22771</td>
<td>UGRD</td>
<td>0</td>
<td>1069</td>
<td>INTL</td>
<td>16.000</td>
<td>1.250</td>
<td>2.250</td>
</tr>
<tr>
<td>6</td>
<td>22771</td>
<td>UGRD</td>
<td>0</td>
<td>1069</td>
<td>INTL</td>
<td>16.000</td>
<td>3.650</td>
<td>3.633</td>
</tr>
<tr>
<td>7</td>
<td>22771</td>
<td>UGRD</td>
<td>0</td>
<td>1069</td>
<td>NM</td>
<td>13.000</td>
<td>3.000</td>
<td>3.391</td>
</tr>
<tr>
<td>8</td>
<td>22694</td>
<td>UGRD</td>
<td>0</td>
<td>1069</td>
<td>NM</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>9</td>
<td>22785</td>
<td>UGRD</td>
<td>0</td>
<td>1069</td>
<td>INTL</td>
<td>16.000</td>
<td>3.325</td>
<td>3.408</td>
</tr>
<tr>
<td>10</td>
<td>22471</td>
<td>UGRD</td>
<td>0</td>
<td>1069</td>
<td>UG</td>
<td>0.000</td>
<td>0.000</td>
<td>3.973</td>
</tr>
</tbody>
</table>
Advanced Functions - **ROWNUM** (p2)

Query Tip # 1: **LIMIT NUMBER OF ROWS RETURNED**

**ROWNUM <= 100**

- Do not use Equal to (=) or Greater Than (>).
- If an ORDER BY clause follows ROWNUM in the same query, then the rows will be reordered by the ORDER BY clause.
Advanced Functions (continued)

The next three functions have similar functionality, yet each subsequent function is more powerful than the previous one.

- COALESCE
- DECODE
- CASE

- All three perform ‘IF-THEN’ operations
Advanced Functions - **COALESCE**

- **COALESCE** - returns the first non-null expression in the list (if all expressions evaluate to null, then the coalesce function will return null)
  - `coalesce(expr1, expr2, ..., expr_n)
    - ‘IF-THEN’ functionality
    - `coalesce(mickey, minnie, goofy)`
      - IF mickey exists (not null) THEN result = mickey;
      - ELSIF minnie exists (not null) THEN result = minnie;
      - ELSIF goofy exists (not null) THEN result = goofy;
      - ELSE result = null;
      - END IF
    - The `coalesce` function compares each value one by one
Advanced Functions - **DECODE**

- **DECODE** - performs the functionality of an ‘IF-THEN-ELSE’ statement, also comparing each value, one by one, but now with specific search criteria
  - `decode(expression, search, result[, search, result]...[, default])`
  - **expression** is the value to compare, **search** is the value that is compared to expression and **result** is the value returned, if expression equals search
  - **default** is optional, if no matches are found **decode** returns the default value (unless omitted, then statement returns null)
    - `decode(char_id,01,'Mickey',02,'Minnie',03,'Goofy','Donald')`
    - IF `char_id` = 01 THEN `result` = Mickey;
    - ELSIF `char_id` = 02 THEN `result` = Minnie;
    - ELSIF `char_id` = 03 THEN `result` = Goofy;
    - ELSE `result` = Donald;
    - END IF
Advanced Functions - **CASE**

- **CASE** - performs the functionality of an “IF-THEN-ELSE” statement with greater possibilities.
  - **CASE expression**
    - WHEN condition\(_1\) THEN result\(_1\)
    - WHEN condition\(_2\) THEN result\(_2\)
    - WHEN condition\(_n\) THEN result\(_n\)
    - ELSE result END
IF .... THEN .... ELSE

CASE WHEN ..... THEN ..... ELSE ..... END

• CASE expressions are ANSI-standard.
• CASE was introduced in Oracle8i and enhanced in Oracle9i.
• CASE is part of the SQL standard, whereas DECODE is not.
• Thus, the use of CASE is preferable.
CASE WHEN ..... THEN ..... ELSE ..... END

CASE WHEN B.FERPA = 'Y'

THEN 'FERPA - DO NOT DISCLOSE'

ELSE "

END

CASE WHEN B.FERPA = 'Y' THEN 'FERPA - DO NOT DISCLOSE' ELSE " END
Advanced Functions -  

**CASE, LENGTH, SUBSTR, ||, TRIM**

**Zip Code Plus 4**

```sql
CASE WHEN (B.COUNTRY = 'USA'  AND LENGTH(TRIM(B.POSTAL)) = 9)
    THEN SUBSTR(B.POSTAL,1,5) || '-' || SUBSTR(B.POSTAL,6,4)
ELSE TRIM(B.POSTAL)
END
```

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>989267405</td>
<td>98926-7405</td>
</tr>
<tr>
<td>98020</td>
<td>98020</td>
</tr>
<tr>
<td>98948-3722</td>
<td>98948-3722</td>
</tr>
</tbody>
</table>
CASE WHEN (SUM(C.UNT_TRNSFR * C.GRD_PTS_PER_UNIT) / SUM(C.UNT_TRNSFR)) IS NULL THEN A.CUM_GPA ELSE (CASE WHEN SUM(C.UNT_TRNSFR) IS NOT NULL OR A.TOT_TAKEN_GPA IS NOT NULL THEN (CASE WHEN SUM(C.GRD_PTS_PER_UNIT * C.UNT_TRNSFR) IS NULL THEN A.TOT_GRADE_POINTS ELSE SUM(C.GRD_PTS_PER_UNIT * C.UNT_TRNSFR) + A.TOT_GRADE_POINTS END / CASE WHEN SUM(C.UNT_TRNSFR) IS NULL THEN A.TOT_TAKEN_GPA ELSE SUM(C.UNT_TRNSFR) + A.TOT_TAKEN_GPA END) ELSE 0 END) END
CASE WHEN (SUM(C.UNT_TRNSFR * C.GRD_PTS_PER_UNIT) / SUM(C.UNT_TRNSFR)) IS NULL THEN A.CUM_GPA
ELSE (CASE WHEN SUM(C.UNT_TRNSFR) IS NOT NULL OR A.TOT_TAKEN_GPA IS NOT NULL THEN (CASE WHEN SUM(C.GRD_PTS_PER_UNIT * C.UNT_TRNSFR) IS NULL THEN A.TOT_GRADE_POINTS ELSE SUM(C.GRD_PTS_PER_UNIT * C.UNT_TRNSFR) + A.TOT_GRADE_POINTS END / CASE WHEN SUM(C.UNT_TRNSFR) IS NULL THEN A.TOT_TAKEN_GPA ELSE SUM(C.UNT_TRNSFR) + A.TOT_TAKEN_GPA END)
END / CASE WHEN SUM(C.UNT_TRNSFR) IS NULL THEN A.TOT_TAKEN_GPA ELSE SUM(C.UNT_TRNSFR) + A.TOT_TAKEN_GPA END)
ELSE 0
END)
Notes:

- Oracle Database uses short-circuit evaluation, so place the MOST restrictive condition FIRST.

