

RMS Report Designing

The image displays two overlapping software windows. The top window is the 'Resident Management System' showing a list of contracts for the Los Angeles District. The bottom window is the 'RMS Report Designer' showing a report design for 'Interest Payments' based on the contract data.

Resident Management System - Contracts - Los Angeles District

Contract Office	Contract ID/No	Contract Title
CESPL-CO-AH	DACA05-02-C-0010 NA	FY02 Replacement Army Family
CESPL-CO-AH	DACA09-01-C-0007 NA	Building 62703 - GIB Addition
CESPL-CO-AH	DACA09-01-D-0014 0007	Naco Security System
CESPL-CO-AH	DACA09-01-D-0023 0003	Border Patrol Station Paving
CESPL-CO-AH	DACA09-01-D-0023 0004	King's Ranch Road
CESPL-CO-AH	DACA09-01-D-0023 0005	Construct New Parkin
CESPL-CO-AH	DACA09-02-C-0007 NA	Effluent Reuse Syst

RMS Report Designer [spl_rms.world]: Interest Payments

Report Design: Interest Payments

Location	Current Contract	Invoice	Check No	Check Date	Days Late	Interest Amount
Header						
FMT_CONTRACT_DELIVERY PROJ NAME						
Group Header(0) FMT_CONTRACT_DELIVERY						
LOCATION	URRENT CONTRACT	INV NO	CHECK NUMBE	CHECK DATE	YS LATE	INTEREST AMT
Detail						
Total Interest for FMT_CONTRACT_DELIVER						Sum(INTEREST AMT)
Group Footer(0) FMT_CONTRACT_DELIVERY						
Total Interest for ALL Contracts						Sum(INTEREST AMT)
Footer						
Summary						

RMS Report Writing

Examples for designing custom report in RMS

by RMS Support Center

RMS uses the Report Builder report writing tool to allow users to design customized Reports using RMS data. This banded report writing tool is flexible and powerful, but requires patience and skill.

This document contains many examples of reports that illustrate interesting and powerful ways to access and display RMS data using the report writer. Step through the example one by one in order to get the most out of this document

Table of Contents

Foreword	0
Part I RMS Query Browser	3
1 RMS Query Browser	3
2 Database Info	4
3 SQLEditor	5
Part II SQL Primer	8
1 Select Statement	8
2 Distinct Clause	11
3 Where Clause	13
4 Dates and SQL	15
5 Order By Clause	21
6 Case Statement	23
7 SQL Comparison Operators	23
8 Oracle Functions	25
Part III RAP Primer	32
1 Overview	32
2 Data Types	33
3 Operators	33
4 If Statement	35
5 Case Statement	37
6 While Statement	40
7 Procedures and Functions	41
8 Gotchas	42
Index	0

RMS Report Writing

Example Report
For Training Purposes

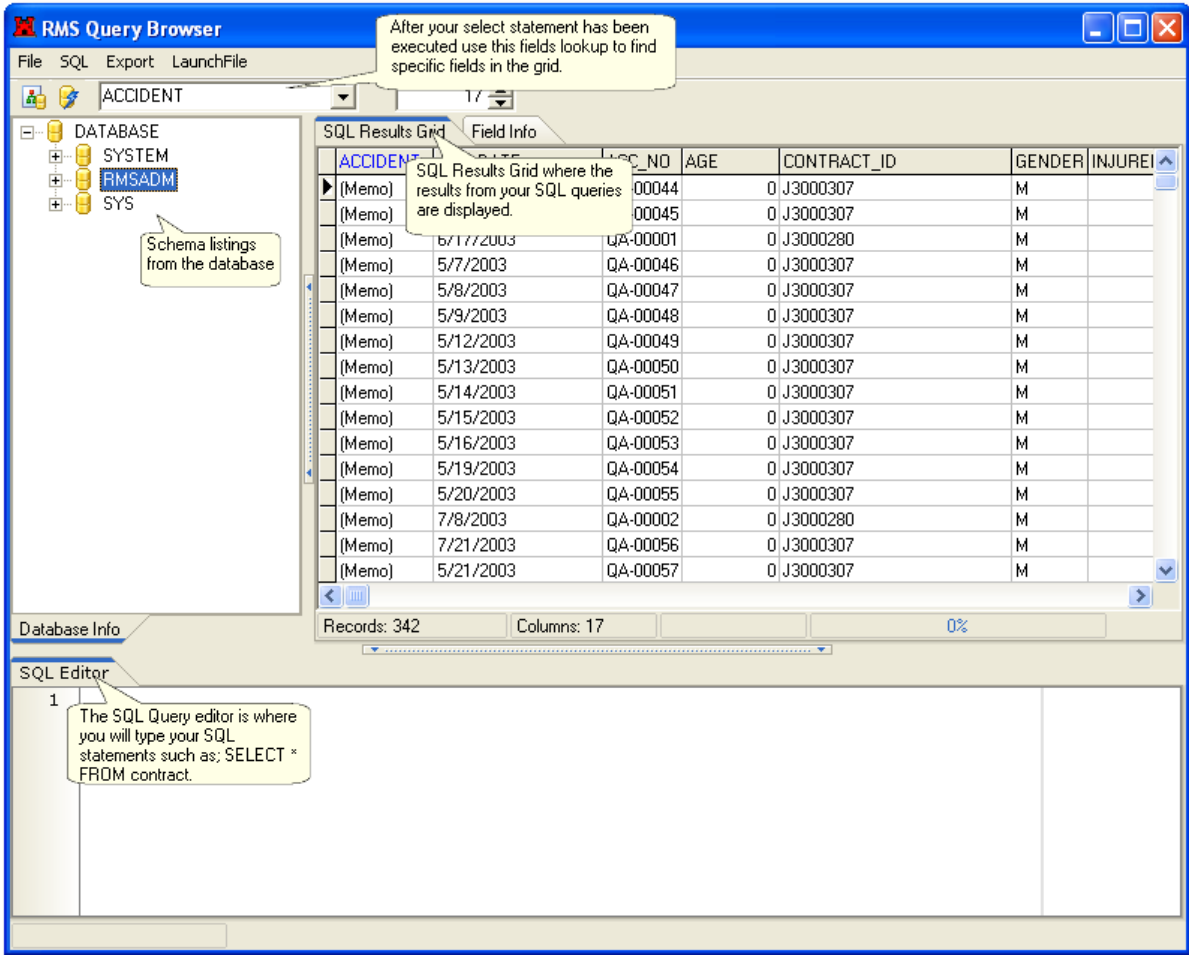
Example



1 RMS Query Browser

1.1 RMS Query Browser

The RMS Query Browser is a custom tool inside of the RMS Reports designer which will allow you to test your standard select queries for functionality. You can also use the Exporting capabilities to save you results sets out to native .xls format or to a .csv file.

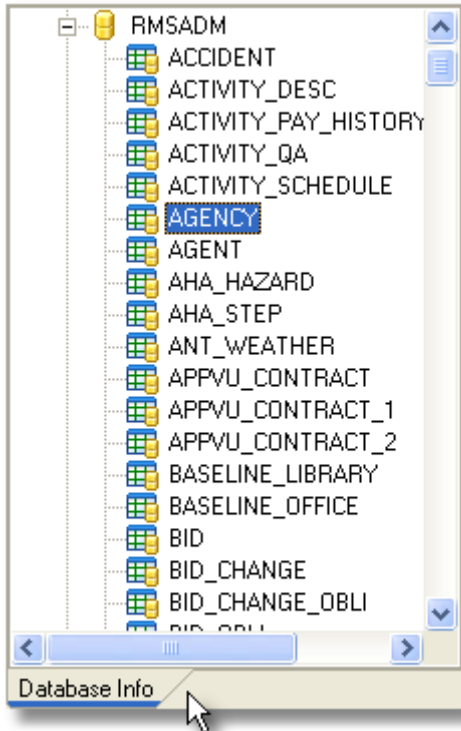


Clicking on the table names under the Schema listings will automatically select the table and display its Field Info. Use this information to create powerful select statements which you can use to enhance the quality of your reports.

Note: You will not be able to perform any type of updates or alter the database in anyway using this utility. It is strictly used for browsing and exporting data to native excel format.

1.2 Database Info

Clicking on a table name on the Database Info tab will automatically select all of the data from this table and display it in the SQL Results Grid.



