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

ALLIANCE'07

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 10*g*
- 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('MINNIE 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 - *LENGTH*

- **LENGTH – returns the number of a characters in a string or field.**
  - LENGTH(char)
    - **It returns a Number.**
    - **It counts all characters including trailing blanks.**
  - **LENGTH('Mickey Mouse') = 12**
  - **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)

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

```
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]...)**

*(The first expr will determine the data type that is **returned**.)*

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 is 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.**

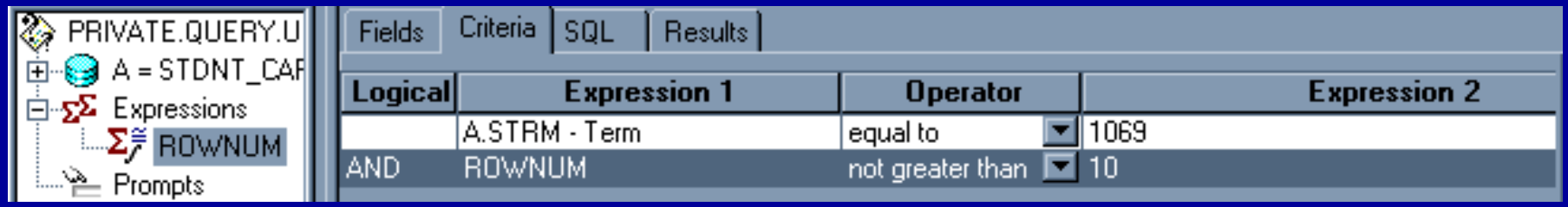
ROWNUM	ID	Career	Career Nbr	Term	Prim Prog	Take Prgrs	GPA	GPA
1	22393	UGRD	0	1069	UG	14.000	2.964	2.964
2	22519	UGRD	1	1069	UG	15.000	3.100	3.100
3	22647	UGRD	0	1069	INTL	0.000	0.000	0.000
4	22752	UGRD	0	1069	NM	0.000	0.000	0.000
5	22771	UGRD	0	1069	INTL	16.000	1.250	2.250
6	22771	UGRD	0	1069	INTL	16.000	3.650	3.633
7	22771	UGRD	0	1069	NM	13.000	3.000	3.391
8	22694	UGRD	0	1069	NM	0.000	0.000	0.000
9	22785	UGRD	0	1069	INTL	16.000	3.325	3.408
10	22471	UGRD	0	1069	UG	0.000	0.000	3.973

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 then 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**

Advanced Functions – *CASE* (p1)

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.

Advanced Functions – *CASE* (p2)

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* (p3)

CASE, LENGTH, SUBSTR, | |, TRIM

Zip Code Plus 4

```
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
```

Before	After
989267405	98926-7405
98020	98020
98948-3722	98948-3722

Advanced Functions – *CASE* (p4)

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

Nested

```
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
```

Advanced Functions – *CASE* (p5)