- Case expressions enable use of full mathematic & SQL logic. (=, <>, >, <, +, -, *, /, AND, OR, IN, BETWEEN, etc.)

- The maximum number of arguments in a CASE expression is 255, and each WHEN ... THEN pair counts as two arguments. To avoid exceeding the limit of 128 choices, you can nest CASE expressions.

CASE WHEN ..... THEN ..... ELSE ..... END
Questions?
Analytic Functions - Definition

• Analytic functions compute an aggregate value based on a group of rows.

  ▪ They differ from aggregate functions in that they return multiple rows for each group.
  
  ▪ The group of rows is called a window.
  
  ▪ Analytic functions are the last set of operations performed in a query except for the final ORDER BY clause. All joins and all WHERE, GROUP BY, and HAVING clauses are completed before the analytic functions are processed.
  
  ▪ Analytic functions are commonly used to compute cumulative, moving, centered, and reporting aggregates.
  
  ▪ Calculations are independent of output.
Analytic Functions - Syntax

Partition Statement Syntax

..... (.....) OVER (PARTITION BY .....)

COUNT (.....) OVER (PARTITION BY .....)

COUNT (A.EMPLID) OVER (PARTITION BY A.STRM)
Analytic Functions - **COUNT**

`COUNT (A.EMPLID) OVER (PARTITION BY A.STRM)`

<table>
<thead>
<tr>
<th>ID</th>
<th>Career</th>
<th>Strt Level</th>
<th>Take Prgrs</th>
<th>Term</th>
<th>Count per Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>0.000</td>
<td>1051</td>
<td>540</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>9.000</td>
<td>1053</td>
<td>507</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>4.000</td>
<td>1056</td>
<td>479</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>5.000</td>
<td>1059</td>
<td>754</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>9.000</td>
<td>1051</td>
<td>540</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1053</td>
<td>507</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1056</td>
<td>479</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1059</td>
<td>754</td>
</tr>
<tr>
<td>443</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>754</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1051</td>
<td>540</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1053</td>
<td>507</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>12.000</td>
<td>1056</td>
<td>479</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>40</td>
<td>13.000</td>
<td>1059</td>
<td>754</td>
</tr>
<tr>
<td>690</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>754</td>
</tr>
<tr>
<td>768</td>
<td>UGRD</td>
<td>30</td>
<td>15.000</td>
<td>1051</td>
<td>540</td>
</tr>
<tr>
<td>768</td>
<td>UGRD</td>
<td>30</td>
<td>18.000</td>
<td>1053</td>
<td>507</td>
</tr>
<tr>
<td>768</td>
<td>UGRD</td>
<td>40</td>
<td>11.000</td>
<td>1056</td>
<td>479</td>
</tr>
<tr>
<td>768</td>
<td>UGRD</td>
<td>40</td>
<td>24.000</td>
<td>1059</td>
<td>754</td>
</tr>
</tbody>
</table>

- Function operation and grouping happens after all query criteria have been met.
- Calculation is independent of output.
- The value repeats for each row with that group/partition.

Rows Fetched = 2280
**Analytic Functions - Change Group**

COUNT (A.EMPLID) OVER (PARTITION BY A.STRM)

COUNT (A.EMPLID) OVER (PARTITION BY A.ACAD_LEVEL_BOT)

<table>
<thead>
<tr>
<th>ID</th>
<th>Career</th>
<th>Strt Level</th>
<th>Take Prgrs</th>
<th>Term</th>
<th>Count per Term</th>
<th>Count per Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>0.000</td>
<td>1051</td>
<td>540</td>
<td>907</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>9.000</td>
<td>1053</td>
<td>507</td>
<td>907</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>4.000</td>
<td>1056</td>
<td>479</td>
<td>907</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>5.000</td>
<td>1059</td>
<td>754</td>
<td>907</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>9.000</td>
<td>1051</td>
<td>540</td>
<td>563</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1053</td>
<td>507</td>
<td>563</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1056</td>
<td>479</td>
<td>563</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1059</td>
<td>754</td>
<td>563</td>
</tr>
<tr>
<td>443</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>754</td>
<td>563</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1051</td>
<td>540</td>
<td>907</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1053</td>
<td>507</td>
<td>907</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1056</td>
<td>479</td>
<td>907</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>12.000</td>
<td>1059</td>
<td>754</td>
<td>299</td>
</tr>
<tr>
<td>690</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1051</td>
<td>540</td>
<td>563</td>
</tr>
<tr>
<td>728</td>
<td>UGRD</td>
<td>28</td>
<td>15.000</td>
<td>1051</td>
<td>540</td>
<td>907</td>
</tr>
</tbody>
</table>

Rows Fetched = 2280
Analytic Functions - *Distinct*

COUNT (A.EMPLID) OVER (PARTITION BY A.STRM)

COUNT (A.EMPLID) OVER (PARTITION BY A.ACAD_LEVEL_BOT)

COUNT (DISTINCT A.EMPLID) OVER (PARTITION BY A.ACAD_LEVEL_BOT)

<table>
<thead>
<tr>
<th>ID</th>
<th>Career</th>
<th>Strt Level</th>
<th>Take Prgrs</th>
<th>Term</th>
<th>Count per Term</th>
<th>Count per Grade</th>
<th>Distinct Count per Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>0.000</td>
<td>1051</td>
<td>540</td>
<td>907</td>
<td>335</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>9.000</td>
<td>1053</td>
<td>507</td>
<td>907</td>
<td>335</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>4.000</td>
<td>1056</td>
<td>479</td>
<td>907</td>
<td>335</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>5.000</td>
<td>1059</td>
<td>754</td>
<td>907</td>
<td>335</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>9.000</td>
<td>1051</td>
<td>540</td>
<td>563</td>
<td>411</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1053</td>
<td>507</td>
<td>563</td>
<td>411</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1056</td>
<td>479</td>
<td>563</td>
<td>411</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1059</td>
<td>754</td>
<td>563</td>
<td>411</td>
</tr>
<tr>
<td>443</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>754</td>
<td>563</td>
<td>411</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1051</td>
<td>540</td>
<td>907</td>
<td>335</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1053</td>
<td>507</td>
<td>907</td>
<td>335</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>12.000</td>
<td>1056</td>
<td>479</td>
<td>907</td>
<td>335</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>40</td>
<td>13.000</td>
<td>1059</td>
<td>754</td>
<td>299</td>
<td>130</td>
</tr>
<tr>
<td>690</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>754</td>
<td>563</td>
<td>411</td>
</tr>
<tr>
<td>789</td>
<td>UGRD</td>
<td>30</td>
<td>15.000</td>
<td>1051</td>
<td>540</td>
<td>907</td>
<td>335</td>
</tr>
</tbody>
</table>