The screenshot shows the 'SQL Results Grid' window with the 'AGENCY' table data displayed. The grid has three columns: FL_CODE, SHOW_RECORD, and TITLE. The data is as follows:

FL_CODE	SHOW_RECORD	TITLE
C	Y	C - Corps of Engineers (All Levels)
E	Y	E - Environmental Protection Agency (EPA)
K	Y	K - Contractor (Claim & Suggested Changes)
P	Y	P - Potential Responsible Party (PRP)
S	Y	S - State
T	Y	T - All Other (None of the Above)
U	Y	U - User (Macom or Using Service Request)

At the bottom of the window, it shows 'Records: 7' and 'Columns: 3'.

In addition you can also click on the Field Info to get more detail about the table you are viewing.

Such as the fields type and data length and whether they allow nulls or not.

ID	FIELD_NAME	DATA_LENGTH	ALLOW_NULLS	FIELD_TYPE
1	FL_CODE	1	N	VARCHAR2
2	SHOW_RECORD	1	N	VARCHAR2
3	TITLE	42	Y	VARCHAR2

1.3 SQLEditor

The SQL Editor can be used to type in your select statements.

```
1 select contract_id, proj_ds from contract
```

Pressing F5 will execute your statements and display the results in the SQL Results Grid.

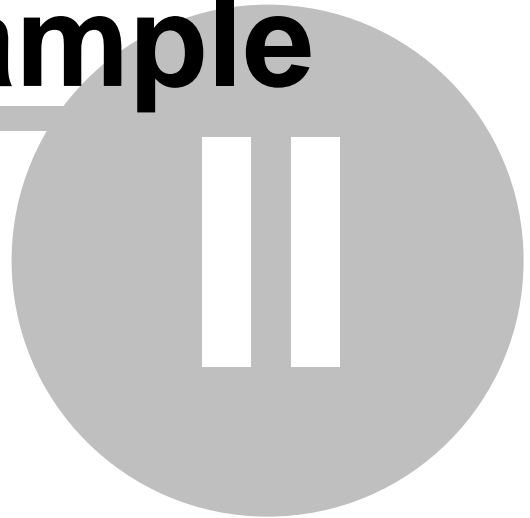
CONTRACT_ID	PROJ_DS
J3000433	Work will include general design and construction projects to include but not limited to
J3000329	FY03 MILCON KNMD023014 Installation of Flightline Security Fencing and Gates,
J3000483	
J3000484	Replace two deteriorated automatic transfer switches (ATS) on the E1 and E2 circuits
J3000089	This is a test project to test the RMS/CMS interface.
J3000111	
J3000112	
97C0058	Replace existing roofs on the 9th and 10th floors of Wings A, B, and C with new to
J3000865	
J3000863	
J3000864	
Pw99C005	Palau Compact Road Project, Babeldaob Island, Republic of Palau
99C0012	FY96 MCA PN 42470, Kolekole Connector Road, Kolekole-Trimble Connector and
J3000353	Basic Contract for Indefinite Delivery Indefinite Quantity (IDIQ) Construction and Se
DACA83-01-T-0001	
J3000899	

Records: 346 Columns: 2 0%

RMS Report Writing

Example Report
For Training Purposes

Example



2 SQL Primer

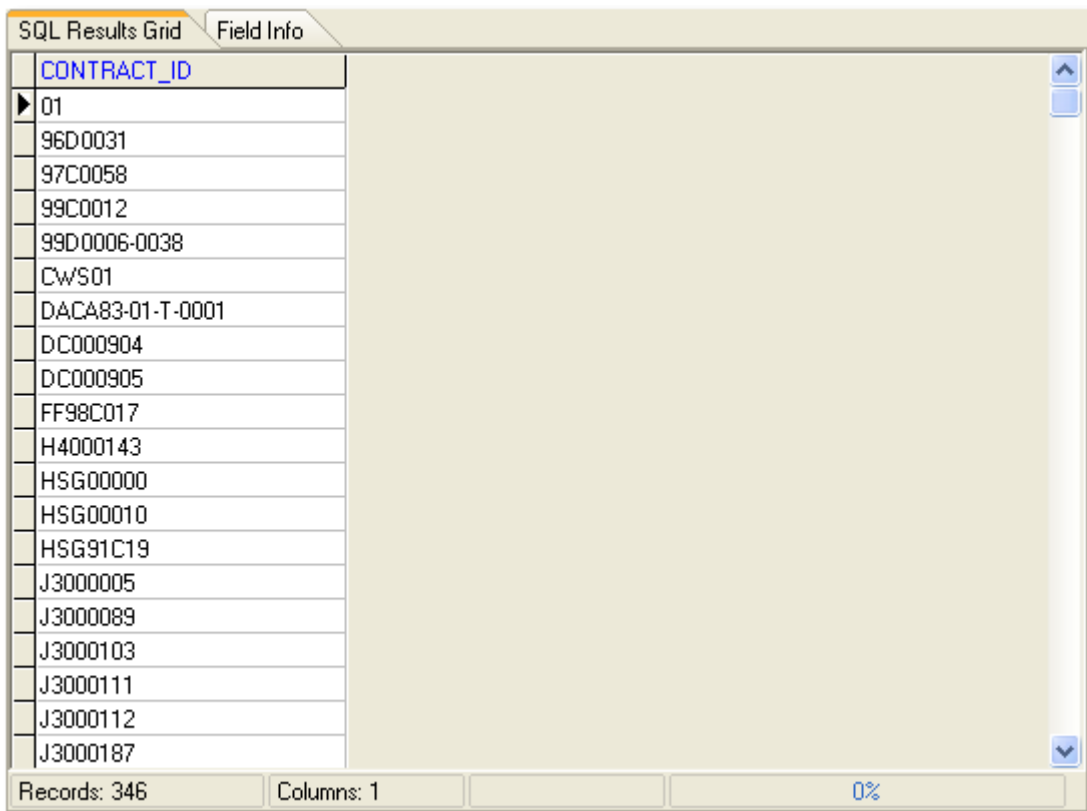
2.1 Select Statement

The select statement is used to retrieve data from the database. The format is:

select columns from tables;

Let's get a list of contract id's from the contract table.

```
1 select contract_id from contract
```



The screenshot shows a window titled "SQL Results Grid" with a "Field Info" tab. The grid displays a single column named "CONTRACT_ID" with 346 records. The first record is "01", followed by "96D0031", "97C0058", "99C0012", "99D0006-0038", "CWS01", "DACA83-01-T-0001", "DC000904", "DC000905", "FF98C017", "H4000143", "HSG00000", "HSG00010", "HSG91C19", "J3000005", "J3000089", "J3000103", "J3000111", "J3000112", and "J3000187". The status bar at the bottom indicates "Records: 346", "Columns: 1", and "0%".

CONTRACT_ID
01
96D0031
97C0058
99C0012
99D0006-0038
CWS01
DACA83-01-T-0001
DC000904
DC000905
FF98C017
H4000143
HSG00000
HSG00010
HSG91C19
J3000005
J3000089
J3000103
J3000111
J3000112
J3000187

SQL is not case sensitive. Keywords can be place in caps, but that is not a requirement. Case is important when we get to the actual data but only for the data. In other words, if I query looking for "dc000904", then "DC000904" and "Dc000904" will not be returned.

You can also alias a column using the **AS** keyword, or you can leave it out. If your new column name includes a space, you need to enclose the alias in quotes.

