# Table of Contents

INTRODUCTION ................................................................................................... 3

DATA TYPES.......................................................................................................... 3

- SELECT ............................................................................................................. 3
- WHERE CLAUSE ............................................................................................. 3
- DESCR ............................................................................................................... 4
- JOINS ................................................................................................................. 4
- ORDER BY ........................................................................................................ 5
- AGGREGATES, GROUP BY and HAVING .............................................................. 6

USEFUL FUNCTIONS ........................................................................................... 8

- || (concatenation) ............................................................................................ 8
- rtrim, ltrim ....................................................................................................... 9
- rpad, lpad ......................................................................................................... 10
- to_char, to_date, sysdate .................................................................................. 10
- rownum ............................................................................................................ 11
- &variables ....................................................................................................... 11
- EXISTS ............................................................................................................. 12
- UNION clause .................................................................................................. 12
- spool ................................................................................................................ 13
- save, @ ............................................................................................................. 14

SOME SPECIFIC USES OF SQL IN ALEPH ......................................................... 15
Introduction

The following tutorial is for users of SQL in the SQL+ environment. All the examples used come exclusively from ALEPH Z tables.

For consistency the following script color conventions are used:

- **Red** = user input
- **Blue** = screen response
- **Purple** = variables

**DATA TYPES**

There are several datatypes in Oracle. For this tutorial, we will be using CHAR, VARCHAR (both used for text strings), NUMBER and DATE. Note that in ALEPH, dates are of type NUMBER, not type DATE.

**SELECT**

Basic SQL command to retrieve data.

Examples:

- To select all columns for all rows of a table
  ```sql
  select * from z13;
  ```
- To select specific columns of a table
  ```sql
  select Z13_AUTHOR,Z13_TITLE,Z13_YEAR from z13;
  ```

**WHERE CLAUSE**

Limits the select using a variety of operators: (=, < , > , <=, like, etc.)

Example 1:

```sql
select Z13_AUTHOR,Z13_TITLE,Z13_YEAR from z13
where Z13_YEAR = 1888;
```

---

**Z13_AUTHOR**

---**Z13_TITLE**

--- **Z13_YEAR**

---

Twain, Mark, 1835-1910 nnnn
new title
1888

Twain, Mark
Hakelbry fine.
1888

Example 2:

Note the use of a “wildcard” character (%) to indicate that any character(s) can come before or after 1835-1910 in the Z13_AUTHOR column.

```sql
select Z13_YEAR from z13
where Z13_AUTHOR like '%1835-1910%';
```

<table>
<thead>
<tr>
<th>Z13_YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1888</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

**DESCR**

Some column names are complicated and difficult to type. To get a list of all the columns in a particular table, or “describe” the table, use the `descr` command. You can then copy and paste the column names. This command is also useful if you need to know the data type and length of a column.

Example 1:

To get a list of all columns in the z11 table:

```sql
descr z11;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z11_REC_KEY</td>
<td>NOT NULL</td>
<td>CHAR(35)</td>
</tr>
<tr>
<td>Z11_DOC_NUMBER</td>
<td>NOT NULL</td>
<td>CHAR(7)</td>
</tr>
<tr>
<td>Z11_ALPHA</td>
<td></td>
<td>CHAR(1)</td>
</tr>
<tr>
<td>Z11_TEXT</td>
<td></td>
<td>VARCHAR2(40)</td>
</tr>
</tbody>
</table>

**JOINS**

A **join** is the way to link two different tables together using a field common to both. The field does not always have the same name, and just having the same name in both tables is no guarantee that the field can be used for a join.

Ex 1:

To get borrower info. from 2 different tables:
select substr(Z303_NAME,1,20),Z303_BIRTH_DATE,Z304_TELEPHONE
from z303,z304
where Z303_REC_KEY = substr(Z304_REC_KEY,1,12)
and Z303_NAME like '%Binoche%';

<table>
<thead>
<tr>
<th>SUBSTR(Z303_NAME,1,2</th>
<th>Z303_BIRTH_DATE</th>
<th>Z304_TELEPHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binoche, Juliette</td>
<td>19621201</td>
<td>01.34.58.58.66</td>
</tr>
</tbody>
</table>

Note that the join cannot be built on a direct match of Z303_REC_KEY and Z304_REC_KEY. We must use the substr (substring) function to compare only the first 12 digits of Z304_REC_KEY with the whole Z303_REC_KEY. This is the most frustrating part of doing SQL in ALEPH. Similarly named fields are often of different lengths and made up of combinations of different information: IDS, sequence numbers, sublibrary codes, etc.

**Hint:** In addition to learning the `substr` function, always have documentation on the Z tables close at hand to see the component parts of different rec_key fields.

Substring is also VERY useful for clear presentation. Using a previous example, but adding substr:

select substr(Z13_AUTHOR,1,20),substr(Z13_TITLE,1,20),Z13_YEAR
from z13
where Z13_YEAR = 1888;

<table>
<thead>
<tr>
<th>SUBSTR(Z13_AUTHOR,1,20</th>
<th>SUBSTR(Z13_TITLE,1,20</th>
<th>Z13_YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twain, Mark</td>
<td>1835-19</td>
<td>1888</td>
</tr>
<tr>
<td>Twain, Mark</td>
<td>Hakelbry fine.</td>
<td>1888</td>
</tr>
<tr>
<td>Twain, Mark</td>
<td>Hakelbry fine.</td>
<td>1888</td>
</tr>
</tbody>
</table>

**ORDER BY**

You will almost always want to order your select results so they will be easier to understand. To do this, use `order by`.

select substr(Z13_AUTHOR,1,20),substr(Z13_TITLE,1,20)
from z13
where Z13_AUTHOR like 'Al%'

<table>
<thead>
<tr>
<th>SUBSTR(Z13_AUTHOR,1,20</th>
<th>SUBSTR(Z13_TITLE,1,20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alazard, Jean</td>
<td>L'Orient et la peint</td>
</tr>
<tr>
<td>Alazraki, Jaime.</td>
<td>Borges and the Kabb</td>
</tr>
<tr>
<td>Alberto, Rafael, 190</td>
<td>Marinero en tierra.</td>
</tr>
<tr>
<td>Albuquerque, Ruy de.</td>
<td>As represálias; estu</td>
</tr>
<tr>
<td>Alcock, N. W. (Natha</td>
<td>Stoneleigh villagers</td>
</tr>
<tr>
<td>Alger, John Goldwort</td>
<td>Paris in 1789-94.</td>
</tr>
<tr>
<td>Allaby, Michael.</td>
<td>Inventing tomorrow :</td>
</tr>
<tr>
<td>Allen, G. C. (George</td>
<td>British industry and</td>
</tr>
<tr>
<td>Allsopp, Bruce.</td>
<td>The professional art</td>
</tr>
<tr>
<td>Alon, Dafna</td>
<td>Is an Arab-Jewish Re</td>
</tr>
<tr>
<td>Alpert, Carl</td>
<td>Technion the story o</td>
</tr>
</tbody>
</table>
To reverse the alphabetical order of the output, just add **desc** at the end of the **order by**:

```sql
select substr(Z13_AUTHOR,1,20),substr(Z13_TITLE,1,20)
from z13
where Z13_AUTHOR like 'Al%'
order by 1 desc;
```

Note that you can refer to the column in the order by clause **either** by its name **or** by its number in the select line. In this case, 1 refers to Z13_AUTHOR only because that happens to be the first column selected. If you order by something you haven’t selected, you **must** refer to the column by its name. You can also order by as many columns as you want:

```sql
select Z68_SUB_LIBRARY,Z68_ORDER_STATUS,Z68_ORDER_STATUS_DATE
from z68
where Z68_VENDOR_CODE = 'BLACKWELL'
and Z68_ORDER_STATUS_DATE < 20000101
order by 1,2,3;
```

```
+----------+---------------------+---------------------+
| Z68_SUB_LIBRARY | Z68_ORDER_STATUS | Z68_ORDER_STATUS_DATE |
| USMA1      | NEW                | 19990822             |
| USMA1      | NEW                | 19990822             |
| USMA1      | NEW                | 19991226             |
| USMA1      | NEW                | 19991226             |
| USMA1      | RSV                | 19990412             |
| USMA2      | NEW                | 19991226             |
```

And in any order:

```sql
select Z68_SUB_LIBRARY,Z68_ORDER_,Z68_ORDER_STATUS_DATE
from z68
where Z68_VENDOR_CODE = 'BLACKWELL'
and Z68_ORDER_STATUS_DATE < 20000101
order by 3 desc,1,2;
```