Rows Fetched = 2280
### Analytic Functions - *Multiple Groups*

<table>
<thead>
<tr>
<th>ID</th>
<th>Career</th>
<th>Start Level</th>
<th>Take Prgrs</th>
<th>Term</th>
<th>Count per Term</th>
<th>Count per Grade</th>
<th>Distinct Count per Grade</th>
<th>Distinct # in Grade per Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>0.000</td>
<td>1051</td>
<td>540</td>
<td>907</td>
<td>335</td>
<td>245</td>
</tr>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>9.000</td>
<td>1053</td>
<td>507</td>
<td>907</td>
<td>335</td>
<td>237</td>
</tr>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>4.000</td>
<td>1056</td>
<td>479</td>
<td>907</td>
<td>335</td>
<td>203</td>
</tr>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>5.000</td>
<td>1059</td>
<td>754</td>
<td>907</td>
<td>335</td>
<td>222</td>
</tr>
<tr>
<td>2378</td>
<td>UGRD</td>
<td>10</td>
<td>9.000</td>
<td>1051</td>
<td>540</td>
<td>563</td>
<td>411</td>
<td>129</td>
</tr>
<tr>
<td>2378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1053</td>
<td>507</td>
<td>563</td>
<td>411</td>
<td>100</td>
</tr>
<tr>
<td>2378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1056</td>
<td>479</td>
<td>563</td>
<td>411</td>
<td>55</td>
</tr>
<tr>
<td>2378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1059</td>
<td>754</td>
<td>563</td>
<td>411</td>
<td>279</td>
</tr>
<tr>
<td>2443</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>754</td>
<td>563</td>
<td>411</td>
<td>279</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1051</td>
<td>540</td>
<td>907</td>
<td>335</td>
<td>245</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1053</td>
<td>507</td>
<td>907</td>
<td>335</td>
<td>237</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>12.000</td>
<td>1056</td>
<td>479</td>
<td>907</td>
<td>335</td>
<td>203</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>40</td>
<td>13.000</td>
<td>1059</td>
<td>754</td>
<td>299</td>
<td>130</td>
<td>119</td>
</tr>
<tr>
<td>690</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>754</td>
<td>563</td>
<td>411</td>
<td>279</td>
</tr>
</tbody>
</table>

Rows Fetched = 2280
Analytic Functions - *SUM*

**Total Credits by ID**

\[
\text{SUM} (\text{A.UNT\_TAKEN\_PRGRSS}) \text{ OVER (PARTITION BY A.EMPLID)}
\]

<table>
<thead>
<tr>
<th>ID</th>
<th>Career</th>
<th>Strt Level</th>
<th>Take Prgrs</th>
<th>Term</th>
<th>Total Units per ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>0.000</td>
<td>1051</td>
<td>18.000</td>
</tr>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>9.000</td>
<td>1053</td>
<td>18.000</td>
</tr>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>4.000</td>
<td>1056</td>
<td>18.000</td>
</tr>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>5.000</td>
<td>1059</td>
<td>18.000</td>
</tr>
<tr>
<td>2378</td>
<td>UGRD</td>
<td>10</td>
<td>9.000</td>
<td>1051</td>
<td>9.000</td>
</tr>
<tr>
<td>2378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1053</td>
<td>9.000</td>
</tr>
<tr>
<td>2378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1056</td>
<td>9.000</td>
</tr>
<tr>
<td>2378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1059</td>
<td>9.000</td>
</tr>
<tr>
<td>2443</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>15.000</td>
</tr>
<tr>
<td>2612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1051</td>
<td>51.000</td>
</tr>
<tr>
<td>2612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1053</td>
<td>51.000</td>
</tr>
<tr>
<td>2612</td>
<td>UGRD</td>
<td>30</td>
<td>12.000</td>
<td>1056</td>
<td>51.000</td>
</tr>
<tr>
<td>2612</td>
<td>UGRD</td>
<td>40</td>
<td>13.000</td>
<td>1059</td>
<td>51.000</td>
</tr>
<tr>
<td>2690</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>15.000</td>
</tr>
<tr>
<td>2768</td>
<td>UGRD</td>
<td>30</td>
<td>15.000</td>
<td>1051</td>
<td>68.000</td>
</tr>
</tbody>
</table>

Rows Fetched = 2280
**Analytic Functions - Query Tip #2**

**Group by a Constant**

- `COUNT (A.EMPLID) OVER ( )`
- `COUNT (DISTINCT A.EMPLID) OVER ( )`

---

### Analytic Functions Examples

<table>
<thead>
<tr>
<th>ID</th>
<th>Career</th>
<th>Sttl Level</th>
<th>Take Prgrs</th>
<th>Term</th>
<th>Total Rows in Query</th>
<th>Distinct ID Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>0.000</td>
<td>1051</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>9.000</td>
<td>1053</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>4.000</td>
<td>1056</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>2261</td>
<td>UGRD</td>
<td>30</td>
<td>5.000</td>
<td>1059</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>9.000</td>
<td>1051</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1053</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1056</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1059</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>443</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1051</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1053</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>12.000</td>
<td>1056</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>40</td>
<td>13.000</td>
<td>1059</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>690</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>2280</td>
<td>943</td>
</tr>
<tr>
<td>768</td>
<td>UGRD</td>
<td>30</td>
<td>15.000</td>
<td>1051</td>
<td>2280</td>
<td>943</td>
</tr>
</tbody>
</table>