```
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
```

Advanced Functions – *CASE* (p6)

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

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)

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

Rows Fetched = 2280

- 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.

Analytic Functions – *Change Group*

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

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

ID	Career	Strt Level	Take Prgrs	Term	Count per Term	Count per Grade
2261	UGRD	30	0.000	1051	540	907
2261	UGRD	30	9.000	1053	507	907
2261	UGRD	30	4.000	1056	479	907
2261	UGRD	30	5.000	1059	754	907
2378	UGRD	10	9.000	1051	540	563
2378	UGRD	10	0.000	1053	507	563
2378	UGRD	10	0.000	1056	479	563
2378	UGRD	10	0.000	1059	754	563
2443	UGRD	10	15.000	1059	754	563
2612	UGRD	30	13.000	1051	540	907
2612	UGRD	30	13.000	1053	507	907
2612	UGRD	30	12.000	1056	479	907
2612	UGRD	40	13.000	1059	754	299
2690	UGRD	10	15.000	1059	754	563

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)

ID	Career	Strt Level	Take Prgrs	Term	Count per Term	Count per Grade	Distinct Count per Grade
2261	UGRD	30	0.000	1051	540	907	335
2261	UGRD	30	9.000	1053	507	907	335
2261	UGRD	30	4.000	1056	479	907	335
2261	UGRD	30	5.000	1059	754	907	335
2378	UGRD	10	9.000	1051	540	563	411
2378	UGRD	10	0.000	1053	507	563	411
2378	UGRD	10	0.000	1056	479	563	411
2378	UGRD	10	0.000	1059	754	563	411
2443	UGRD	10	15.000	1059	754	563	411
2612	UGRD	30	13.000	1051	540	907	335
2612	UGRD	30	13.000	1053	507	907	335
2612	UGRD	30	12.000	1056	479	907	335
2612	UGRD	40	13.000	1059	754	299	130
2690	UGRD	10	15.000	1059	754	563	411

Rows Fetched = 2280

Analytic Functions – *Multiple Groups*

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

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

COUNT (DISTINCT A.EMPLID) OVER
(PARTITION BY A.ACAD_LEVEL_BOT, **A.STRM**)

ID	Career	Strt Level	Take Prgrs	Term	Count per Term	Count per Grade	Distinct Count per Grade	Distinct # in Grade per Term
2261	UGRD	30	0.000	1051	540	907	335	245
2261	UGRD	30	9.000	1053	507	907	335	237
2261	UGRD	30	4.000	1056	479	907	335	203
2261	UGRD	30	5.000	1059	754	907	335	222
2378	UGRD	10	9.000	1051	540	563	411	129
2378	UGRD	10	0.000	1053	507	563	411	100
2378	UGRD	10	0.000	1056	479	563	411	55
2378	UGRD	10	0.000	1059	754	563	411	279
2443	UGRD	10	15.000	1059	754	563	411	279
2612	UGRD	30	13.000	1051	540	907	335	245
2612	UGRD	30	13.000	1053	507	907	335	237
2612	UGRD	30	12.000	1056	479	907	335	203
2612	UGRD	40	13.000	1059	754	299	130	119
2690	UGRD	10	15.000	1059	754	563	411	279

Rows Fetched = 2280

Analytic Functions – *SUM*

Total Credits by ID

SUM (A.UNT_TAKEN_PRGRSS) OVER (PARTITION BY A.EMPLID)

ID	Career	Strt Level	Take Prgrs	Term	Total Units per ID
2261	UGRD	30	0.000	1051	18.000
2261	UGRD	30	9.000	1053	18.000
2261	UGRD	30	4.000	1056	18.000
2261	UGRD	30	5.000	1059	18.000
2378	UGRD	10	9.000	1051	9.000
2378	UGRD	10	0.000	1053	9.000
2378	UGRD	10	0.000	1056	9.000
2378	UGRD	10	0.000	1059	9.000
2443	UGRD	10	15.000	1059	15.000
2612	UGRD	30	13.000	1051	51.000
2612	UGRD	30	13.000	1053	51.000
2612	UGRD	30	12.000	1056	51.000
2612	UGRD	40	13.000	1059	51.000
2690	UGRD	10	15.000	1059	15.000
2768	UGRD	30	15.000	1051	68.000

Rows Fetched = 2280

Analytic Functions – *Query Tip #2*

Group by a Constant

COUNT (A.EMPLID) OVER ()

COUNT (DISTINCT A.EMPLID) OVER ()

ID	Career	Strt Level	Take Prgrs	Term	Total Rows in Query	Distinct ID Count
2261	UGRD	30	0.000	1051	2280	943
2261	UGRD	30	9.000	1053	2280	943
2261	UGRD	30	4.000	1056	2280	943
2261	UGRD	30	5.000	1059	2280	943
2378	UGRD	10	9.000	1051	2280	943
2378	UGRD	10	0.000	1053	2280	943
2378	UGRD	10	0.000	1056	2280	943
2378	UGRD	10	0.000	1059	2280	943
2443	UGRD	10	15.000	1059	2280	943
2612	UGRD	30	13.000	1051	2280	943
2612	UGRD	30	13.000	1053	2280	943
2612	UGRD	30	12.000	1056	2280	943
2612	UGRD	40	13.000	1059	2280	943
2690	UGRD	10	15.000	1059	2280	943
2768	UGRD	30	15.000	1051	2280	943

Rows Fetched = 2280

Analytic Functions – *Query Tip #3*

Count Multiple ID's

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

ID	Career	Strt Level	Take Prgrs	Term	Distinct ID Count	# of Rows per ID
2261	UGRD	30	0.000	1051	943	4
2261	UGRD	30	9.000	1053	943	4
2261	UGRD	30	4.000	1056	943	4
2261	UGRD	30	5.000	1059	943	4
2378	UGRD	10	9.000	1051	943	4
2378	UGRD	10	0.000	1053	943	4
2378	UGRD	10	0.000	1056	943	4
2378	UGRD	10	0.000	1059	943	4
2443	UGRD	10	15.000	1059	943	1
2612	UGRD	30	13.000	1051	943	4
2612	UGRD	30	13.000	1053	943	4
2612	UGRD	30	12.000	1056	943	4
2612	UGRD	40	13.000	1059	943	4
2690	UGRD	10	15.000	1059	943	1
2768	UGRD	30	15.000	1051	943	4

Rows Fetched = 2280

Analytic Functions – *ORDER BY*

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

ID	Career	Strt Level	GPA	Percent Rank In Class	% Rank in Class
7601	UGRD	10	.927	.974	97.4
6317	UGRD	10	.836	.978	97.8
2983	UGRD	10	.617	.983	98.3
2539	UGRD	10	.591	.987	98.7
4783	UGRD	10	.425	.991	99.1
8545	UGRD	10	.206	.995	99.5
7218	UGRD	10	.133	1	100.0
8087	UGRD	20	4.000	0	0.0
121	UGRD	20	4.000	0	0.0
8227	UGRD	20	4.000	0	0.0
7578	UGRD	20	3.759	.058	5.8
6408	UGRD	20	3.684	.078	7.8
4582	UGRD	20	3.683	.098	9.8
6193	UGRD	20	3.667	.117	11.7
6356	UGRD	20	3.628	.137	13.7

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

- **RANK - calculates the rank of a value in a group of values.**
  - RANK( ) OVER ([query_partition_clause] order_by_clause)
  - **Returns the rank as a NUMBER.**
  - **RANK computes the rank of each row returned from a query with respect to the other rows returned in 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 – *LAG / LEAD*

- **LAG | 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, (LAG|LEAD) provides access to a row at a given physical offset (prior|beyond) that position.

  - LAG(value_expr [, 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 – *LAG / LEAD* (p2)

Compare Address Changes

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