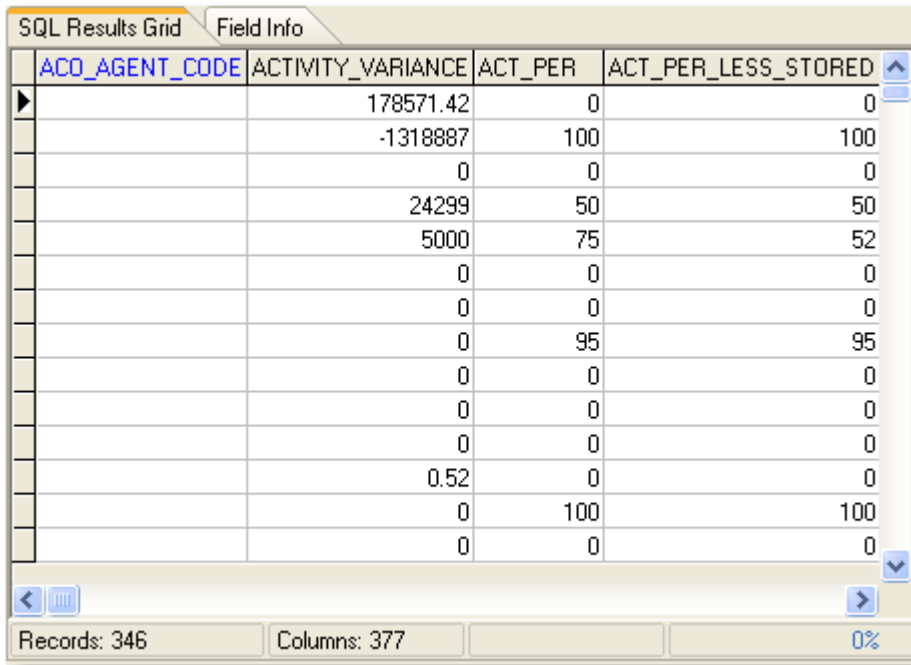
```
1 select contract_id AS "CONTRACT ID" from contract
```

```
1 select contract_id "CONTRACT ID" from contract
```

CONTRACT ID
01
96D0031
97C0058
99C0012
99D0006-0038
CWS01
DACA83-01-T-0001
DC000904
DC000905
FF98C017
H4000143
HSG00000
HSG00010
HSG91C19
J3000005
J3000089
J3000103
J3000111
J3000112
J3000187

If I wanted to select all of the columns then I would do a select * pronounced "Select Star" or "Select All". It would look like this:

```
1 select * from contract|
```



ACO_AGENT_CODE	ACTIVITY_VARIANCE	ACT_PER	ACT_PER_LESS_STORED
	178571.42	0	0
	-1318887	100	100
	0	0	0
	24299	50	50
	5000	75	52
	0	0	0
	0	0	0
	0	95	95
	0	0	0
	0	0	0
	0	0	0
	0.52	0	0
	0	100	100
	0	0	0

Records: 346 Columns: 377 0%

You can also do math on number columns. Math in SQL follows the normal order of precedence. Multiplication (*) and Division (/) before addition (+) and Subtraction (-). Operators of the same priority are evaluated left to right. Use parentheses to change the order of evaluation.

```
1 select contract_id, act_tot, 2*act_tot+10 "My Custom Number" from contract
```

CONTRACT_ID	ACT_TOT	My Custom Number
J3000433	0	10
J3000329	1318887	2637784
J3000483	0	10
J3000484	431387	862784
J3000089	2253873	4507756
J3000111	0	10
J3000112	0	10
97C0058	462797	925604
J3000865	0	10
J3000863	0	10
J3000864	0	10
Pw99C005	38799726.48	177599462.96
99C0012	35529653	71059316
J3000353	0	10
DACA83-01-T-0001	0	10

Records: 346 Columns: 3 0%

NOTE: A NULL value is a column value that has not been assigned or has been set to NULL. It is not a blank, space or a zero. It is undefined. Because a NULL is undefined, there is no such thing as NULL math. As a result the following will equal NULL;

- $\text{NULL} + 4 = \text{NULL}$
- $\text{NULL} * 3 = \text{NULL}$

Since NULL is undefined, all math using a NULL returns a NULL

2.2 Distinct Clause

Many times there are multiple rows with the same value, and we want to return only one copy of the row. If the boss wants a list of the offices currently containing contracts, we can query that from the contract table.

```
1 select office_symbol from contract
```

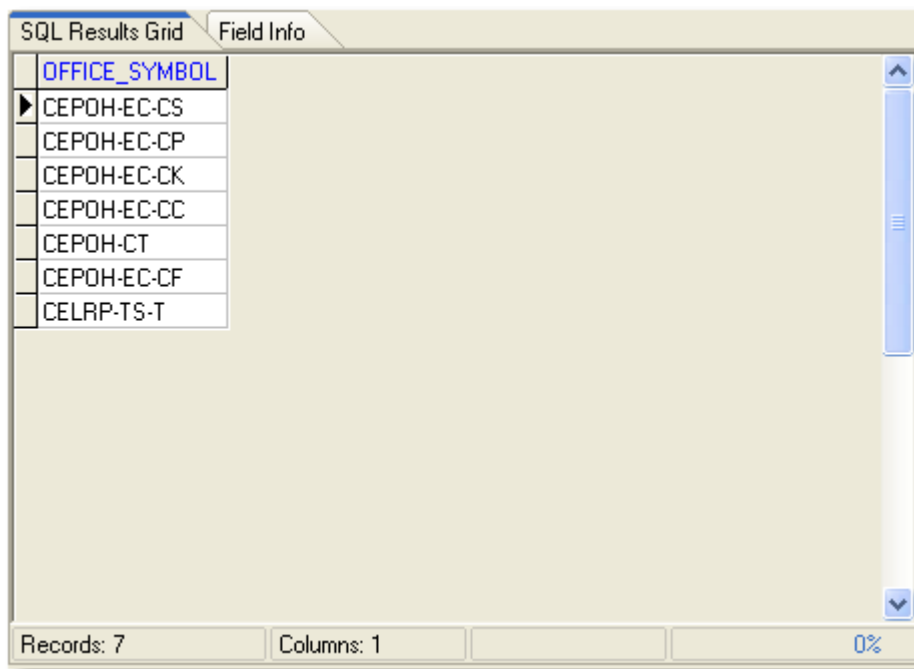


The screenshot shows a window titled "SQL Results Grid" with a "Field Info" tab. The main area displays a list of office symbols. The first row is "OFFICE_SYMBOL". Below it, there are 346 rows of data, each representing a contract. The symbols are: CEPOH-CT, CEPOH-EC-CF, CEPOH-EC-CF, CEPOH-EC-CF, CEPOH-EC-CC, CEPOH-EC-CC, CEPOH-EC-CC, CEPOH-EC-CC, CEPOH-EC-CF, CEPOH-EC-CS, CEPOH-EC-CS, CEPOH-EC-CF, CEPOH-EC-CP, CEPOH-EC-CS, CEPOH-EC-CF, and CEPOH-EC-CC. The status bar at the bottom indicates "Records: 346", "Columns: 1", and "0%".

OFFICE_SYMBOL
CEPOH-CT
CEPOH-EC-CF
CEPOH-EC-CF
CEPOH-EC-CF
CEPOH-EC-CC
CEPOH-EC-CC
CEPOH-EC-CC
CEPOH-EC-CC
CEPOH-EC-CF
CEPOH-EC-CS
CEPOH-EC-CS
CEPOH-EC-CF
CEPOH-EC-CP
CEPOH-EC-CS
CEPOH-EC-CF
CEPOH-EC-CC

We have 346 contracts, and we got 346 rows back. But, notice that some contracts have the same office_symbol. What we want is a list of the distinct office_symbols (one row for each office_symbol). SQL provides the DISTINCT clause for this result.

```
1 select Distinct(office_symbol) from contract
```



The Distinct Clause removes the duplicate rows.

2.3 Where Clause

The WHERE clause also limits the number of rows in the results set. The WHERE clause is a logical comparison and returns the row if the WHERE clause is true and excludes the row if the clause is false.

TRUE

```
1 select fmt_contract_delivery from contract
2 where contract_id = 'J3000111'
3
```



Notice that the column used in the WHERE clause is not one of my selected columns. It can be, but there is no requirement for it to be selected. Also, note that I capitalized the argument. The field is a varchar2 or a character string. Character strings are enclosed in single quotes. Although capitalization does not matter in the SQL command syntax, it does matter with data. The information that is stored in the database might be capitalized or not. If I use my WHERE clause regardless of case I could get back a result set that is not what I wanted.

FALSE

```
1 select fmt_contract_delivery from contract
2 where contract_id = 'j3000111'
3
```



If for some reason you do not know how a data element is stored you could use one of the built in functions Oracle provides to force your case on the data field.


```
1 select fmt_contract_delivery from contract
2 where lower(contract_id) = 'j3000111'
3
```



SQL Logical Operators

An SQL query can only have one WHERE clause; however, that clause can contain multiple comparisons. Each comparison returns a TRUE, FALSE or NULL. You evaluate these TRUE/FALSE results using AND and OR to end up with the single TRUE or FALSE for the entire WHERE clause. The AND operator (called conjunction) returns TRUE if both comparisons are TRUE and returns FALSE if either comparison is FALSE.

```
Where act_tot < 100000          --TRUE
AND contract_id = 'J3000111'   --TRUE
```

Since Both comparisons are TRUE, the WHERE clause is TRUE. The logical OR operator (called a disjunction) returns TRUE if either comparison is TRUE, otherwise returns FALSE.

```
WHERE act_tot < 100000         --TRUE
OR contract_id = 'J3000111'   --FALSE
```

The WHERE clause returns TRUE.

