

## Appendix A SQL Scripts



Many of the SQL and SQL\*PLUS programs appearing in this book are general enough that they could prove useful in a variety of ad hoc query situations. This appendix collects these programs in one place.

### Utilities

Several programs get used repeatedly by other ad hoc query programs, much like subroutines or procedural calls in structured programming languages. These utilities get executed with the `START` or `@` commands.

Utility programs include those which perform housekeeping functions, date and time stamp reports, create SQL\*PLUS environments suitable for reports intended for the monitor or the printer, define default column headers for reports, and capture standard user prompts for reuse.

#### `clears.sql`

This program (see Figure A-1) clears any preexisting `BREAK`, `COMPUTE`, or `COLUMN` settings. It's a good idea to run `clears.sql` prior to and after each query. In this way you ensure a tabula rasa for each query and you also tidy up after the query to prevent conflicts with subsequent queries.

```
CLEAR BREAKS  
CLEAR COMPUTES  
CLEAR COLUMNS
```

Figure A-1: `Clears.sql`

#### `date.sql`

This date utility (see Figure A-2) retrieves the current system date, formats it as `dd-MON-yyyy`, and captures the formatted value in a user

variable to use when datestamping reports with the TTITLE command.

```
COLUMN today NEW_VALUE xDate
SELECT TO_CHAR(SYSDATE, 'dd-MON-yyyy') "today" FROM DUAL;
```

Figure A-2: Date.sql

### time.sql

This program (see Figure A-3) retrieves SYSDATE, formats the time as hh24:mi, and captures the value in a user variable called xTime that can be used to timestamp a report with the TTITLE command.

```
COLUMN now NEW_VALUE xTime
SELECT TO_CHAR(SYSDATE, 'hh24:mi') "now" FROM DUAL;
```

Figure A-3: Time.sql

### screen.sql

Query writers work in the SQL environment via their monitor screens. While working at the screen or spooling quick reports to the screen, you'll want to customize the SQL environment one way. When spooling reports to an output file, however, you'll usually change several environment settings. This screen utility customizes the SQL environment for work at the monitor. Run it at login and again after the completion of any ad hoc reports spooled to output files that require special formatting.

Screen settings depend largely on individual preferences. The program screen.sql (see Figure A-4) shows one way to customize the screen. It sets the screen size as 24 lines by 80 characters, pauses after each screen in long reports, eliminates top titles, echoes the SQL to the screen, deactivates any forced formfeeds appropriate for printed reports (with NEWPAGE 1), undefines the escape character (\) to use in strings such as path names, and ensures that output spools to the screen.

```
SET PAUSE "ENTER to continue..."
SET PAUSE ON
SET LINESIZE 80
SET PAGESIZE 24
SET NEWPAGE 1
TTITLE OFF
SET ESCAPE OFF
SET ECHO ON
SET TERMOUT ON
```

Figure A-4: Screen.sql

### printer.sql

There are different ways to control the printing of reports. The `SPOOL OUT` command will print a report at your default printer. Using this technique, any special printing instructions must be embedded in the output report, or the printer must first be configured to accommodate the spooled report. More options exist, however, if you first spool to an output file instead of directly to the printer. You can then pass the report file through any number of intermediate programs that allow you to change fonts, point sizes, styles, margins, page layouts, and many other features. You could, for example, open the report file with a word processing program and then execute a macro to set standard report characteristics.

The utility `printer.sql` (see Figure A-5) illustrates this latter technique. It assumes that the report will be printed in landscape mode with an 8-point monospaced font. In Microsoft Word using the MS Line Draw font, this nicely duplicates the standard green-bar printout of 132 characters. Depending on your software, some experimentation with `PAGESIZE` and `LINESIZE` settings may be necessary. `PAGESIZE 65` and `LINESIZE 80` is suitable for portrait reports using 9-point MS Line Draw.

```
SET PAUSE OFF
SET PAGESIZE 52
SET LINESIZE 85
SET ECHO OFF
SET TIMING OFF
SET FEEDBACK ON
SET TTITLE ON
SET DOC OFF
```

Figure A-5: Printer.sql

This program turns off pausing so that the report spools continuously to the report file, sets echoing off so that the SQL does not appear with the report, turns `FEEDBACK` on so that a summary line appears at the end of the report, and ensures that no timing statistics appear in the report.