```
LAG(A.ADDRESS1, 1, 'NO CHANGE' ) OVER
    (PARTITION BY A.EMPLID ORDER BY A.EFFDT)
```

Analytic Functions - *NTILE*

- **NTILE - divides an ordered data set into the number of buckets as indicated and assigns the appropriate bucket number to each row.**
  - NTILE(expr) OVER ([query_partition_clause] order_by_clause)
  - **Used to evenly distribute a group into subgroups.**
  - **The return value is a NUMBER.**
  - **The number of rows in the buckets can differ by at most 1.**
  - **The remainder values are distributed one for each bucket, starting with bucket 1.**
    - **NTILE(6) OVER (ORDER BY A.LAST_NAME)**

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:

	ID	Career	Term	Strt Level	Take Prgrs	Most Units Taken
1	AA0014	UGRD	0350	20	16.000	1
2	AA0005	UGRD	0350	20	13.000	2
3	AA0003	UGRD	0350	20	10.000	3
4	AA0009	UGRD	0350	20	9.500	4
5	AA0006	UGRD	0350	20	9.000	5
6	AA0001	UGRD	0350	20	9.000	6
7	AA0002	UGRD	0350	10	9.000	7
8	AA0004	UGRD	0350	10	6.000	8
9	AA0007	UGRD	0350	10	6.000	9
10	AA0008	UGRD	0350	20	6.000	10