**Rows Fetched = 2280**
# Analytic Functions - Query Tip #3

## Count Multiple ID’s

**COUNT (A.EMPLID) OVER (PARTITION BY A.EMPLID)**

<table>
<thead>
<tr>
<th>ID</th>
<th>Career</th>
<th>Strt Level</th>
<th>Take Prgrs</th>
<th>Term</th>
<th>Distinct ID Count</th>
<th># of Rows per ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>0.000</td>
<td>1051</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>9.000</td>
<td>1053</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>4.000</td>
<td>1053</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>261</td>
<td>UGRD</td>
<td>30</td>
<td>5.000</td>
<td>1059</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>9.000</td>
<td>1051</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1053</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1056</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>378</td>
<td>UGRD</td>
<td>10</td>
<td>0.000</td>
<td>1059</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>443</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>943</td>
<td>1</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1051</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>13.000</td>
<td>1053</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>30</td>
<td>12.000</td>
<td>1056</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>612</td>
<td>UGRD</td>
<td>40</td>
<td>13.000</td>
<td>1059</td>
<td>943</td>
<td>4</td>
</tr>
<tr>
<td>690</td>
<td>UGRD</td>
<td>10</td>
<td>15.000</td>
<td>1059</td>
<td>943</td>
<td>1</td>
</tr>
<tr>
<td>768</td>
<td>UGRD</td>
<td>30</td>
<td>15.000</td>
<td>1051</td>
<td>943</td>
<td>4</td>
</tr>
</tbody>
</table>

Rows Fetched = 2280
### Analytic Functions - ORDER BY

**PERCENT_RANK() OVER (PARTITION BY A.ACAD_LEVEL_BOT ORDER BY A.CUM_GPA DESC)**

<table>
<thead>
<tr>
<th>ID</th>
<th>Career</th>
<th>Strt Level</th>
<th>GPA</th>
<th>Percent Rank In Class</th>
<th>% Rank in Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>601</td>
<td>UGRD</td>
<td>10</td>
<td>0.927</td>
<td>.974</td>
<td>97.4</td>
</tr>
<tr>
<td>317</td>
<td>UGRD</td>
<td>10</td>
<td>0.836</td>
<td>.978</td>
<td>97.8</td>
</tr>
<tr>
<td>983</td>
<td>UGRD</td>
<td>10</td>
<td>0.617</td>
<td>.983</td>
<td>98.3</td>
</tr>
<tr>
<td>539</td>
<td>UGRD</td>
<td>10</td>
<td>0.591</td>
<td>.987</td>
<td>98.7</td>
</tr>
<tr>
<td>783</td>
<td>UGRD</td>
<td>10</td>
<td>0.425</td>
<td>.991</td>
<td>99.1</td>
</tr>
<tr>
<td>545</td>
<td>UGRD</td>
<td>10</td>
<td>0.206</td>
<td>.995</td>
<td>99.5</td>
</tr>
<tr>
<td>218</td>
<td>UGRD</td>
<td>10</td>
<td>0.133</td>
<td>1</td>
<td>100.0</td>
</tr>
<tr>
<td>087</td>
<td>UGRD</td>
<td>20</td>
<td>4.000</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>121</td>
<td>UGRD</td>
<td>20</td>
<td>4.000</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>227</td>
<td>UGRD</td>
<td>20</td>
<td>4.000</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>578</td>
<td>UGRD</td>
<td>20</td>
<td>3.759</td>
<td>.058</td>
<td>5.8</td>
</tr>
<tr>
<td>408</td>
<td>UGRD</td>
<td>20</td>
<td>3.684</td>
<td>.078</td>
<td>7.8</td>
</tr>
<tr>
<td>582</td>
<td>UGRD</td>
<td>20</td>
<td>3.683</td>
<td>.098</td>
<td>9.8</td>
</tr>
<tr>
<td>193</td>
<td>UGRD</td>
<td>20</td>
<td>3.667</td>
<td>.117</td>
<td>11.7</td>
</tr>
<tr>
<td>356</td>
<td>UGRD</td>
<td>20</td>
<td>3.628</td>
<td>.137</td>
<td>13.7</td>
</tr>
</tbody>
</table>

\[(\text{PERCENT\_RANK()}()\ \text{OVER}(\text{PARTITION \ BY}\ A.ACAD\_LEVEL\_BOT \ \text{ORDER \ BY} A.CUM\_GPA \ \text{DESC})) \times 100\]
Analytic Functions - Syntax Review

..... (.....) OVER (PARTITION BY ..... )

..... () OVER ()

..... (.....) OVER (PARTITION BY ..... ORDER BY ..... DESC)

ASC | DESC Specify the ordering sequence (ascending or descending).

ASC is the default.

..... (.....) OVER (PARTITION BY ..... ORDER BY ..... DESC NULLS LAST)

NULLS LAST is the default for ascending order.
NULLS FIRST is the default for descending order.
Analytic Functions - \textit{RANK}

- **RANK** - calculates the rank of a value in a group of values.

  - \( \text{RANK}( ) \ \text{OVER} \ ( [ \text{query\_partition\_clause} ] \ \text{order\_by\_clause} ) \)
  
  - Returns the rank as a \textit{NUMBER}.
  - \textit{RANK} computes the rank of each row returned from a query with respect to the other rows returned in \textit{the group}.
  - Rows with equal values for the ranking criteria receive the same rank.
Analytic Functions - *DENSE_RANK*

- **DENSE_RANK** - computes the rank of a row in an ordered group of rows.
  
  - `DENSE_RANK( ) OVER([ query_partition_clause ] order_by_clause)`

  - Returns the rank as a NUMBER.
  - The ranks are consecutive integers beginning with 1.
  - Rank values are not skipped in the event of ties.
  - Rows with equal values for the ranking criteria receive the same rank.
Analytic Functions - *PERCENT_RANK*

- **PERCENT_RANK** - calculates the rank of \( r \) minus 1, divided by 1 less than the number of rows being evaluated (the entire query result set or a partition).

  - `PERCENT_RANK()` OVER ([ query_partition_clause ] order_by_clause)

- The return value is a NUMBER.
- The range of values returned by **PERCENT_RANK** is 0 to 1, inclusive.
- The first row in any set has a **PERCENT_RANK** of 0.
Analytic Functions - \textit{LAG} / \textit{LEAD}

- \textit{LAG} | \textit{LEAD} - provide access to more than one row of a table at the same time without a self join.

Given a series of rows returned from a query and a position of the cursor, (\textit{LAG} | \textit{LEAD}) provides access to a row at a given physical offset (prior| beyond) that position.