Many of the programs appearing later in this appendix are formatted for landscape mode with a `LINESIZE` width of 85. This formatting made it easy to import the report files directly into the software that produced this book. For spooling to printers that can accommodate larger line sizes, change some of the `COLUMN` formats in the programs.

### owncol.sql

Query writers tend to concentrate on portions of a database. They work with one or two modules (e.g., a finance or a human resource module),

and within these modules they typically work with only selected tables. Consequently, certain data columns get used repeatedly.

It is helpful to use column headings that are consistent from report to report. You'll also save considerable time because you no longer need to create headings in each query. You simply start a command file to use your default column headings. Then only the adjustments to these defaults must be made. Over time this program will grow in size to include hundreds of data columns you use most frequently. The version shown in Figure A-6 provides only an example of what's possible.

```

/* name */
COLUMN name_id HEADING 'id'
COLUMN name_last HEADING 'last|name'
COLUMN name_first HEADING 'first|name'
COLUMN name_middle HEADING 'middle|name' FORMAT A6
COLUMN name_seriesno HEADING 'name|seqnum' FORMAT 999999

/* address */
COLUMN address_id HEADING 'id'
COLUMN address_type HEADING 'address|type' FORMAT A7
COLUMN address_street HEADING 'street|address'
COLUMN address_city HEADING 'city|address'
COLUMN address_loc HEADING 'state|prov|etc' FORMAT A5
COLUMN address_pcode HEADING 'postal|code'
COLUMN address_country HEADING 'nation|addr|code' FORMAT A7
COLUMN address_seriesno HEADING 'addr|seqnum' FORMAT 9999999

```

Figure A-6: Owncol.sql

### getowner.sql

As you write queries that prompt users for runtime information, you'll find that some prompts get used repeatedly. These can be conveniently captured in SQL programs and then used by issuing a `START` command in your query shell. The program `getowner.sql` (see Figure A-7) prompts a the user for a table owner and then stores the response in the variable called `xOwner`. The `VERIFY` setting is disabled to suppress a listing of the SQL whenever the `xOwner` variable gets used in your program. The same technique can be used for capturing other user responses.

```

SET VERIFY OFF
SET TERMOUT ON
ACCEPT Owner PROMPT 'Enter table owner: '
SET TERMOUT OFF

COLUMN owner NEW_VALUE xOwner NOPRINT
SELECT UPPER('&&Owner') "owner" FROM DUAL;

```

Figure A-7: Getowner.sql

**gettable.sql**

The program `gettable.sql` (see Figure A-8), which is exactly analogous to `getowner.sql`, prompts the user for a table name and then captures the response by defining a user variable called `xTable`.

```
SET VERIFY OFF
SET TERMOUT ON
ACCEPT Table PROMPT 'Enter table name: '
SET TERMOUT OFF

COLUMN tablename NEW_VALUE xTable NOPRINT
SELECT UPPER('&&Table') "tablename" FROM DUAL;
```

**Figure A-8:** `Gettable.sql`

## Login Settings

SQL\*PLUS supports something called a *site profile*, which is an SQL\*PLUS command file generally named `glogin.sql` that allows the database administrator to create a default SQL environment for all users. The default name and location are system dependent.

SQL\*PLUS also supports a *user profile*, which is a command file named `login.sql` that's run after `glogin.sql`. Each time you enter the SQL\*PLUS environment, `login.sql` gets executed from your current directory if it exists or from a system-dependent path if it is not in your current directory. This allows you to customize the SQL environment to your choosing.

The `login.sql` command file shown in Figure A-9 defines a number of path names, establishes a default editor, and executes the screen utility listed above. It is, of course, operating system specific. You'll want to modify the `login.sql` program to reflect your own circumstances.