```
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 - *RATIO_TO_REPORT*

- **RATIO_TO_REPORT - calculates the ratio of a value to the sum of a set of values**
  - **ratio_to_report(expr) over ([query partition clause]) *if expr* is null, then 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:**
      - *Select employee_name, hours, ratio_to_report(hours) over ()*

EMPLOYEE	HOURS	RATIO_TO_REPORT
Mickey	20	0.166666667
Minnie	50	0.416666667
Goofy	10	0.083333333
Donald	40	0.333333333

Questions?

THE POWER COMBO

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

```
SUM (B.UNT_PRGRSS) OVER (PARTITION BY A.EMPLID)
```

Total Credits per Person as of Date

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

Power Combo – Example Continued

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'
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
```

```
CASE WHEN 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(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
```

Methodology - Function Statements (p3)

11. Query Tip Review:

#1: Limit Number of Rows Returned

```
ROWNUM <= 100
```

#2: Unique Count

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

#3: Multiple Rows Count

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

Methodology - Function Statements (p4)

12. Using/Viewing SQL

What's REALLY going on?

Fields | Criteria | SQL | Results

Col	Record.Field	Format	Re	Ord	Xlt	Agg	Heading
1	A.ACAD_CAREER - Academic Career	Char4	X				Career
2	A.STRM - Term	Char4	X				Term
3	A.TERM_BEGIN_DT - Term Begin Date	Date					Begin Date
4	NEXT_DAY(SYSDATE,'TUESDAY')	Date					Next Tuesday

Fields | Criteria | SQL | Results