2.4 Dates and SQL

Dates are stored in the database as large numbers. The actual size of the data number is dependent on the operating system supporting the database. When a date is requested, it is returned in a human readable form.

When date values are compared in the WHERE clause, the format of the date must match the format that the database is using or the comparison will fail. Alternately, if you are using another format, then you must tell the database how your dates is formatted. The default date format for RMS is MM-

DD-YYYY. If you are using an RMS tool this format is set for you each time a new session is created. If you are using your own tool such as TOAD or some other SQL tool you will need to set your session variable using the " ALTER SESSION SET NLS_DATE_FORMAT = 'MM-DD-YYYY' " statement.

To get the current date, you select from a function called SYSDATE. SYSDATE returns the current date from the server operating system supporting the database.

```
1 select sysdate from dual
2
```

SYSDATE
8/18/2007 2:40:48 PM

The dual table is a pseudo-table that allows you to execute functions that require selecting from a table.

Lastly, because a date is stored in the database as a number, you can perform date math.

```
1 select sysdate today,
2 sysdate -1 yesterday,
3 sysdate +1 tomorrow from dual
```

TODAY	YESTERDAY	TOMORROW
8/18/2007 3:03:58 PM	8/17/2007 3:03:58 PM	8/19/2007 3:03:58 PM

Records: 1 Columns: 3 0%

Date Functions

Oracle also has some specific functions geared toward the manipulation and formatting of the dates.

add_months(d,n)

The add_months function gives you the same day, n number of months away. The n can be positive or negative.

```

1 select
2   SYSDATE,
3   ADD_MONTHS(SYSDATE,-1),
4   ADD_MONTHS(SYSDATE,2),
5   ADD_MONTHS(SYSDATE,-3)
6 from
7   dual

```

SYSDATE	ADD_MONTHS(SYSDATE,-1)	ADD_MONTHS(SYSDATE,2)	ADD_MONTHS(SYSDATE,-3)
8/20/2007 8:00:53 AM	7/20/2007 8:00:53 AM	10/20/2007 8:00:53 AM	5/20/2007 8:00:53 AM

last_day(d)

The last_date function returns the last day of the month of the date d. If you want to find the first day of the next month simply add one to the last_day results.

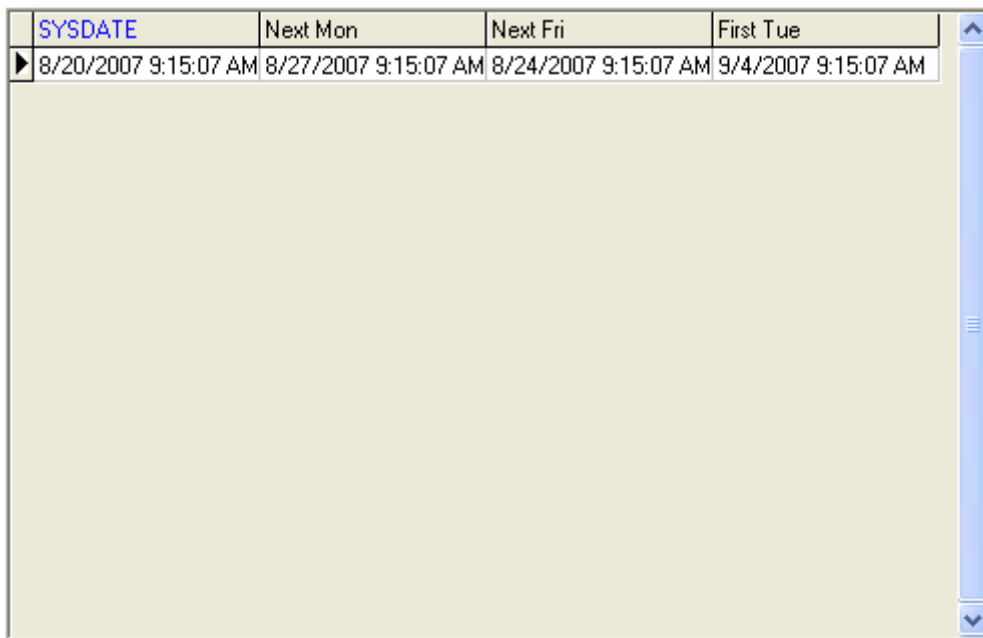
```
1 select
2     SYSDATE,
3     LAST_DAY(SYSDATE) EOM,
4     LAST_DAY(SYSDATE)+1 FOM
5 FROM
6     dual
```

SYSDATE	EOM	FOM
8/20/2007 9:10:04 AM	8/31/2007 9:10:04 AM	9/1/2007 9:10:04 AM

next_day(d, day_of_week)

The `next_day` function returns the date of the `day_of_week` after date `d`. `day_of_week` can be the full name or abbreviation. Below, we get the date for next Monday, next Friday, and the first Tuesday of next month.

```
1 select
2     SYSDATE,
3     NEXT_DAY(SYSDATE, 'MONDAY') "Next Mon",
4     NEXT_DAY(SYSDATE, 'FRIDAY') "Next Fri",
5     NEXT_DAY(LAST_DAY(SYSDATE)+1, 'TUESDAY') "First Tue"
6 from
7     dual
```



SYSDATE	Next Mon	Next Fri	First Tue
8/20/2007 9:15:07 AM	8/27/2007 9:15:07 AM	8/24/2007 9:15:07 AM	9/4/2007 9:15:07 AM

to_char(date, format)

The `to_char` function will change a date to characters in the format defined in the format field. If you do not define a format, the date will be returned in the default format format set for the database. Formatting commands are enclosed in single quotes and are case sensitive and can include any valid data format element.

Date Formatting Elements

D	Day of the week as in 1 thru 7.
DD	Day of the Month as in 1 thru 31.
DDD	Day of the Year as in 1 thru 366.
DY	Day of the Week abbreviated Mon thru Sun
DAY	Day of the Week Monday, Tuesday...
W	Week of the Month as in 1 thru 5
WW	Week of the Year as in 1 thru 53.
MON	Month abbreviated Jan thru Dec.
MONTH	Month Spelled out. January thru December.
YY	Year, Last two digits as in 06 and 07
YYYY	Year, four digit as in 2006 and 2007
YEAR	Year spelled out.
HH	Hour in 12 hour clock. 1 thru 12.
HH24	Hour in 24 hour clock 1 thru 24.
MI	Minutes as in 1 thru 59
SS	Seconds as in 1 thru 59
SSSSS	Seconds of the day as in 1 thru 86399
AM, PM	Meridian Indicator
A.M., P.M.	Meridian Indicator with periods.

Example 1

```

1 select
2   TO_CHAR(SYSDATE) "SYSDATE",
3   TO_CHAR(SYSDATE, 'MON-DD-YYYY') "Date 1",
4   TO_CHAR(SYSDATE, 'MON:DD:YYYY') "Date 2",
5   TO_CHAR(SYSDATE, 'DAY, MONTH DD, YEAR') "Date 3"
6 from
7   dual

```

SYSDATE	Date 1	Date 2	Date 3
08-20-2007	AUG-20-2007	AUG:20:2007	MONDAY ,AUGUST 20,TWO THOUSAND SEVEN

Example 2

```

1 select
2   TO_CHAR(SYSDATE) "SYSDATE",
3   TO_CHAR(SYSDATE, 'HH24') "Time 1",
4   TO_CHAR(SYSDATE, 'HH AM') "Time 2",
5   TO_CHAR(SYSDATE, 'HH:MM:SS AM') "Time 3"
6 from
7   dual

```

SYSDATE	Time 1	Time 2	Time 3
08-20-2007	09	09 AM	09:08:10 AM

to_date(text, format)

The `to_date` function takes text and uses the formatting codes to convert the text into a date data type. The format is telling the database that the text is in that format. Remember that the database stores a date as a number, so it must understand what the text is representing as part of that date. The format codes listed in the `to_char` table are the same for the `to_date`.

```

1 select
2   SYSDATE,
3   TO_DATE(TO_CHAR(SYSDATE, 'MM/DD/YYYY'), 'MM/DD/YYYY') "DATE 1",
4   TO_DATE('05/05/2006', 'MM/DD/YYYY') "DATE 2",
5   TO_DATE('08/20/2007', 'MM/DD/YYYY') "DATE 3"
6 from
7   dual