Defining path names in `login.sql` provides two advantages: (1) you can change the location of utilities without making changes to individual SQL queries that call these programs (by making a single change in the path definition in `login.sql`), and (2) if the defined names are short, they're much more convenient to use at the SQL prompt than longer path names. Modify the `login.sql` shown below to reflect your own preferences. You might, for example, wish to capture the user name and the Oracle instance and then reset the SQL prompt with these values.

```
SET TERMOUT OFF
DEFINE object = d:\orawin95\komenda\objects
DEFINE tool = d:\orawin95\komenda\tools
DEFINE query = d:\orawin95\komenda\query
DEFINE _EDITOR = c:\windows\notepad.exe
START &&object.\screen
SET TERMOUT ON
```

Figure A-9: Login.sql

## Owners

Examining a frequency distribution of data tables listed by their owners provides one overview of an Oracle database. In databases with multiple modules, for example, the modules often will have different table owners that are revealed in the frequency report.

### Shell: owners.sql

Running the program owner.sql (see Figure A-10) produces a frequency report of database tables by their owner. Information for the report is retrieved through a second program (xowners.sql) started by the shell.

```

SET TERMOUT OFF
START &&object.\clears
START &&object.\printer
START &&object.\date
START &&object.\time

TTITLE 'Database tables and views by owner' SKIP 1 -
      LEFT 'Date: ' xDate '      Time: ' xTime -
      RIGHT 'Page: ' FORMAT 999 SQL.PNO SKIP 1 -
      LEFT 'Report name: owners.lis' -
      RIGHT 'SQL name: owners.sql' SKIP 2;

SPOOL &&query.\owners.lis
      start &&tool.\xowners
SPOOL OFF

START &&object.\clears
START &&object.\screen

```

Figure A-10: Owners.sql

### Data program: xowners.sql

The program xowners.sql (see Figure A-11) retrieves data needed for the report. The BREAK and COMPUTE commands produce the grand total that appears in the report.

```

BREAK ON REPORT
COMPUTE SUM OF count ON REPORT
SELECT
      owner "owner",
      count(*) "count"
FROM
      ALL_CATALOG
WHERE
      table_type in ('TABLE','VIEW')
GROUP BY
      owner
ORDER BY 1;

```

Figure A-11: Xowners.sql

## Tables

Lists of data tables that include a brief description of each table's purpose prove very useful for ad hoc query writers.

### Shell: tables.sql

Running table.sql (see Figure A-12) produces a report of all tables for a specified owner. The table owner gets specified by the user at runtime.

```

SET TERMOUT OFF
START &&object.\clears
START &&object.\printer
START &&object.\date
START &&object.\time
START &&object.\getowner
START &&tool.\deftabl

TTITLE 'Tables for owner: ' xOwner SKIP 1 -
      LEFT 'Date: ' xDate '      Time: ' xTime -
      RIGHT 'Page: ' FORMAT 999 SQL.PNO SKIP 1 -
      LEFT 'Report name: ' xReport -
      RIGHT 'SQL name: tables.sql' SKIP 2;

SPOOL &&query.\&&xReport
      START &&tool.\xtables
SPOOL OFF

START &&object.\clears
START &&object.\screen

```

Figure A-12: Tables.sql

### Data program: xtables.sql

The program xtables.sql (see Figure A-13) retrieves data needed for the report. If the comments in the report are blank, it means that the database designer or table owner neglected to include documentation.

```

COLUMN comments FORMAT A40 WORD_WRAP
BREAK ON type SKIP 1
SELECT
      t.table_name "name",
      t.table_type "type",
      c.comments "comments"
FROM
      ALL_TAB_COMMENTS C,
      ALL_CATALOG T
WHERE
      t.owner = '&&xOwner'
      AND c.owner = t.owner
           AND c.table_name = t.table_name
           AND c.table_type = t.table_type
ORDER BY 2, 1;

```

Figure A-13: Xtables.sql

**Defined variables: deftabl.sql**

The shell program runs deftabl.sql (see Figure A-14) to create a user variable called xReport that defines the output report name.

```
COLUMN reportname NEW_VALUE xReport NOPRINT  
SELECT '&&xowner'||'.tab' "reportname" FROM DUAL;
```

**Figure A-14:** Deftabl.sql

## Dictionaries

Data dictionaries list each data column in a table, describe its datatype and characteristics such as length or precision, and describe its meaning.