- \texttt{LAG(valueExpr [, offset ] [, default ] ) OVER ([ query_partition_clause ] order_by_clause)}

- If you do not specify offset, then its default is 1.
- The optional default value is returned if the offset goes beyond the scope of the window.
- If you do not specify default, then its default is null.
**Analytic Functions** - \( \text{LAG} / \text{LEAD} \) (p2)

**Compare Address Changes**

<table>
<thead>
<tr>
<th>ID</th>
<th>Addr Type</th>
<th>Eff Date</th>
<th>Status</th>
<th>Address 1</th>
<th>Old St. Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001182</td>
<td>MAIL</td>
<td>12/16/2002</td>
<td>A</td>
<td>APDO 37, Calle Del Pilar #38</td>
<td>NO CHANGE</td>
</tr>
<tr>
<td>1001184</td>
<td>MAIL</td>
<td>12/01/2003</td>
<td>A</td>
<td>1939 Nora Springs Court</td>
<td>NO CHANGE</td>
</tr>
<tr>
<td>1001187</td>
<td>MAIL</td>
<td>09/01/1969</td>
<td>A</td>
<td>1804 Abel Pl</td>
<td>NO CHANGE</td>
</tr>
<tr>
<td>1001187</td>
<td>MAIL</td>
<td>06/14/2004</td>
<td>A</td>
<td>2437 Wheaton Drive</td>
<td>1804 Abel Pl</td>
</tr>
<tr>
<td>1001187</td>
<td>MAIL</td>
<td>05/04/2005</td>
<td>A</td>
<td>2437 N. Wheaton CT</td>
<td>2437 Wheaton Drive</td>
</tr>
<tr>
<td>1001204</td>
<td>MAIL</td>
<td>06/01/2003</td>
<td>A</td>
<td>1114 Reo Dr.</td>
<td>NO CHANGE</td>
</tr>
<tr>
<td>1001222</td>
<td>MAIL</td>
<td>07/07/2004</td>
<td>A</td>
<td>107 W 9th Ave</td>
<td>NO CHANGE</td>
</tr>
<tr>
<td>1001322</td>
<td>MAIL</td>
<td>08/13/1975</td>
<td>A</td>
<td>305 N Elliott</td>
<td>NO CHANGE</td>
</tr>
</tbody>
</table>

\[ \text{LAG}(\text{A.ADDRESS1}, 1, '\text{NO CHANGE}') \] \text{OVER} \( \text{PARTITION BY A.EMPLID ORDER BY A.EFFDT} \)
Analytic Functions - \textit{NTILE}

\begin{itemize}
\item \textit{NTILE} - divides an ordered data set into the number of buckets as indicated and assigns the appropriate bucket number to each row.
\end{itemize}

\begin{itemize}
\item NTILE(expr) OVER ([ \text{query\_partition\_clause} ] \text{order\_by\_clause})
\end{itemize}

\begin{itemize}
\item Used to evenly distribute a group into subgroups.
\item The return value is a \texttt{NUMBER}.
\item The number of rows in the buckets can differ by at most 1.
\item The remainder values are distributed one for each bucket, starting with bucket 1.
\end{itemize}

\begin{itemize}
\item NTILE(6) OVER (ORDER BY A.LAST\_NAME)
\end{itemize}
Analytic Functions – ROW_NUMBER

- ROW_NUMBER - assigns a unique number to each row within a group in the ordered sequence of rows specified in the order-by-clause

  - ROW_NUMBER( ) OVER ([ query_partition_clause ] order_by_clause)

- Can perform TOP-N query functionality.
- It is similar to ROWNUM in that it numbers the output rows, although ROWNUM is one unbroken sequence over the whole rowset, and ROW_NUMBER resets back to one for each partition defined within the set.

  - ROW_NUMBER( ) OVER (PARTITION BY A.ACAD_LEVEL_BOT ORDER BY A.UNT_TAKEN_PRGRSS DESC)
Analytic Functions - *ROW_NUMBER* (p2)

Providing Top-$N$ functionality by combining *ROW_NUMBER* with *ROWNUM*:

```sql
SELECT A.EMPLID, A.ACAD_CAREER, A.STRM, A.ACAD_LEVEL_BOT, A.UNT_TAKEN_PRGRSS, ROW_NUMBER() OVER (ORDER BY A.UNT_TAKEN_PRGRSS DESC) FROM PS_STDNT_CAR_TERM A
WHERE A.INSTITUTION = 'PSUNV' AND A.STRM = '0350' AND ROWNUM <= '10'
```
Analytic Functions - \texttt{RATIO\_TO\_REPORT}

- \texttt{RATIO\_TO\_REPORT} - calculates the ratio of a value to the sum of a set of values
  - \texttt{ratio\_to\_report(expr) over ([query partition clause])}
  - if \texttt{expr} is null, then \texttt{ratio\_to\_report} value is null as well
- value set is determined by the query partition clause (if the query partition clause is omitted, ratio-to-report is calculated over all returned rows)
- In this example, we’ll calculate the value of each employee’s hours spent on greeting visitors (by each employee) as compared to the total hours spent by all employees
  - SQL statement syntax:
  - \texttt{Select employee\_name, hours, ratio\_to\_report(hours) over ()}

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>HOURS</th>
<th>RATIO_TO_REPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mickey</td>
<td>20</td>
<td>0.166666667</td>
</tr>
<tr>
<td>Minnie</td>
<td>50</td>
<td>0.416666667</td>
</tr>
<tr>
<td>Goofy</td>
<td>10</td>
<td>0.083333333</td>
</tr>
<tr>
<td>Donald</td>
<td>40</td>
<td>0.333333333</td>
</tr>
</tbody>
</table>
Questions?
Power Combo - Introduction

```
CASE WHEN ..... THEN ..... ELSE ..... END

( ..... ) OVER (PARTITION BY ..... )

CASE WHEN ( ..... ( ..... ) OVER (PARTITION BY ..... ) )
> 0 THEN ..... ELSE ..... END

( CASE WHEN ..... THEN ..... ELSE ..... END )
OVER (PARTITION BY ..... )
```
Power Combo - Example

Total Credits per Person

\[
\text{SUM (B.UNT_PRGRSS) OVER (PARTITION BY A.EMPLID)}
\]

Total Credits per Person as of Date

\[
\text{SUM (CASE WHEN B.ENRL_ADD_DT} \leq :2 \text{ THEN B.UNT_PRGRSS END) OVER (PARTITION BY A.EMPLID)}
\]
Enrollment Status as of Date