```

SYSDATE	DATE 1	DATE 2	DATE 3
8/20/2007 9:55:40 AM	8/20/2007	5/5/2006	8/20/2007

2.5 Order By Clause

The only way to insure that the rows are ordered the way you want them is have the query specify the sorting using the `ORDER BY` clause. Also, sorting is always performed last.

```

1 select
2   distinct (office_symbol)
3 from
4   contract
5 order by
6   office_symbol

```

OFFICE_SYMBOL
▶ CELRP-TS-T
CEPOH-CT
CEPOH-EC-CC
CEPOH-EC-CF
CEPOH-EC-CK
CEPOH-EC-CP
CEPOH-EC-CS

You can also order by multiple columns.

```
1 select
2     contract_id,
3     office_symbol
4 from
5     contract
6 order by
7     office_symbol, contract_id
```

CONTRACT_ID	OFFICE_SYMBOL
▶ H4000143	CELRP-TS-T
J3000428	CEPOH-CT
J3000431	CEPOH-CT
J3000432	CEPOH-CT
J3000433	CEPOH-CT
J3000435	CEPOH-CT
J3000436	CEPOH-CT
J3000437	CEPOH-CT
J3000513	CEPOH-CT
J3000514	CEPOH-CT
J3000515	CEPOH-CT
J3000516	CEPOH-CT
J3000517	CEPOH-CT
J3000607	CEPOH-CT
J3000608	CEPOH-CT

2.6 Case Statement

The case statement is used to change values. It uses the syntax:

CASE selection WHEN x THEN y WHEN q THEN r ELSE z END

```

1  select
2     contract_id,
3     MOD_NO,
4     agencycode,
5     CASE AGENCYCODE
6         WHEN 'C' THEN 'Corps of Engineers'
7         WHEN 'E' THEN 'Environmental Protection Agency'
8         WHEN 'K' THEN 'Contractor'
9         WHEN 'P' THEN 'Potential Responsible Party'
10        WHEN 'S' THEN 'State'
11        WHEN 'T' THEN 'All other'
12        WHEN 'U' THEN 'User'
13        ELSE 'Unknown'
14        END "Agency Key"
15 from
16     pc_log

```

CONTRACT_ID	MOD_NO	AGENCYCODE	Agency Key
J3000553	R00001	U	User
J3000475	R00007	C	Corps of Engineers
J3000519	R00008	C	Corps of Engineers
J3000519		C	Corps of Engineers
J3000446		U	User
J3000519		C	Corps of Engineers
J3000534	R00007	C	Corps of Engineers
J3000287	R00014	C	Corps of Engineers
J3000460		C	Corps of Engineers
J3000717	R00003	U	User
J3000462		C	Corps of Engineers
J3000417		C	Corps of Engineers
J3000419	R00013	C	Corps of Engineers
J3000530	R00018	K	Contractor
J3000506	R00001	U	User
J3000449	R00003	C	Corps of Engineers
J3000441	R00010	K	Contractor
J3000442	R00012	C	Corps of Engineers
J3000559		C	Corps of Engineers
J3000219	R00092	U	User

2.7 SQL Comparison Operators

A comparison operator evaluates two values and returns a TRUE, FALSE, or NULL. Comparison operators are used in the WHERE clause to limit the number of returned rows.

Equals	=	WHERE first_name = 'DORINDA'
Not Equals	!=	WHERE state != 'CA'
	<>	WHERE state <> 'CA'
	^=	WHERE state ^= 'CA'
Less Than	<	WHERE pay < min_wage
Greater Than	>	WHERE pay > my_pay
Less Than or Equal	<=	WHERE pay <= 2000
Greater Than or Equal	>=	WHERE pay >= 100000

There are special comparison operator that are used with multiple values.

BETWEEN... AND	Validates that a value is between the first and second values, inclusive	WHERE pay between 100000 and 150000
IN (...)	Validates that a value is contained in the list of values.	WHERE state IN ('CA','FL','GA','CO')
LIKE	Like matches a character pattern. There are two special chracters used to match characters. Ther percent % is zero or more chracters wildcard. The underscore _ is a single character wildcard. If you are looking to match one of the special characters then you will need to use the escape character \ so taht the database treats it as a literal and not as a wildcard character.	WHERE contract_id LIKE 'J30001%' (This will return any match that starts with 'J30001') WHERE process_name LIKE 'ora_%' (This will return all the rows where process_name started with ora_)
NOT	The NOT operator simply negates the operator following.	WHERE pay NOT BETWEEN 100000 and 150000 WHERE state NOT IN ('CA','FL','GA','CO')
EXIST	The EXIST operator returns TRUE if a subquery returns at least one row. Likewise, NOT EXIST returns TRUE if the subquery does not return at least one row.	WHERE contract_id NOT LIKE 'J30001%' WHERE EXIST(select contract_id from contract where contract_id is not null)
IS NULL	Returns TRUE if the value is NULL. IS NOT NULL returns TRUE if the value is not NULL	WHERE contract_id is NULL

2.8 Oracle Functions

Built-in SQL Functions

Basically a function takes one or more inputs and returns one value as a result. All programming languages support functions; and the Oracle database uses a language called PL/SQL to supply most of the standard functions.

Character or Text Functions

upper(string)

```

1 select
2     upper(proj_ds)|
3 from
4     contract
5 where
6     proj_ds is not null

```

UPPER(PROJ_DS)
WORK WILL INCLUDE GENERAL DESIGN AND CONSTRUCTION PROJECTS TO INCLUDE BUT NOT LIMITED TO SUCH AR
FY03 MILCON KNMD023014 INSTALLATION OF FLIGHTLINE SECURITY FENCING AND GATES, HICKAM AIR FORCE BASE, I
REPLACE TWO DETERIORATED AUTOMATIC TRANSFER SWITCHES (ATS) ON THE E1 AND E2 CIRCUITS WITH NEW ATS
THIS IS A TEST PROJECT TO TEST THE RMS/CMS INTERFACE.
REPLACE EXISTING ROOFS ON THE 9TH AND 10TH FLOORS OF WINGS A, B, AND C WITH NEW TOUCHED APPLIED MOO
PALAU COMPACT ROAD PROJECT, BABELDAOB ISLAND, REPUBLIC OF PALAU
FY96 MCA PN 42470, KOLEKOLE CONNECTOR ROAD, KOLEKOLE-TRIMBLE CONNECTOR AND FY98 MCA PN 44839 PH 1D-
BASIC CONTRACT FOR INDEFINITE DELIVERY INDEFINITE QUANTITY (IDIQ) CONSTRUCTION AND SERVICES CONTRACT
FY02 MCA PN50846 COLD STORAGE FACILITY AND FY01 RDT&E REPAIR WATER TANKS, U.S. ARMY KWAJALEIN ATOLL
OBJECTIVE OF THIS PROJECT IS TO EXPAND THE EXISTING HOT CARGO PAD FACILITY WITH TWO ADDITIONAL HOT C
SANITARY LANDFILL EXPANSION FOR THE FORT IRWIN ARMY INSTALLATION.
GENERAL CONTRACTING SERVICES FOR VARIOUS SCHOOLS, DAHU, HAWAII, IN ACCORDANCE WITH REVISED SCOPE I

lower(string)

```

1 select
2     lower|(proj_ds)
3 from
4     contract
5 where
6     proj_ds is not null

```

LOWER(PROJ_DS)
work will include general design and construction projects to include but not limited to such areas as civil, architectural, structural, me
fy03 milcon knmd023014 installation of flightline security fencing and gates, hickam air force base, oahu, hawaii
replace two deteriorated automatic transfer switches (ats) on the e1 and e2 circuits with new ats with bypass capability in the g1 b su
this is a test project to test the rms/cms interface.
replace existing roofs on the 9th and 10th floors of wings a, b, and c with new touched applied modified bitumen roofing. existing roc
palau compact road project, babeldaob island, republic of palau
fy96 mca pn 42470, kolekole connector road, kolekole-trimble connector and fy98 mca pn 44839 ph 1d-2 and fy99 mca pn 46901 p
basic contract for indefinite delivery indefinite quantity (idiq) construction and services contract, honolulu/alaska engineer district are
fy02 mca pn50846 cold storage facility and fy01 rdt&e repair water tanks, u.s. army kwajalein atoll
objective of this project is to expand the existing hot cargo pad facility with two additional hot cargo pads.
sanitary landfill expansion for the fort irwin army installation.
general contracting services for various schools, oahu, hawaii, in accordance with revised scope of work dated nov. 16, 2005.