### Shell: tabldict.sql

Running the program tabldict.sql (see Figure A-15) produces a data dictionary. The table name and owner get specified by the user at runtime.

```

SET TERMOUT OFF
START &&object.\clears
START &&object.\printer
START &&object.\date
START &&object.\time
START &&object.\gettable
START &&object.\getowner
START &&tool.\defdict

TTITLE LEFT xDate ' ' xTime CENTER xType ': ' xTable -
        RIGHT 'Page: ' FORMAT 999 SQL.PNO SKIP 1 -
        LEFT 'Report: ' xReport CENTER 'Owner: ' xOwner -
        RIGHT 'SQL: tabldict.sql' SKIP 1 -
        CENTER xTable_desc SKIP 2 -
        LEFT 'Column Name' COL 22 'Type' COL 32 'Width' -
        COL 39 'Scale Nulls' COL 55 'Column comments' SKIP 1 -
        LEFT xLine SKIP 1;

SPOOL &&query.\&&xReport
START &&tool.\xtabldic
SPOOL OFF

START &&object.\clears
START &&object.\screen

```

Figure A-15: Tabldict.sql

Normally, `HEADING` is enabled; here it is disabled and the `TTITLE` command provides the report headings. This technique proves useful whenever you want specially formatted report headers.

### Data program: xtabldic.sql

The program xtabldic.sql (see Figure A-16) retrieves data used in the dictionary report. Note that if the datatype is `NUMBER`, the dictionary contains the precision of the column. However, if the datatype is `CHAR` or `VARCHAR2`, the length of the data column appears in the report.

### Defined variables: defdict.sql

At runtime the user is prompted for a table name and owner. An output report name is then concatenated from the table name and a file

```

SET HEADING OFF
SET RECSEP OFF
COLUMN name FORMAT A20
COLUMN type FORMAT A8
COLUMN width FORMAT 99999
COLUMN scale FORMAT 99999
COLUMN nulls FORMAT A9
COLUMN comments FORMAT A30 WORD_WRAP
SELECT
    t.column_name "name",
    t.data_type "type",
    DECODE(t.data_type, 'NUMBER', t.data_precision,
           t.data_length) "width",
    t.data_scale "scale",
    DECODE(t.nullable, 'N', 'NOT NULL', 'Y', 'NULL') "nulls",
    c.comments "comments"
FROM ALL_COL_COMMENTS C, ALL_TAB_COLUMNS T
WHERE t.owner = '&&xOwner'
      AND t.table_name = '&&xTable'
      AND c.owner = t.owner
      AND c.table_name = t.table_name
      AND c.column_name = t.column_name
ORDER BY column_id;

```

Figure A-16: Xtabldic.sql

extension (.dic). The table and owner responses are also used to access ALL\_TAB\_COMMENTS to retrieve the table type (view or table) and any table comments to use as documentation in the report title.

The program defdict.sql (see Figure A-17) illustrates a useful technique in which several user variables get defined in a single query. This requires only a small trick in which each DEFINE after the first one is forced to start on a new line (see the NextLine alias). The variable xLine is used in the TTITLE command to separate headers from the report body.

```

/* construct a report name */
COLUMN reportname NEW_VALUE xReport NOPRINT
SELECT '&&xTable'||'.dic' "reportname" FROM DUAL;

/* get several column values from all_tab_comments */
COLUMN tabletype NEW_VALUE xType NOPRINT
COLUMN table_comments NEW_VALUE xTable_desc NOPRINT
SELECT
    table_type "tabletype",
    DECODE(comments, NULL, 'No description available',
           ''''||comments||'''' ) "table_comments"
FROM ALL_TAB_COMMENTS
WHERE owner = UPPER('&&xOwner')
      AND table_name = UPPER('&&xTable');

DEFINE xLine =
'=====
'

```

Figure A-17: Defdict.sql

## Indexes

Indexes are essential for tuning queries that affect system performance. But they're also helpful if you must ensure that only one row gets returned for each item in the report population. Examining unique indexes can help you craft a query to have the intended effect.