```
+----------+---------------------+---------------------+
| Z68_SUB_LIBRARY | Z68_ORDER_ | Z68_ORDER_STATUS_DATE |
| USMA1      | NEW        | 19991226             |
| USMA1      | NEW        | 19991226             |
| USMA2      | NEW        | 19991226             |
| USMA1      | NEW        | 19990822             |
| USMA1      | NEW        | 19990822             |
| USMA1      | RSV        | 19990412             |
```

**AGGREGATES, GROUP BY and HAVING**

In the example above, we limited the output of the select by using where **Z68_VENDOR_CODE = 'BLACKWELL'**. But suppose you just want to know how many orders each vendor has or the latest (or oldest) order for each vendor? This is
done by using aggregate, or “grouping” functions (count, sum, avg, min and max).
The simplest use is to merely count the number of rows in a table:

```
select count(*) from z68;
```

```
COUNT(*)
-------
 334
```

In most cases, aggregates are only meaningful if you are looking at 2 or more columns. In order to use aggregates on more than one column, you must use a group by clause.

Example 1:

To find out how many orders each vendor has in any given status:

```
select Z68_VENDOR_CODE, Z68_ORDER_STATUS, count(*)
from z68
group by Z68_VENDOR_CODE, Z68_ORDER_STATUS;
```

```
Z68_VENDOR_CODE      Z68_ORDER_STATUS COUNT(*)
-------------------- ------------------- -------
300                   CLS                   2
300                   DNB                   3
300                   NEW                   10
300                   RSV                   1
300                   SV                    27
300                   SV+                   1
ABS                    CLS                   2
ABS                    DNB                   2
ABS                    NEW                   24
ABS                    RSV                   16
ABS                    SV                    23
SWETSEDI               NEW                   8
SWETSEDI               SV                    4
THAER                   SV                   1
ZZZ                    SV                    1
ZZZ1                   NEW                   2
ÜBER                   NEW                   1
< Lots more rows >
```

Example 2:

To see how many orders are sent to vendor, or ready to send to vendor, for each vendor in each sublibrary. Note the use of in rather than = to limit the output. In gives you the ability to list more than one value with which to limit the select:

```
select Z68_SUB_LIBRARY, Z68_VENDOR_CODE, sum(Z68_NO_UNITS)
from z68
where Z68_ORDER_STATUS in ('RSV','SV')
group by Z68_SUB_LIBRARY, Z68_VENDOR_CODE;
```

```
Z68_SUB_LIBRARY Z68_VENDOR_CODE SUM(Z68_NO_UNITS)
USMA1            300                       32
USMA1            ABS                       54
USMA1            ACA                       12
```

© Ex Libris Ltd., 2002
ALEPH SQL Tutorial
All Releases
Updated: June 19, 2002
Example 3:

If you’re only interested in seeing vendors with 10 or more units on order for a particular sublibrary, use a `having` clause to limit the groups selected in the previous example:

```
select Z68_SUB_LIBRARY, Z68_VENDOR_CODE, sum(Z68_NO_UNITS)
from z68
where Z68_ORDER_STATUS in ('RSV','SV')
group by Z68_SUB_LIBRARY, Z68_VENDOR_CODE
having sum(Z68_NO_UNITS) >= 10;
```

```
Z68_SUB_LIBRARY Z68_VENDOR_CODE SUM(Z68_NO_UNITS)
---------- -------------------- ----------------- 
USMA1 300                                 32
USMA1 ABS                                 54
USMA1 ACA                                 12
USMA1 ANNELIE                             10
USMA1 BLACKWELL                           21
```

USEFUL FUNCTIONS

|| (concatenation)

This function allows you to join two columns to each other or a column to text when selecting. The concatenated group is treated as a single column when displaying, ordering or grouping.

Example 1:

To add text “Vendor: ” to the value in the column:

```
select Z70_COUNTRY “Country”,
       'Vendor: '
       || substr(Z70_VENDOR_NAME,1,20)||' --
       Contact: '||substr(Z70_VENDOR_CONTACT,1,20) “Vendor Info”
from z70
order by 1;
```

```
Country     Vendor Info
---------- -----------------------------------------
--
```
Hint: When concatenating text, you must always enclose the text in single quotes, e.g., 'Vendor: '

The following functions are very useful for analyzing data:

**rtrim, ltrim**

Trims a character from a column of type string, starting from left or right. Note: These can only be used with columns of type CHAR.

**Usage**: `rtrim(column_to_be_trimmed,'string_to_eliminate')`

Compare this:

```sql
select  distinct Z13_IMPRINT from z13
where Z13_IMPRINT like 'Chicago%';
```

<table>
<thead>
<tr>
<th>Z13_IMPRINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago</td>
</tr>
<tr>
<td>Chicago :</td>
</tr>
<tr>
<td>Chicago,</td>
</tr>
<tr>
<td>Chicago, ;</td>
</tr>
<tr>
<td>Chicago, Ill. :</td>
</tr>
<tr>
<td>Chicago, Ill.</td>
</tr>
</tbody>
</table>

To this:

```sql
select distinct `rtrim(rtrim(rtrim(Z13_IMPRINT,:),'),','),',' from z13
where Z13_IMPRINT like 'Chicago%';
```

<table>
<thead>
<tr>
<th>RTRIM(RTRIM(RTRIM(Z13_IMPRINT,:),''),','),','</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago</td>
</tr>
<tr>
<td>Chicago, Ill.</td>
</tr>
</tbody>
</table>

Colons, spaces and commas are eliminated from the right, in that order. Note that `rtrim` commands can be “nested” (used inside one another).
**rpad, lpad**

Pads a column of type char with a particular character to a designated length. Essentially this is the opposite of **rtrim, ltrim**:

Usage:

```sql
lpad(column_to_be_padded,'string_to_pad_with',what_length_to_pad_to)
```

Compare this:

```sql
select z70_vendor_contact||' --- '||z70_country "Contact/Country"
from z70
where length(rtrim(z70_vendor_contact,' ')) > 0
and length(rtrim(z70_country)) > 0;
```

Contact/Country
-----------------
Ms. Johnson --- USA
Ms. Stein --- England
Teodoro Sacristán --- España
Ms. Fine --- USA
Bernd Zimmermann --- Germany
Mel --- Canada
Mercedes Baquero --- España
Ms. Hacohen --- Israel
Mel --- Canada
John Smith --- England

To this:

```sql
select rpad(z70_vendor_contact,20,' ')||' --- '||z70_country "Contact/Country"
from z70
where length(rtrim(z70_vendor_contact,' ')) > 0
and length(rtrim(z70_country)) > 0;
```

Contact/Country
-----------------
Ms. Johnson         --- USA
Ms. Stein           --- England
Teodoro Sacristán   --- España
Ms. Fine            --- USA
Bernd Zimmermann    --- Germany
Mel                 --- Canada
Mercedes Baquero    --- España

**to_char, to_date, sysdate**

Oracle uses columns of various data types, including number, char (text) and date. In the 2 examples, note the date arithmetic using **sysdate** (today’s date) and the fact that
you must change z68_open_date from a number to a date, before you can change it to a char. Note also that ALEPH dates are type number NOT type date.

Example 1:

To see orders created today:

```sql
select Z68_ORDER_NUMBER, Z68_ORDER_TYPE, Z68_VENDOR_CODE
from z68
where to_char(to_date(Z68_OPEN_DATE, 'YYYYMMDD'), 'YYYYMMDD') = to_char(sysdate, 'YYYYMMDD');
```

Example 2:

To see orders created one week ago today:

```sql
select Z68_ORDER_NUMBER, Z68_ORDER_TYPE, Z68_VENDOR_CODE
from z68
where to_char(to_date(Z68_OPEN_DATE, 'YYYYMMDD'), 'YYYYMMDD') = to_char(sysdate - 7, 'YYYYMMDD');
```

rownum

If you ever want to quickly view just a few rows from a table, use rownum in the where clause. This will save lots of time when looking at big tables. **Hint:** If you use rownum, don’t use an order by clause. It will be ignored.

```sql
select Z20_REC_KEY, Z20_CLAIM_DATE from z20
where rownum < 6;
```

<table>
<thead>
<tr>
<th>Z20_REC_KEY</th>
<th>Z20_CLAIM_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000300010000021</td>
<td>19980609</td>
</tr>
<tr>
<td>000005500020000020</td>
<td>19980609</td>
</tr>
<tr>
<td>000107000001000011</td>
<td>19980616</td>
</tr>
<tr>
<td>000101900058000012</td>
<td>19990110</td>
</tr>
<tr>
<td>000164200001000012</td>
<td>19990106</td>
</tr>
</tbody>
</table>

&variables

The previous example for ORDER BY showed:

```sql
select substr(Z13_AUTHOR,1,20),substr(Z13_TITLE,1,20)
from z13
where Z13_AUTHOR like 'Al'
order by 1 desc;
```

But what if you want to run the same SQL over and over, without rewriting “where Z13_AUTHOR like 'Al%'” each time to cover a different part of the alphabet? You would then use an amper (&) variable. Note that since Z13_AUTHOR is datatype CHAR, it is expecting a text string. So, you need to put the variable in quotes.
select substr(Z13_AUTHOR,1,20),substr(Z13_TITLE,1,20)
from z13
where Z13_AUTHOR like ' &author& %'
order by 1 desc;

When you run the SQL, you are prompted to input the variable

Enter value for author:

If you then enter Au, you get:

<table>
<thead>
<tr>
<th>SUBSTR(Z13_AUTHOR,1,</th>
<th>SUBSTR(Z13_TITLE,1,2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autor, Heinz</td>
<td>Titel of the book</td>
</tr>
<tr>
<td>Author, Tom</td>
<td>Title</td>
</tr>
<tr>
<td>Author</td>
<td>Titel ist notwendig</td>
</tr>
<tr>
<td>Author</td>
<td>History of molecular</td>
</tr>
<tr>
<td>Austen-Leigh, Mary</td>
<td>A Personal aspects of</td>
</tr>
<tr>
<td>Austen, Jane 1775-18</td>
<td>Pride and prejudice</td>
</tr>
<tr>
<td>Auchincloss, Louis.</td>
<td>The house of the pro</td>
</tr>
<tr>
<td>Auchincloss, Louis.</td>
<td>The house of the pro</td>
</tr>
</tbody>
</table>

**EXISTS**

Often, you’ll want to select fields from only one table, but have the values from
another table influence your selection. In other words, you want to select rows only
from table A, for which some conditions exist in table B. This can be very useful.

Note that we don’t care what is actually selected from z305, just that a certain
condition exists in Z305 for those rows related to z303 by the join on
Z30X_REC_KEY. ‘x’ could just as well be ‘y’ or ‘1’ or any of the field names from
z305. Note also that the entire sub-select after exists must be enclosed in single
parentheses.

In the example below, we get a count of all the people in Z303 (Global User
Information) who have 1 of 3 particular borrower statuses in any of the various
sublibraries for which that person is registered in Z305 (Local User Information).

```sql
select count(*) from z303
where exists
  (select 'x'
   from z305
   where substr(Z305_REC_KEY,1,12) = z303_REC_KEY
   and Z305_BOR_STATUS in ('01','06','08'))
```

**UNION clause**

If you want to get data from 2 or more tables as if they were 1 table, use a UNION
clause. Note that you must use the word SELECT on both sides of the union, and that
you must select the same number of columns on both sides. To get the total number of
loans for a sublibrary, we need data from both Loans (Z36) and Loans history (Z36H). We use a UNION clause to do this in a single query.

```sql
select 'Number of loans for sublibrary '||'&&sub_library'||': '||sum(x)
from (select count(*) as x
      from z36
      where z36_sub_library = '&&sub_library'
      and (z36_loan_date >= 19991001 and z36_loan_date <= 20000930)
      UNION
      select count(*) as x
      from z36h
      where z36h_sub_library = '&&sub_library'
      and (z36h_loan_date >= 19991001 and z36h_loan_date<= 20000930)
    );
```

sub_library: USMA1

You enter USMA1, and SQL+ replaces the variable with the value you entered (see &variables section above):

```sql
old   1: select 'Number of loans for sublibrary 
'||'&&sub_library'||': '||sum(x)
new   1: select 'Number of loans for sublibrary '||'USMA1'||': '||sum(x)
old   6: where z36_sub_library = '&&sub_library'
new   6: where z36_sub_library = 'USMA1'
old  12: where z36h_sub_library = '&sub_library'
new  12: where z36h_sub_library = 'USMA1'
```

The final result:

```
'NUMBEROFLOANSFORSUBLIBRARY'||'USMA1'||':'||SUM(X)
----------------------------------------------------
Number of loans for sublibrary USMA1: 723
```

Hint: in order to get a single number, rather than two (this is a count on two tables, after all), we must use `sum(x)` in the first line of the paragraph. Also, use `as x` in the count(*) line in both sides of the UNION.

**spool**

If you want to save the output of a query to a file, use the spool command. This is not part of standard SQL, but rather a feature of Oracle SQL+

Using a previous example:

```sql
SQL> spool save_me
SQL> select substr(Z13_AUTHOR,1,20),substr(Z13_TITLE,1,20)
    from z13
```
where Z13_AUTHOR like '&author%'
order by 1 desc;

   Enter value for author: Au
old  3: where Z13_AUTHOR like '&author%
new  3: where Z13_AUTHOR like 'Au%

SUBSTR(Z13_AUTHOR,1, SUBSTR(Z13_TITLE,1,2
-------------------- --------------------
   Autor, Heinz         Titel of the book
   Author, Tom          Title
<other rows>
SQL> spool off

Leaving SQL+, you will see a new file, with extension .lst created in whatever directory you happen to be in. You can then edit this file in vi.

Warning: Make sure you’re in a “safe” directory such as scratch before spooling in SQL+.

save, @

If you want to save your SQL (but not the output) and re-run the query later, use the save command. This will create a new file, with extension .sql, created in whatever directory you happen to be in. You then use the @ command to run your new file.

Using a previous example:

SQL> select distinct rtrim(rtrim(rtrim(Z13_IMPRINT,':'),' '),',')
from z13
where Z13_IMPRINT like 'Chicago%';

RTRIM(RTRIM(RTRIM(Z13_IMPRINT,:'),',''),')
-------------------------------------------
   Chicago
   Chicago, Ill.
SQL> save chicago
Created file chicago

Now, to rerun your saved script:

SQL> @chicago.sql

RTRIM(RTRIM(RTRIM(Z13_IMPRINT,:'),',')','
-------------------------------------------
   Chicago
   Chicago, Ill.

SQL>

After using save and spool you’ll see 2 new files in the directory you were in when you entered SQL+ at the beginning of your SQL session:
Some specific uses of SQL in ALEPH

You can use SQL to find out how many ADM record numbers have more than 200 items, and how many items each ADM record number has. Note the addition of heading titles by using them in quoted text after the column name:

```sql
select substr(Z30_REC_KEY,1,7) "adm nbr",
       count(substr(Z30_REC_KEY,1,7)) "has this many items"
from z30
group by substr(Z30_REC_KEY,1,7)
having count(substr(Z30_REC_KEY,1,7)) > 200;
```

```
adm nbr has this many items
------- -------------------
0000501                 506
0002217                 452
```

To find out which BIB record numbers have more than 200 attached items, use the following SQL. In order to get column headings, you can use `as` after the column name instead of quotes.

```sql
select Z103_REC_KEY_1 as BIB_NBR
from z103
where substr(Z103_REC_KEY_1,4,2) = '01'
and substr(Z103_REC_KEY,6,7) in (  
select distinct(substr(Z30_REC_KEY,1,7)) from z30
  group by substr(Z30_REC_KEY,1,7)
  having count(substr(Z30_REC_KEY,1,7)) > 200
);
```

```
BIB_NBR
--------
USM010001001
USM010109051
USM010109885
```

To get chars to sort like numbers with letters coming after numbers, use the `translate` function:

```sql
select Z303_REC_KEY
from z303
order by length(rtrim(Z303_REC_KEY,' ')),lpad(rtrim(Z303_REC_KEY,' '),'12',"0"),translate(Z303_REC_KEY,'ABCDEFGHIJKLMNOPQRSTUVWXYZ','99999999999999999999999');
```

To eliminate rows with non-numeric record numbers:

```sql
select to_number(Z303_REC_KEY) from z303
```

Where:
translate(upper(Z303_rec_key),'01234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ','01234567890') = Z303_rec_key

order by to_number(rtrim(Z303_REC_KEY,' '));

To create a script for loading values into a table
(In this case, the currency table Z82):

select 'insert into z82 values ('||''''||z82_currency_name||''''||',''''|| Z82_DATE||''''||','||''''||Z82_RATIO||''''||');' from z82;

'INSERTINTOZ82VALUES('||''''||Z82_CURRENCY_NAME||''''||','||''''||Z82_DATE||''''
---------------------------------------------------------------------
--
insert into z82 values ('USD','19990621','000004084000');
insert into z82 values ('USD','19990625','000004117000');
insert into z82 values ('USD','19990630','000001000000');
insert into z82 values ('ITL','19990718','001790000000');
insert into z82 values ('NIS','19990718','000004000000');
insert into z82 values ('EUR','19990101','000004000000');

You can now use save and @ to store and re-run your script. Or ftp the .sql file to another server and run it there.