initcap(string)

<pre> 1 select 2 initcap(proj_ds) 3 from 4 contract 5 where 6 proj_ds is not null </pre>	
--	--

INITCAP(PROJ_DS)
Work Will Include General Design And Construction Projects To Include But Not Limited To Such Areas As Civil, Architectural, Struc
Fy03 Milcon Knmd023014 Installation Of Flightline Security Fencing And Gates, Hickam Air Force Base, Oahu, Hawaii
Replace Two Deteriorated Automatic Transfer Switches (Ats) On The E1 And E2 Circuits With New Ats With Bypass Capability In Th
This Is A Test Project To Test The Rms/Cms Interface.
Replace Existing Roofs On The 9th And 10th Floors Of Wings A, B, And C With New Touched Applied Modified Bitumen Roofing. E
Palau Compact Road Project, Babeldaob Island, Republic Of Palau
Fy96 Mca Pn 42470, Kolekole Connector Road, Kolekole-Trimble Connector And Fy98 Mca Pn 44839 Ph 1d-2 And Fy99 Mca Pn 4
Basic Contract For Indefinite Delivery Indefinite Quantity (Idiq) Construction And Services Contract, Honolulu/Alaska Engineer Distric
Fy02 Mca Pn50846 Cold Storage Facility And Fy01 Rdt&E Repair Water Tanks, U.S. Army Kwajalein Atoll
Objective Of This Project Is To Expand The Existing Hot Cargo Pad Facility With Two Additional Hot Cargo Pads.
Sanitary Landfill Expansion For The Fort Irwin Army Installation.
General Contracting Services For Various Schools, Oahu, Hawaii, In Accordance With Revised Scope Of Work Dated Nov. 16, 200

concat(string1,string2)

<pre> 1 select 2 concat('My Contract ID is ',contract id) "Data 1", 3 concat(contract_id,contract_id) "Data 2", 4 contract_id ' - ' 5 contract_id "Data 3" 6 from 7 contract 8 where 9 proj_ds is not null </pre>	
---	--

Data 1	Data 2	Data 3
My Contract ID is J3000433	J3000433J3000433	J3000433 - J3000433
My Contract ID is J3000329	J3000329J3000329	J3000329 - J3000329
My Contract ID is J3000484	J3000484J3000484	J3000484 - J3000484
My Contract ID is J3000089	J3000089J3000089	J3000089 - J3000089
My Contract ID is 97C0058	97C005897C0058	97C0058 - 97C0058
My Contract ID is Pw99C005	Pw99C005Pw99C005	Pw99C005 - Pw99C005
My Contract ID is 99C0012	99C001299C0012	99C0012 - 99C0012
My Contract ID is J3000353	J3000353J3000353	J3000353 - J3000353
My Contract ID is J3000220	J3000220J3000220	J3000220 - J3000220
My Contract ID is J3000449	J3000449J3000449	J3000449 - J3000449
My Contract ID is PROMISTEST	PROMISTESTPROMISTEST	PROMISTEST - PROMISTEST

substr(string1, start, count)

```

1  select
2     contract_id,
3     substr(contract_id, 1, 2) Prefix
4  from
5     contract
6  where
7     proj_ds is not null

```

CONTRACT_ID	PREFIX
J3000433	J3
J3000329	J3
J3000484	J3
J3000089	J3
97C0058	97
Pw99C005	Pw
99C0012	99
J3000353	J3
J3000220	J3
J3000449	J3
PROMISTEST	PR
J3000617	J3

length(string)

```

1  select
2     Length(Proj_ds) "Proj DS Length"
3  from
4     contract
5  where
6     proj_ds is not null

```

Proj DS Length
223
113
221
53
252
96
188
158
99
110
65
547

ltrim(s1,s2) / rtrim (s1, s2)

Sometimes you need to remove characters from the beginning and or end of a string. Normally you are removing spaces, but you may need to remove other characters. Ltrim removes any character in s2 from the front of s1. Think of s2 as a list of characters rather than a word. Rtrim does the same thing except it removes the character from the end of s1. The string s2 defaults to a space. If the characters in s2 are not in s1, then s1 is returned unchanged.

trim(s2 from s1)

The trim function incorporates both ltrim and rtrim in one command. You can set trim to remove the leading, trailing or both. The default is both. The string s2 defaults to a space. When using trim, you can only define one character to trim. With rtrim or ltrim, you are not restricted.

```

1  select
2     LTRIM('abcdedcba', 'abc') Left,
3     RTRIM('abcdedcba', 'abc') Right,
4     TRIM(LEADING 'a' FROM 'abcdedcba') TRIML,
5     TRIM(TRAILING 'a' FROM 'abcdedcba') TRIMR,
6     TRIM(BOTH 'a' FROM 'abcdedcba') TRIMB,
7     TRIM(' abcdedcba ') TRIMSPACE
8  from
9     dual
10

```

LEFT	RIGHT	TRIML	TRIMR	TRIMB	TRIMSPACE
dedcba	abcded	bcdedcba	abcdedcb	bcdedcb	abcdedcba

Number Functions

round(n,d)

The round function rounds a number n to the specified decimal d. The decimal d can be positive, negative or zero.

```
1 select
2     ROUND(1234.345, 2),
3     ROUND(1234.345, 0),
4     ROUND(1234.345, -2)
5 from
6     dual
```

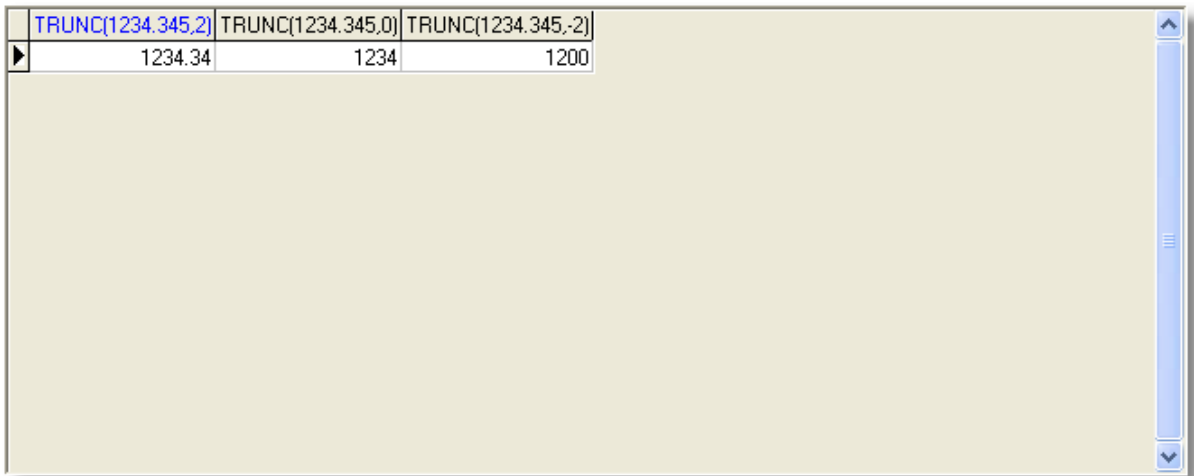
	ROUND(1234.345,2)	ROUND(1234.345,0)	ROUND(1234.345,-2)
▶	1234.35	1234	1200

trunc(n,d)

The trunc or truncate function simply drops the digits without rounding. The decimal d can again be positive, negative or zero. If the number truncated is five or higher, it is still dropped without rounding the next digit up.

```
1 select
2     TRUNC(1234.345, 2),
3     TRUNC(1234.345, 0),
4     TRUNC(1234.345, -2)
5 from
6     dual
```

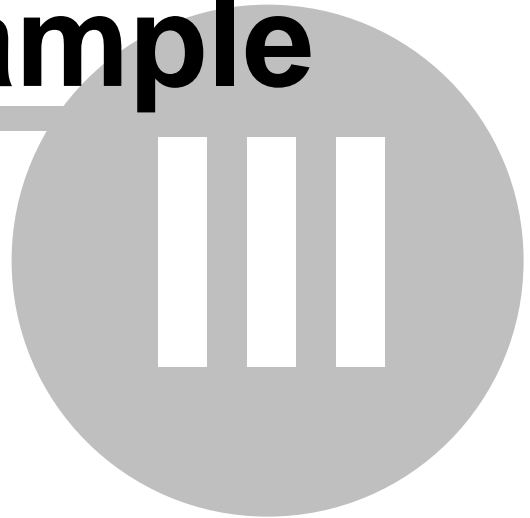
	<code>TRUNC(1234.345,2)</code>	<code>TRUNC(1234.345,0)</code>	<code>TRUNC(1234.345,-2)</code>
▶	1234.34	1234	1200



RMS Report Writing

Example Report
For Training Purposes

Example



3 RAP Primer

3.1 Overview

RAP is a subset of Delphi's Object Pascal and stands for Report Application Pascal (RAP). Most language elements in Delphi are recognized by the RAP compiler and can be found in the Code Toolbox. The following is a list of elements contained in the Toolbox.