CASE WHEN (SUM (CASE WHEN B.ENRL_ADD_DT <= :2 THEN B.UNT_PRGRSS END) OVER (PARTITION BY A.EMPLID)) >= 12 THEN 'Full'

WHEN (SUM (CASE WHEN B.ENRL_ADD_DT <= :2 THEN B.UNT_PRGRSS END) OVER (PARTITION BY A.EMPLID)) BETWEEN 9 AND 11 THEN '3Quarter'

WHEN (SUM (CASE WHEN B.ENRL_ADD_DT <= :2 THEN B.UNT_PRGRSS END) OVER (PARTITION BY A.EMPLID)) BETWEEN 6 AND 8 THEN 'Half'

ELSE 'Less'

END
Intermission
Methodology - Query

1. Identify What Information is Really Needed

2. Determine Criteria Logic

3. Use Appropriate Records, Tables, and Fields

4. Perform Table Dumps to Learn Tables
   a. Identify Key Fields
   b. Develop Criteria for Table
   c. Identify Example/ Sample Data
Methodology - Query (continued)

5. **Create Table/Record Joins**

   Run query after each new table join to compare what has changed – add/lost rows/data.

6. **Verify Data Set**

   Is this the data you want to use?
Methodology - Function Statements (p1)

7. Determine Use of Function Statements
   How Do you want to see it?

8. Identify Data Type
   a. Numbers
   b. Characters
   c. Date

9. Identify Needed Manipulation
   a. Data Type Conversion
   b. Totals
   c. Grouping
   d. If-Then Logic
Methodology - Function Statements (p2)

10. Build & Test in Increments

CASE WHEN (SUM (CASE WHEN B.ENRL_ADD_DT <= :2 THEN B.UNT_PRGRSS END) OVER (PARTITION BY A.EMPLID)) >= 12 THEN 'Full'

CASE WHEN (SUM(C.UNT_TRNSFR)) IS NULL THEN A.CUM_GPA

WHEN (SUM (CASE WHEN B.ENRL_ADD_DT <= :2 THEN B.UNT_PRGRSS END) OVER (PARTITION BY A.EMPLID)) BETWEEN 9 AND 11 THEN '3Quarter'

WHEN (SUM (CASE WHEN B.ENRL_ADD_DT <= :2 THEN B.UNT_PRGRSS END) OVER (PARTITION BY A.EMPLID)) BETWEEN 6 AND 8 THEN 'Half'

WHEN (SUM(C.GRD_PTS_PER_UNIT * C.UNT_TRNSFR)) IS NULL THEN A.TOT_TAKEN_GPA ELSE

ELSE 'Less'

ELSE 0 END

END
Methodology - Function Statements (p3)

11. Query Tip Review:

#1: Limit Number of Rows Returned

\[
\text{ROWNUM} \leq 100
\]

#2: Unique Count

\[
\text{COUNT (DISTINCT A.EMPLID) OVER ()}
\]

#3: Multiple Rows Count

\[
\text{COUNT (A.EMPLID) OVER (PARTITION BY A.EMPLID)}
\]
12. Using/Viewing SQL

What’s REALLY going on?

```
SELECT A.ACAD_CAREER, A.STRM, TO_CHAR(A.TERM_BEGIN_DT,'YYYY-MM-DD'),
       NEXT_DAY(SYSDATE,'TUESDAY')
FROM PS_TERM_TBL_A
WHERE A.ACAD_CAREER = 'UGRD'
```
Questions?
Exploring Further

Travel with us as we Go Deeper into the Mysterious Universe of using Function Statements with the PS Query Tool.
Exploring Further – SQL Clauses

The Four Basic Areas of SQL:

SELECT = Defines Columns & Output Format
FROM = Identifies Source of Data
WHERE = Determines the Rows by Criteria
ORDER BY = Organizes Final Order
Exploring Further – WHERE Clause

Using Function Expressions as Criteria:

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Expression1</th>
<th>Condition Type</th>
<th>Expression 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical</td>
<td>A.INSTITUTION - Academic Institution</td>
<td>equal to</td>
<td>PSUNV</td>
</tr>
<tr>
<td>AND</td>
<td>A.EMPLID - EmplID</td>
<td>equal to</td>
<td>CASE WHEN A.ACAD_CAREER = 'UGRD' AND A.CUM_GPA &gt; 3.6 THEN A.EMPLID END</td>
</tr>
</tbody>
</table>

CASE WHEN A.ACAD_CAREER = 'UGRD' AND A.CUM_GPA > 3.6 THEN A.EMPLID
WHEN A.ACAD_CAREER = 'PBAC' AND A.UNT_TAKEN_PRGRSS > 3 THEN A.EMPLID
WHEN A.ACAD_CAREER = 'GRAD' AND A.CUR_GPA > 2.8 THEN A.EMPLID
END

CASE statements in the criteria! WOW!
Exploring Further: \( 1=1 \)

This profound concept is your key to full SQL access to the WHERE clause!

The Criteria simply states \( 1 = 1 \), but the SQL states:

```
SELECT A.EMPLID, A.ACAD_CAREER, A.ASTNT_CAR, A.IBR, A.ASTNT.OPER, A.ACA_LEVEL_BOT, A.UNT_TAKEN_PRGRSS, A.CUM_GPA
FROM PS_STDNT_CAR_TERM A
WHERE 1 = 1
AND A.STRM = '1061'
AND A.EMPLID > '22800000'
AND A.EMPLID = CASE WHEN A.ACAD_CAREER = 'UGRD' AND A.CUM_GPA > 3.6 THEN A.EMPLID
                WHEN A.ACAD_CAREER = 'PBAC' THEN A.EMPLID
        END
```
Exploring Further –  $1 = 1$ (p2)

By Using an Expression and straight SQL!
Exploring Further – 1=1 (p3)