### Shell: indexes.sql

The shell program indexes.sql (see Figure A-18) prompts the user for a table owner, uses this response to create an output report name, and then calls the data retrieval program. The report shows all indexes for tables with the specified owner.

```

SET TERMOUT OFF
START &&object.\clears
START &&object.\printer
START &&object.\date
START &&object.\time
START &&object.\getowner
START &&tool.\defindx

TTITLE LEFT 'Indexes on tables owned by: ' xOwner SKIP 1 -
        LEFT 'Date: ' xDate ' Time: ' xTime -
        RIGHT 'Page: ' FORMAT 999 SQL.PNO SKIP 1 -
        LEFT 'Report name: ' xReport -
        RIGHT 'SQL name: indexes.sql' SKIP 2;

SPOOL &&query.\&&xReport
START &&tool.\xindexes
SPOOL OFF

START &&object.\clears
START &&object.\screen

```

Figure A-18: indexes.sql

### Data program: xindexes.sql

The program xindexes.sql (see Figure A-19 on page 325) retrieves data used in the index report. Note that the report identifies the position of each data column included in an index. This item is important. You can inadvertently but dramatically affect query performance by constructing a query so that the leading edge of an index is unknown and unavailable.

### Defined variables: defindx.sql

The shell program runs defindx.sql (see Figure A-20 on page 325) to create a user variable called xReport that defines the output report name. The NEW\_VALUE option in the COLUMN command coordinates with a SELECT against the DUAL table to implicitly define xReport.

```
COLUMN table FORMAT A15
COLUMN index FORMAT A25
COLUMN uniqueness FORMAT A10
COLUMN column FORMAT A25
COLUMN position HEADING 'pos' FORMAT 999
BREAK ON table ON index ON uniqueness SKIP 1
SELECT
  a.table_name "table",
  a.index_name "index",
  b.uniqueness "uniqueness",
  a.column_name "column",
  a.column_position "position"
FROM
  ALL_IND_COLUMNS A,
  ALL_INDEXES B
WHERE
  b.owner = '&&xOwner'
  AND a.index_owner = b.owner
      AND a.index_name = b.index_name
      AND a.table_owner = b.table_owner
      AND a.table_name = b.table_name
ORDER BY 1, 2, 5
;
```

Figure A-19: Xindexes.sql

```
COLUMN reportname NEW_VALUE xReport NOPRINT
SELECT '&&xOwner' || '.idx' "reportname" FROM DUAL;
```

Figure A-20: Defindx.sql

## Constraints

Constraint reports help query writers identify the primary and unique keys in a table, determine if any foreign keys reference primary or unique keys in other tables, and find the conditions enforced on data columns.

### Shell: constrnt.sql

Running `constrnt.sql` (see Figure A-21) produces a constraint report. At runtime the user is prompted for a table name and owner. The report includes the constraint name and type, any conditions enforced by the constraint, the names of other constraints referenced by the constraint, whether the constraint is enabled, and the data columns included in the constraint.

```
SET TERMOUT OFF
START &&object.\clears
START &&object.\printer
START &&object.\date
START &&object.\time
START &&object.\gettable
START &&object.\getowner
START &&tool.\defcnsrt

TTITLE LEFT 'Constraints on ' xOwnerTable SKIP 1 -
        LEFT 'Date: ' xDate ' Time: ' xTime -
        RIGHT 'Page: ' FORMAT 999 SQL.PNO SKIP 1 -
        LEFT 'Report name: ' xReport -
        RIGHT 'SQL name: constrnt.sql' SKIP 2;

SPOOL &&query.\&&xReport
START &&tool.\xconstrn
SPOOL OFF

START &&object.\clears
START &&object.\screen
```

Figure A-21: Constrnt.sql

### Data program: xconstrn.sql

The program `xconstrn.sql` (see Figure A-22 on page 327) retrieves data needed for the constraint report.