Data Types:

- Boolean
- Currency
- Double
- Extended
- Integer
- Single
- Char
- String
- Date
- DateTime
- Time
- Color

Variant

Statements:

- Case statements
- If-then statements
- If-then-else statements
- For loops
- While loops

Operators:

- Assignment (:=)
- Boolean (and, not, or, xor)
- Class (as, is)
- Math (-, +, *, /, div, mod)
- Relational (<, <=, <>, =, >, >=)
- String (+)
- Unary (-, +)

Currently Unsupported Elements:

- Class declarations
- Arrays
- Record types
- Set types

Value and Variable parameters

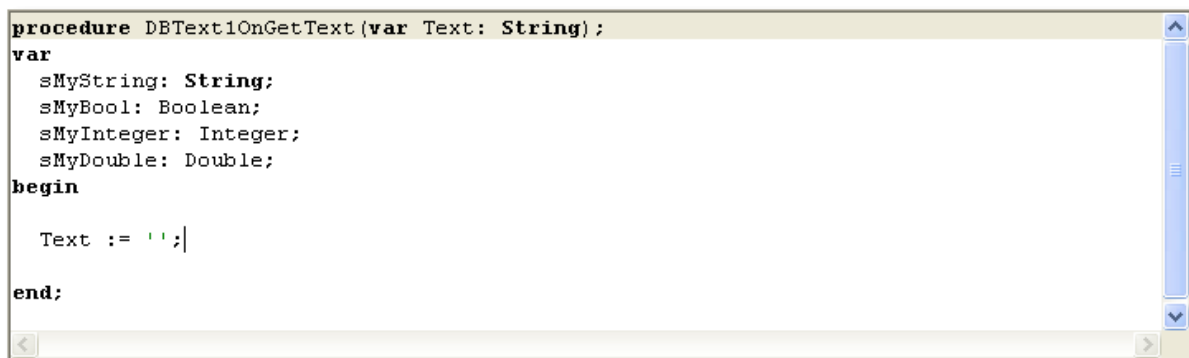
Most parameters are either value parameters (default) or variable (var) parameters. Value parameters are passed by value, while variable parameters are passed by reference. To see what this means, consider the following functions.

3.2 Data Types

Here are the list of Data Types available to be used withing RAP.

Boolean
Currency
Double
Extended
Integer
Single
Char
String
Date
DateTime
Time
Color
Variant

To add a local variable declaration to a function, activate the Code Editor for the current item and place the cursor between the function's declaration and the begin. Type var and declare your variables.



```
procedure DBText1OnGetText (var Text: String);  
var  
  sMyString: String;  
  sMyBool: Boolean;  
  sMyInteger: Integer;  
  sMyDouble: Double;  
begin  
  Text := ' ';  
end;
```

3.3 Operators

Operators behave like predefined functions that are part of the internal RAP language. For example, the *expression* (a construction which returns a value) **(X + Y)** is built from the variables **X** and **Y** --- which are called operands --- with the **+** operator; when X and Y represent integers or real numbers then (X+Y) will return their sums.

Some operators behave differently depending on the type of data passed to them. For example the addition operator (**+**) when used with integers performs a sum but when used with two string performs concatenation.

Example:

X + Y = sum of X and Y
'One' + ' ' + 'One' = One One

Assignment (:=)

The **assignment** operator (:=) assigns a variable to an expression.

```
variable := expression;
```

Examples:

```
l := 3;  
X := Y + Z;  
mString := 'This is my string';  
bMyBoolean := False;
```

Example in Reportbuilder

```
procedure DetailBeforeGenerate;  
var  
    iCount: Integer;  
    bStarted: Boolean;  
    cMyColor: TColor;  
begin  
    ktr.First;  
    iCount := 0;  
  
    while not ktr.EOF do  
        begin  
            if ktr['DUNS4'] = RBVU_DISTINCT_KTR_DUNS4['DUNS'] then  
                begin  
                    iCount := iCount+1;  
                end;  
  
                ktr.Next;  
            end;  
  
            case iCount of  
                0..3: cMyColor := clBlack;  
                4..6: cMyColor := clYellow;  
                7..8: cMyColor := clBlue;  
            else  
                cMyColor := clRed;  
            end;  
  
            ktr.First;  
            Label2.Text := IntToStr(iCount);  
            Label2.Font.Color := cMyColor;  
  
        end;  
end;
```

As you can see we create a variable in the **var** section and then assign it in the code section which is

between the **begin** and **end** statements.

Boolean (and, not, or, xor)

The Boolean operators take operands of any Boolean type and return a value of type Boolean (True or False).

Used in if and While statements.

Class (as, is)

Math (-, +, *, /, div, mod)

Math operators, which take real or integer operands

Relational (<, <=, <>, =, >, >=)

Relational operators are used to compare two operands.

String (+)

Unary (-, +)

3.4 If Statement

Using the If then statement.

There are two forms of the **if** statement: **if...then** and the **if...then...else**. The syntax of an **if...then** statement is

```
if expression then statement
```

where expression returns a Boolean value. If expression is True, then statement is executed; otherwise it is not. For example,

```
if J <> 0 then Result := I/J;
```

The syntax of an if...then...else statement is

```
if expression then statement1 else statement2
```

where expression returns a Boolean value. If expression is True, then statement1 is executed; otherwise statement2 is executed. For example,

```
if J = 0 then  
  Exit  
else  
  Result := I/J;
```

The then and else clauses contain one statement each, but it can be a structured statement. For example,

```
if J <> 0 then  
  begin  
    Result := I/J;  
    Count := Count + 1;  
  end  
else if Count = Last then
```

```
Done := True  
else  
Exit;
```

```
if AValue = '1' then  
    Result := 'FUTURE'  
else if AValue = '2' then  
    Result := 'ACTIVE'  
else if AValue = '3' then  
    Result := 'CONSTRUCTION COMPLETE'  
else if AValue = '4' then  
    Result := 'PHYSICAL COMPLETE'  
else if AValue = '5' then  
    Result := 'FINAL PAYMENT'  
else if AValue = '6' then  
    Result := 'FISCAL COMPLETE'  
else  
    Result := '';
```

Example in Report builder

```
procedure Memo1OnPrint;
var
  lsLine: String;
  lsState: String;
  lsZIP: String;
begin
  {clear memo}
  Memo1.Lines.Clear;

  {add contact}
  lsLine := p1Customer['Contact'];

  {Check to see if lsLine has a value}
  if lsLine <> '' then
    Memo1.Lines.Add(lsLine);

  {Check to see if lsLine has a specific number of characters.}
  if Length(lsLine) = 1 then
    begin
      Memo1.Lines.Add('Has one character.')
    end
  else if Length(lsLine) >= 10 then
    Memo1.Lines.Add('Has multiple characters greater than 10')
  else if (Length(lsLine) > 0) AND (Length(lsLine) < 10) then
    Memo1.Lines.Add('Has multiple characters less than 10.')
  else
    Memo1.Lines.Add('Has no characters');

end;
```

3.5 Case Statement

Case Statements

The **case** statement may provide a readable alternative to deeply nested if conditionals. A **case** statement has the form

```
case selectorExpression of
  caseList1: statement1;
  .
  .
  .
```

```
    caseList_n: statement_n;  
end
```

A **case** statement can have a final else clause:

```
case selectorExpression of  
    caseList1: statement1;  
    .  
    .  
    .  
    caseList_n: statement_n;  
else  
    statements;  
end
```

where *selectorExpression* is any expression of an ordinal type (string types are invalid) and each caseList is one of the following;

A numeral, declared constant, or other expression that the compiler can evaluate.