<table>
<thead>
<tr>
<th>ID</th>
<th>Career</th>
<th>Career Nbr</th>
<th>Term</th>
<th>Strt Level</th>
<th>Take Prgrs</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>22800091</td>
<td>UGRD</td>
<td>0 1061</td>
<td>10</td>
<td>5.000</td>
<td>4.000</td>
<td></td>
</tr>
<tr>
<td>22800095</td>
<td>UGRD</td>
<td>0 1061</td>
<td>10</td>
<td>5.000</td>
<td>3.700</td>
<td></td>
</tr>
<tr>
<td>2280307</td>
<td>PBAC</td>
<td>0 1061</td>
<td>50</td>
<td>7.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>2280308</td>
<td>PBAC</td>
<td>0 1061</td>
<td>50</td>
<td>11.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>2280315</td>
<td>UGRD</td>
<td>0 1061</td>
<td>10</td>
<td>5.000</td>
<td>4.000</td>
<td></td>
</tr>
<tr>
<td>2280319</td>
<td>UGRD</td>
<td>0 1061</td>
<td>10</td>
<td>5.000</td>
<td>4.000</td>
<td></td>
</tr>
<tr>
<td>2280427</td>
<td>PBAC</td>
<td>0 1061</td>
<td>50</td>
<td>4.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>2280514</td>
<td>UGRD</td>
<td>0 1061</td>
<td>10</td>
<td>10.000</td>
<td>4.000</td>
<td></td>
</tr>
<tr>
<td>2280668</td>
<td>PBAC</td>
<td>0 1061</td>
<td>50</td>
<td>3.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>2281056</td>
<td>PBAC</td>
<td>0 1061</td>
<td>50</td>
<td>15.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>2281077</td>
<td>UGRD</td>
<td>0 1061</td>
<td>10</td>
<td>4.000</td>
<td>4.000</td>
<td></td>
</tr>
<tr>
<td>2281279</td>
<td>UGRD</td>
<td>0 1061</td>
<td>10</td>
<td>4.000</td>
<td>4.000</td>
<td></td>
</tr>
<tr>
<td>2281285</td>
<td>UGRD</td>
<td>0 1061</td>
<td>10</td>
<td>4.000</td>
<td>4.000</td>
<td></td>
</tr>
<tr>
<td>2281424</td>
<td>PBAC</td>
<td>0 1061</td>
<td>50</td>
<td>3.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>2281525</td>
<td>PBAC</td>
<td>0 1061</td>
<td>50</td>
<td>4.000</td>
<td>0.000</td>
<td></td>
</tr>
</tbody>
</table>

1
AND A.STRM = '1061'
AND A.EMPLID > '22800000'
AND A.EMPLID = CASE WHEN A.ACAD_CAREER = 'UGRD'
AND A.CUM_GPA > 3.6
THEN A.EMPLID
WHEN A.ACAD_CAREER = 'PBAC' THEN A.EMPLID
END
Exploring Further - Analytic Criteria?

• So, we’ve explored using CASE as criteria.

• We’ve unlocked full access to the WHERE and the ORDER BY clauses by using 1=1.

  • Is that enough?

CAN I USE ANALYTIC FUNCTIONS AS CRITERIA?

• All joins and all WHERE, GROUP BY, and HAVING clauses are completed before the analytic functions are processed.
• Therefore, analytic functions can appear only in the SELECT list or ORDER BY clause - not the WHERE clause.

NO?
SELECT DISTINCT A.EMPLID, A.FIRST_NAME, A.LAST_NAME, C.ADDRESS1, C.ADDRESS2, C.ADDRESS3, C.ADDRESS4, C.CITY, C.STATE, CASE WHEN (C.COUNTRY = 'USA' AND LENGTH(TRIM(C.POSTAL)) = 9) THEN SUBSTR(C.POSTAL,1,5) || '-' || SUBSTR(C.POSTAL,6,4) ELSE TRIM(C.POSTAL) END, C.COUNTRY
FROM PS_PERSONAL_DATA A, PS_ADDR_USAGE_VW C
WHERE A.EMPLID IN
  (SELECT CASE WHEN (PERCENT_RANK () OVER (PARTITION BY B.ACAD_LEVEL_BOT ORDER BY B.CUM_GPA DESC)) * 100 <= 2 THEN B.EMPLID END
   FROM PS_STDNT_CAR_TERM B
   WHERE B.STRM = :1
   AND B.ACAD_LEVEL_BOT IN ('30','40')
   AND B.UNT_TAKEN_PRGRSS > 0
   AND B.ELIG_TO_ENROLL = 'Y')
   AND A.FERPA <> 'Y'
   AND A.EMPLID = C.EMPLID
AND C.ADDR_USAGE = 'DMH'
AND C.ADDR_USAGE_ORDER = (SELECT MIN( D.ADDR_USAGE_ORDER)
                                  FROM PS_ADDR_USAGE_VW D
                                  WHERE D.EMPLID = A.EMPLID
                                  AND D.ADDR_USAGE = 'DMH')
Exploring Further - HINTS

• Oracle SQL Hints
  – What are they, when and why should I use them?
  – Cost Based Optimizer (CBO)
  – Control your query’s own fate...

• Hints are valuable commands that sometimes can be used to help your queries execute more effectively and efficiently.
Exploring Further - HINTS

• What is a cost based optimizer?
  – An Oracle built-in component that uses data statistics to identify the query plan with the lowest cost on system resources, in turn, designing an execution plan for the SQL statement.
  – The CBO’s sole purpose is to optimize the query’s execution. When it is working at its best, no hints should be required.
  – All this is contingent on your data structure.

Unfortunately, sometimes the data in the database changes (oh so frequently) that the statistical information previously gathered by the optimizer is out of date.
Exploring Further - HINTS

That’s where Hints come in...they allow you to make decisions usually made by the optimizer.

Alas, not everything is definite. The caveat to this is when the optimizer is set to lock the statistics when ideally configured.
Exploring Further - HI NTS

There are many different types of hints, which are categorized as follows:

- Optimization Approaches and Goals
- Access Paths and Query Transformations
- Join Orders
- Join Operations
- Parallel Execution

...and several others...
Exploring Further - HINTS

DI SCLAI MER: The majority of these Hints require direct access to write, create or modify sql, so hopefully you have a great working relationship and rapport with your technical personnel.

With that said, let’s focus on a couple hints that you CAN use directly within the PS Query Tool...
Exploring Further - HI NTS

• **All_ROWS** - chooses the cost-based approach to optimize minimum total resource consumption
  - Results returned ONLY after all processing has been completed

  
  /*+ ALL_ROWS */

• **FIRST_ROWS(n)** - chooses the approach to optimize minimum resource usage (response time) to return the first row.
  - Results returned as soon as they are identified

  
  /*+ FIRST_ROWS(n) */
DISCLAIMER: The CBO ignores the FIRST_ROWS hint in SELECT statements that contain any of the following syntax:

- GROUP BY clause
- Group functions
- Use of Distinct
- Set operators
- Union
- Intersect
These statements cannot be optimized for best response time because all rows accessed by the statement must first be retrieved before returning the first row. Although, if the hint is used, the query will still be optimized, but for best minimum resource consumption.