```
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

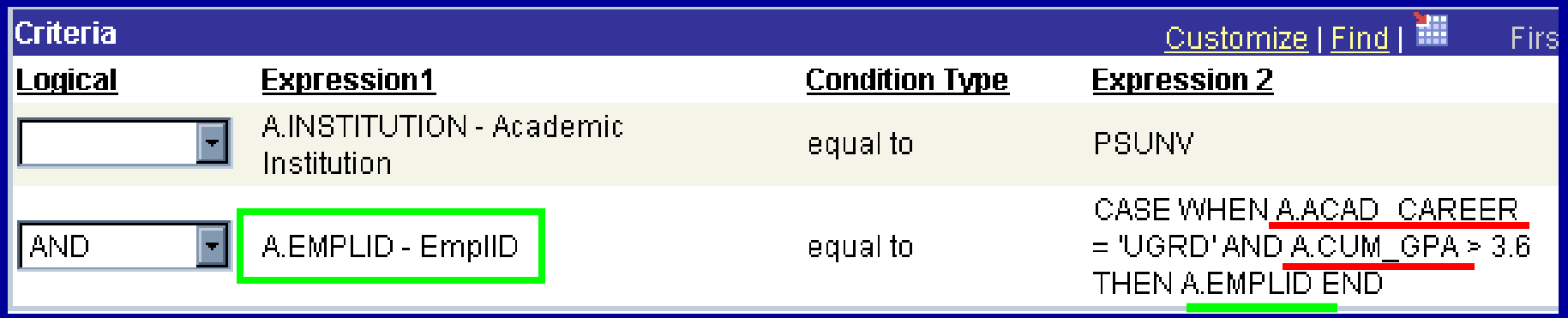
FROM

WHERE = Determines the Rows by Criteria

ORDER BY = Organizes Final Order

Exploring Further – WHERE Clause

Using Function Expressions as Criteria:




```
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!

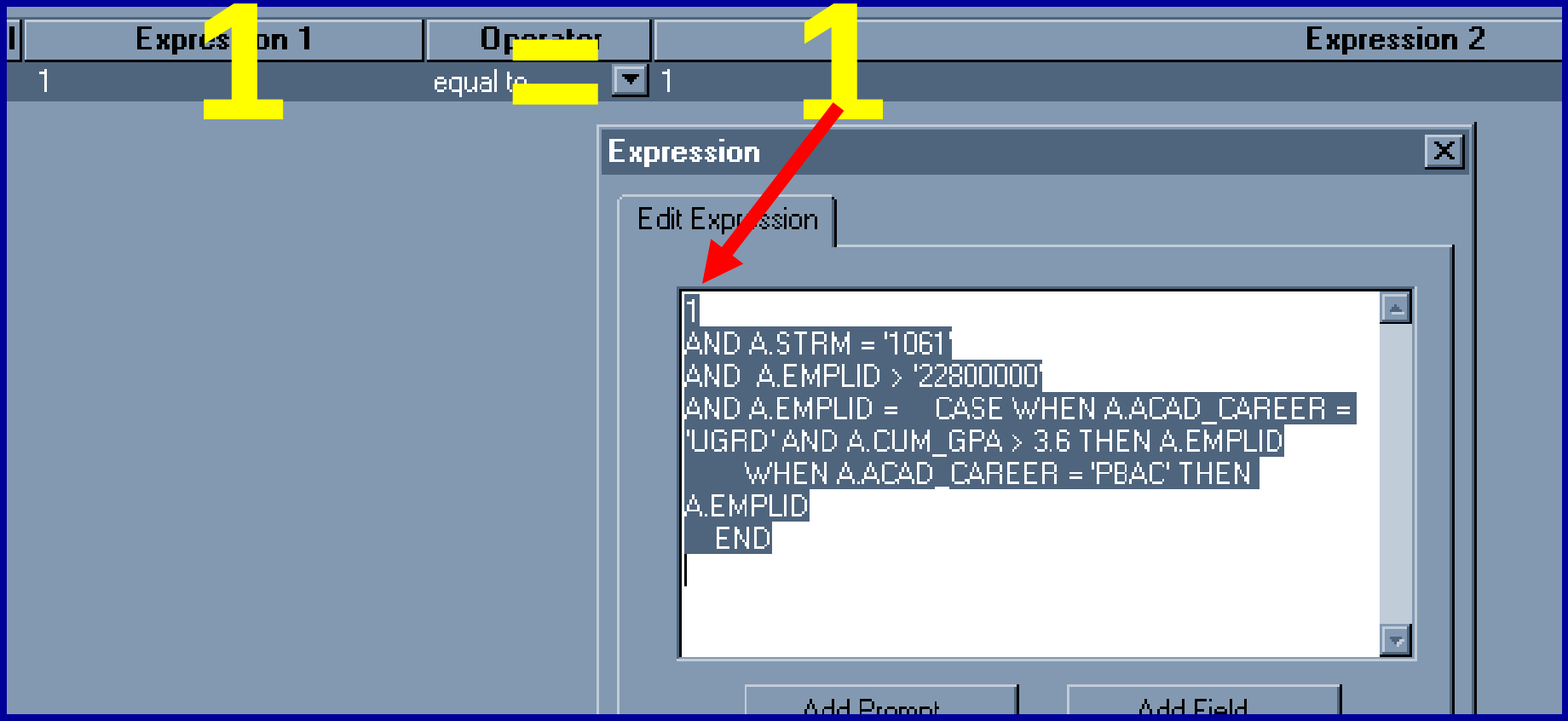
The Criteria simply states 1 = 1, but the SQL states:

HOW?