Examples of case statements

```
case I of  
    1..5: Caption := 'Low';  
    6..9: Caption := 'High';  
    0, 10..99: Caption := 'Out of Range';  
else  
    Caption := "";  
end;
```

is equivalent to the nested conditional

```
if I in [1..5] then  
    Caption := 'Low'  
else if I in [6..10] then  
    Caption := 'High'  
else if (I = 0) or (I in [10..99]) then  
    Caption := 'Out of Range'  
else  
    Caption := "";  
end;
```

Other examples of **case** statements:

```
case MyColor of  
    Red: X := 1;  
    Green: X := 1;  
    Blue: X := 1;  
    Yellow, Orange, Black: X := 0;  
end;
```

```
case Selection of  
    Done: Form1.Close;  
    Compute: CalculateTotal();  
else  
    Beep;  
end;
```



```
case AValue of
  1:Result := 'FUTURE';
  2:Result := 'ACTIVE';
  3:Result := 'CONSTRUCTION COMPLETE';
  4:Result := 'PHYSICAL COMPLETE';
  5:Result := 'FINAL PAYMENT';
  6:Result := 'FISCAL COMPLETE';
else
  Result := '';
end;
```

Example in Report Builder

```
procedure DetailBeforeGenerate;
var
  iCount: Integer;
  bStarted: Boolean;
  cMyColor: TColor;
begin
  ktr.First;
  iCount := 0;

  while not ktr.EOF do
    begin
      if ktr['DUNS4'] = RBVU_DISTINCT_KTR_DUNS4['DUNS'] then
        begin
          iCount := iCount+1;
        end;

      ktr.Next;
    end;

    case iCount of
      0..3: cMyColor := clBlack;
      4..6: cMyColor := clYellow;
      7..8: cMyColor := clBlue;
    else
      cMyColor := clRed;
    end;

    ktr.First;
    Label2.Text := IntToStr(iCount);
    Label2.Font.Color := cMyColor;
  end;
```

3.6 While Statement

While Statements

The syntax of a while statement is

```
while expression do statement
```

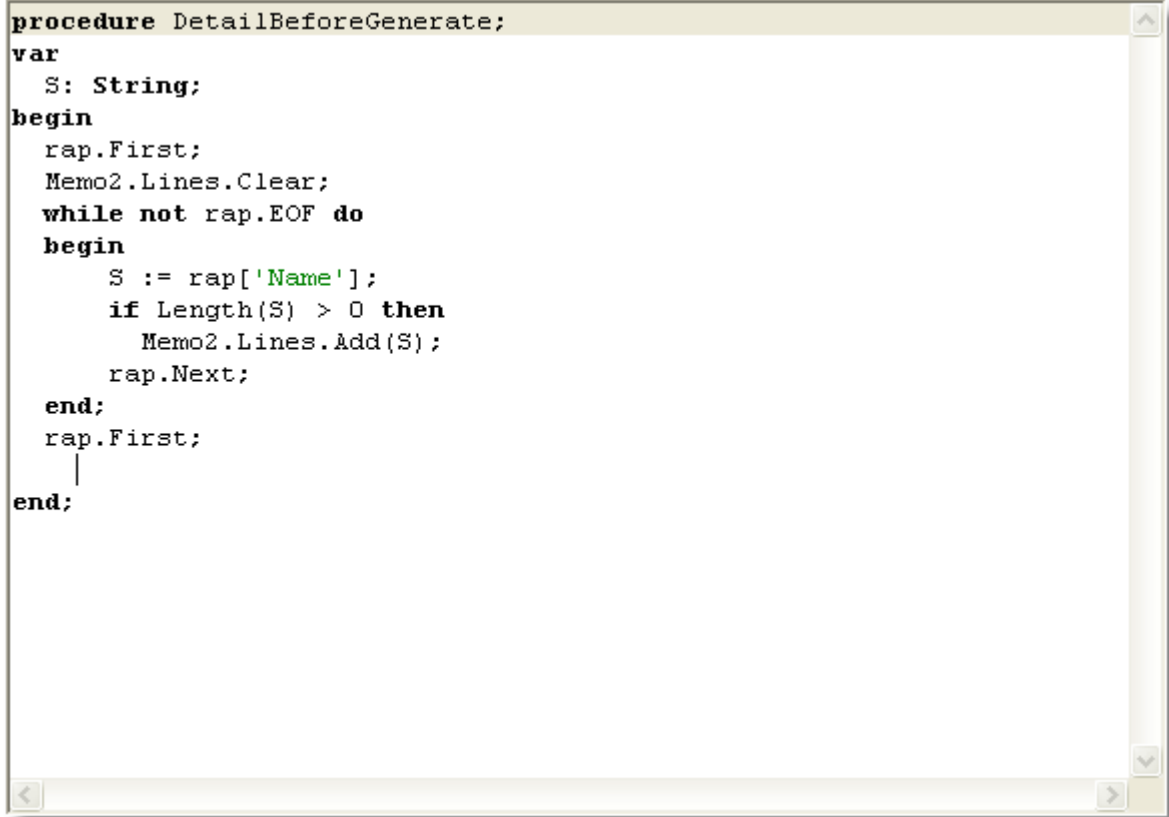
where *expression* returns a Boolean value and *statement* can be a compound statement. The **while** statement executes its constituent *statement* repeatedly, testing *expression* before each iteration. As long as *expression* returns *True*, execution continues.

Examples of a **while** statements include

```
while Data[I] <> X do I := I + 1;
```

```
while I > 0 do  
begin  
  if Odd(I) then Z := Z * X;  
  I := I div 2;  
  X := Sqr(X);  
end;
```

Example in Report Builder



```
procedure DetailBeforeGenerate;  
var  
  S: String;  
begin  
  rap.First;  
  Memo2.Lines.Clear;  
  while not rap.EOF do  
    begin  
      S := rap['Name'];  
      if Length(S) > 0 then  
        Memo2.Lines.Add(S);  
      rap.Next;  
    end;  
  rap.First;  
end;
```

3.7 Procedures and Functions

Procedures and functions are self-contained statement blocks that can be called from different locations within RAP. A function is a routine that returns a value when it executes. A procedure is a routine that does not return a value.

Function calls, because they return a value, can be used as expressions in assignments and operations. For example,

```
I := SomeFunction(X);
```

calls `SomeFunction` and assigns the result to `I`. Function calls cannot appear on the left side of an assignment statement.

```
SomeFunction(X) := I;
```

Function calls can be used as complete statements though.

```
SomeFunction(X);
```

The functions return value is then discarded.

Declaring Procedures and Functions in RAP

A procedure declaration in RAP has the form;

```
procedure procedureName(parameterList);  
localDeclarations; //optional  
begin  
  statements  
end;
```

A function declaration is like a procedure declaration except that it specifies a return type and a return value. Function declarations have the form;

```
function functionName(parameterList): returnType;  
localDeclarations;  
begin  
  statements  
end;
```

ParameterList

Most procedure and function headers include a parameter list. For example;

```
function TotalUp(X: Integer; Y: Integer): Integer;
```

the parameter list is `(X: Integer; Y: Integer)`.

Other types of parameter list examples include;

```
(X, Y: Integer);
```

```
(s1, s2: String);  
(var S: string; X: Integer);
```

Statements

Any valid expression which include other functions or procedures.

Examples include;

```
X := 1 + 2;  
MyString := 'Hello RMS';  
Result := TotalUp(X,Y);
```

3.8 Gotchas

When using the **RAP** code compiler there are many different things that could cause you to stumble when you are new to the language.

Assignment Operator (:=) VS Logical Operator (=)

When to use the semicolon in an if statement. (;)

Example 2:

```
if Project_Stage = 1 then  
  s := 'Future'  
else  
  s := 'Not Future';
```

Example 2:

```
if v = 1 then  
  s := 'Future'  
else if v = 2 then  
  s := 'Active'  
else if v = 3 then  
  s := 'Construction Complete'  
else  
  s := 'Not 1, 2 or 3';
```

```
if v = 1 then  
begin  
  s := 'Future';  
end  
else if v = 2 then  
begin  
  s := 'Future';  
end  
else if v = 3 then
```

```
    s := 'Construction Complete'  
else if v = 4 then  
    s := 'Physical Complete'  
else if v = 5 then  
    s := 'Final Payment'  
else  
    s := 'Fiscal Complete';
```

When to use the begin and end statements.

Used for syntax requirements.

Used for multiline statements.

Back Cover