- **CHOOSE** - chooses between ALL_ROWS or FIRST_ROWS based on statistics gathered
  - Statistics available = ALL_ROWS
  - Statistics unavailable = FIRST_ROWS

/*+ CHOOSE_ROWS */
Questions?
Hands-On Problem Solving

• Audience
• Practice Problems
• Ideas
• Brainstorming
TIPS - Running Total

Generate a Running Total:

```
SELECT A.EMPLID, A.STRM, A.UNT_TAKEN_PRGRSS,
SUM(A.UNT_TAKEN_PRGRSS) OVER (PARTITION BY A.EMPLID
ORDER BY A.STRM, A.EMPLID),
SUM(A.UNT_TAKEN_PRGRSS) OVER (ORDER BY A.STRM, A.EMPLID)
FROM PS_STDNT_CAR_TERM A
WHERE ROWNUM <= '25'
```
TI PS - Numbers to Words (p1)

• Converting Numbers to Words
  – TO_CHAR(TO_DATE(TO_CHAR(A.ACAD_YEAR,'99999999999'),'J'),'J SP')

  • Let’s examine each component function:
  • The inner TO_CHAR converts the number (which would generally be a numeric variable) to CHAR, so the built-in processes can do their work
  • The TO_DATE converts the CHAR using the J (Julian day) format. The Julian day is the number of days since January 1, 4712 BC.
  • Having established the date value, we then convert that date back to a Julian day. Because the TO_CHAR is used in DATE context, we can use the J mask to duplicate the original value, and append the SP (spelling) format mask. 'SP' does exactly that - it converts the number to words, hence the string value above.
## TIPS - Numbers to Words (p2)

<table>
<thead>
<tr>
<th>STUDENT NAME</th>
<th>GRAD YEAR</th>
<th>CLASS OF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mickey</td>
<td>2005</td>
<td>TWO THOUSAND FIVE</td>
</tr>
<tr>
<td>Minnie</td>
<td>1998</td>
<td>ONE THOUSAND NINETEEN HUNDRED NETY-EIGHT</td>
</tr>
<tr>
<td>Goofy</td>
<td>2000</td>
<td>TWO THOUSAND</td>
</tr>
<tr>
<td>Donald</td>
<td>TBD</td>
<td>TO BE DETERMINED</td>
</tr>
</tbody>
</table>
### TIPS - Amounts to Words

**Converting Amounts to Words**

- SELECT 'MICKEY MOUSE', A.LINE_AMT, DECODE(FLOOR(A.LINE_AMT),0,'ZERO',TO_CHAR(TO_DATE(FLOOR(A.LINE_AMT),'J'),'JSP'))||' DOLLARS AND '|| DECODE(MOD( A.LINE_AMT*100,100),0,'ZERO', TO_CHAR(TO_DATE(MOD(A.LINE_AMT*100,100),'J'),'JSP'))||' CENTS' FROM PS_ITEM_LINE_SF A WHERE A.EMPL_NAME = 'Mickey Mouse'

<table>
<thead>
<tr>
<th>Fields</th>
<th>Criteria</th>
<th>SQL</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee Name</td>
<td>Line Amt</td>
<td>In The Amount of</td>
<td></td>
</tr>
<tr>
<td>MICKEY MOUSE</td>
<td>998.00</td>
<td>NINE HUNDRED NINETY-EIGHT DOLLARS AND ZERO CENTS</td>
<td></td>
</tr>
<tr>
<td>MICKEY MOUSE</td>
<td>1100.00</td>
<td>ONE THOUSAND ONE HUNDRED DOLLARS AND ZERO CENTS</td>
<td></td>
</tr>
<tr>
<td>MICKEY MOUSE</td>
<td>1248.00</td>
<td>ONE THOUSAND TWO HUNDRED FORTY-EIGHT DOLLARS AND ZERO CENTS</td>
<td></td>
</tr>
<tr>
<td>MICKEY MOUSE</td>
<td>844.00</td>
<td>EIGHT HUNDRED FORTY-FOUR DOLLARS AND ZERO CENTS</td>
<td></td>
</tr>
<tr>
<td>MICKEY MOUSE</td>
<td>719.00</td>
<td>SEVEN HUNDRED NINETEEN DOLLARS AND ZERO CENTS</td>
<td></td>
</tr>
<tr>
<td>MICKEY MOUSE</td>
<td>352.77</td>
<td>THREE HUNDRED FIFTY-TWO DOLLARS AND SEVENTY-SEVEN CENTS</td>
<td></td>
</tr>
</tbody>
</table>
Questions?
Review

- Functions are SQL commands.
- The Three Main Categories are:
  - Aggregate
  - Single-row
  - Analytic
- Functionality of Functions:
  - Numeric
  - String/Character
  - Conversion
  - Date and Time
  - If-Then Logic
  - Analytic Grouping
- Be Methodical in your Methodology.
- Get Familiar with SQL.
Conclusion

• Reviewed both common function statements and complex expressions.

• Explored the many possibilities of using function statements to provide greater flexibility, functionality and power to queries.

• Discovered creative ways to overcome many of the limitations of the PS Query Tool for improved reporting use.
Resources

Harvard - Key Functions in Oracle SQL
http://vpf-web.harvard.edu/applications/ad_hoc/key_functions_in_oracle_sql.pdf

Oracle 9i SQL Reference
Web:
http://www.cs.ncl.ac.uk/teaching/facilities/swdoc/oracle9i/server.920/a96540/toc.htm
Pdf:
http://www.cs.utah.edu/classes/cs6530/oracle/doc/B10501_01/server.920/a96540.pdf

ORACLE 10g SQL Reference
web:
http://download-west.oracle.com/docs/cd/B19306_01/server.102/b14200/toc.htm
pdf:
http://download-west.oracle.com/docs/cd/B19306_01/server.102/b14200.pdf

GridinSoft Notepad Lite:

HEUG 2006 Power Expressions Presentation:
http://www.heug.org/index.php?mo=do&op=sd&sid=4228&type=0
Questions?
Contacts

Uriel Hernandez
Information Technology Applications Specialist
Project Management & Information Technology Department
Central Washington University
E-mail: hernandu@cwu.edu

Tim McGuire
Information Technology Applications Specialist
Project Management & Information Technology Department
Central Washington University
E-mail: mcguiret@cwu.edu
This presentation and all Alliance 2007 presentations are available for download from the Conference Site.

Presentations from previous meetings are also available.