```
SELECT A.EMPLID, A.ACAD_CAREER, A.STDNT_CAR_NBR, A.STRM, A.ACAD_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)

Fields | Criteria | SQL | Results

ID	Career	Career Nbr	Term	Strt Level	Take Prgrs	GPA
2280091	UGRD	0	1061	10	5.000	4.000
2280095	UGRD	0	1061	10	5.000	3.700
2280307	PBAC	0	1061	50	7.000	0.000
2280308	PBAC	0	1061	50	11.000	0.000
2280315	UGRD	0	1061	10	5.000	4.000
2280319	UGRD	0	1061	10	5.000	4.000
2280427	PBAC	0	1061	50	4.000	0.000
2280514	UGRD	0	1061	10	10.000	4.000
2280668	PBAC	0	1061	50	3.000	0.000
2281056	PBAC	0	1061	50	15.000	0.000
2281077	UGRD	0	1061	10	4.000	4.000
2281279	UGRD	0	1061	10	4.000	4.000
2281285	UGRD	0	1061	10	4.000	4.000
2281424	PBAC	0	1061	50	3.000	0.000

```
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?

Exploring Further – Analytic Subquery

Description: Top 35% of Jr,Sr in Trm

Term: 1069

Top Percent (~ 35): 35

View Results

Criteria

Logical	Expression1	Condition Type	Expression 2
	A.EMPLID - ID	in list	SUBQUERY

Top Level of Query
- Subquery for A.EMPLID - ID
- Subquery for C.ADDR_USAGE_ORDE

Expressions List

	Expression	Use as Field	Add
1	CASE WHEN (PERCENT_RANK () OVER (PARTITION BY B.ACAD_LEVEL_BOT ORDER BY B.CUM_GPA DESC)) * 100 <= :2 THEN B.EMPLID END	Use as Field	

Fields View All

Col	Record.Fieldname	Format	Ord	XLAT	Agg	Heading Text	A
1	CASE WHEN (PERCENT_RANK () OVER (PARTITION BY B.ACAD_LEVEL_BOT ORDER BY B.CUM_GPA DESC)) * 100 <= :	Char11				CASE WHEN (PERCENT_RANK () OV	

Exploring Further – Analytic Subquery (p2)

```
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 it's 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 - HINTS

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

DISCLAIMER: 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 - HINTS

- 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) */
```

Exploring Further - HINTS

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

Exploring Further - HINTS

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:

	ID	Term	Take Prgrs	Student Total	Overall Total
1	10222200	0893	6.000	6	6
2	10222200	0896	9.000	15	15
3	10127040	0931	8.000	8	23
4	10127040	0933	8.000	16	31
5	10127040	0943	8.000	24	39
6	10127040	0946	8.000	32	47
7	10127040	0949	8.000	40	55
8	10127040	0951	8.000	48	63
9	10127040	0953	8.000	56	71
10	10096334	0979	12.000	12	83
11	10100234	0979	16.000	16	99

```
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'
```

TIPS - Numbers to Words (p1)

- Converting Numbers to Words
  - TO_CHAR(TO_DATE(TO_CHAR(A.ACAD_YEAR,'999999999 99'),'J'),'JSP')
    - 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, 4712BC.
    - 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)

STUDENT NAME	GRAD YEAR	CLASS OF
Mickey	2005	TWO THOUSAND FIVE
Minnie	1998	ONE THOUSAND NINETEEN HUNDRED NINETY-EIGHT
Goofy	2000	TWO THOUSAND
Donald	TBD	TO BE DETERMINED

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'

Fields | Criteria | SQL | Results

Employee Name	Line Amt	In The Amount of
MICKEY MOUSE	998.00	NINE HUNDRED NINETY-EIGHT DOLLARS AND ZERO CENTS
MICKEY MOUSE	1100.00	ONE THOUSAND ONE HUNDRED DOLLARS AND ZERO CENTS
MICKEY MOUSE	1248.00	ONE THOUSAND TWO HUNDRED FORTY-EIGHT DOLLARS AND ZERO CENTS
MICKEY MOUSE	844.00	EIGHT HUNDRED FORTY-FOUR DOLLARS AND ZERO CENTS
MICKEY MOUSE	719.00	SEVEN HUNDRED NINETEEN DOLLARS AND ZERO CENTS
MICKEY MOUSE	352.77	THREE HUNDRED FIFTY-TWO DOLLARS AND SEVENTY-SEVEN CENTS

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:
http://www.gridinsoft.com/downloads.php

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