### Defined variables: defcnsrt.sql

Two user variables get defined in `defcnsrt.sql` (see Figure A-23 on page 327). The first names the output report by concatenating a file extension (`.cst`) to the table name. The second concatenates the owner and table names to use in the report title.

```
COLUMN cname HEADING 'constraint' FORMAT A26
COLUMN type HEADING 't|y|p|e' FORMAT A1
COLUMN search HEADING 'search' FORMAT A18 WORD_WRAP
COLUMN rconstraint HEADING 'referential|constraint' FORMAT A11
COLUMN stat HEADING 'status' FORMAT A7
COLUMN col HEADING 'column' FORMAT A16
COLUMN pos HEADING 'pos' FORMAT 99
BREAK ON cname ON type ON search ON rconstraint ON stat SKIP 1
SELECT
  a.constraint_name cname,
  a.constraint_type type,
  a.search_condition search,
  a.r_constraint_name rconstraint,
  a.status stat,
  b.column_name col,
  b.position pos
FROM
  ALL_CONS_COLUMNS B,
  ALL_CONSTRAINTS A
WHERE
  a.owner = '&&xOwner'
    AND a.table_name = '&&xTable'
  AND b.owner = a.owner
    AND b.table_name = a.table_name
    AND b.constraint_name = a.constraint_name
ORDER BY 2, 1, 7;
```

**Figure A-22:** Xconstrn.sql

```
COLUMN reportname NEW_VALUE xReport NOPRINT
SELECT '&&xTable' || '.cst' "reportname" FROM DUAL;

COLUMN ownertable NEW_VALUE xOwnerTable NOPRINT
SELECT '&&xOwner' || '.' || '&&xTable' "ownertable" FROM DUAL;
```

**Figure A-23:** Defcnsrt.sql

## Linesize

When space is at a premium in a report, it's important to know precisely what `LINESIZE` is required to accommodate the requested columns. If the columns require more characters than you have available on a page, then you'll need to compress them. The program `linesize.sql` prompts for data columns you intend to use and then computes the `LINESIZE` needed.

### Shell: `linesize.sql`

The shell program `linesize.sql` (see Figure A-24) prompts for data column names, sets a `LONG` display width, retrieves the `DATE` format in effect, and then computes the `LINESIZE` required for the report.

```

SET TERMOUT OFF
START &&object.\clears
START &&object.\screen2
START &&object.\date
START &&object.\time
START &&object.\getcolumn
START &&tool.\defline

SET TERMOUT ON
TTITLE 'Minimum report linesize needed' SKIP 1 -
      LEFT 'Date: ' xDate '      Time: ' xTime SKIP 2;

START &&tool.\xlinesiz
SET TERMOUT OFF

START &&object.\clears
START &&object.\screen

```

Figure A-24: `Linesize.sql`

### Data program: `xlinesiz.sql`

This program (see Figure A-25) computes the `LINESIZE` needed to include all requested data columns in the report.

```

SELECT
  (SUM(DECODE(data_type,
    'NUMBER', DECODE(data_scale,
      0, data_precision + 2,
      data_precision + 3),
    'CHAR', data_length + 1,
    'VARCHAR2', data_length + 1,
    'DATE', TO_NUMBER(&&xDate_length) + 1,
    'LONG', 81,
    data_length + 1
  )) - 1) "Length"
FROM ALL_TAB_COLUMNS
WHERE owner||'.'||table_name||'.'||column_name IN &&ColumnName;

```

Figure A-25: `Xlinesiz.sql`

**Utility program: screen2.sql**

Output from the linesize report is one number showing how many characters you need across a page. The program screen2.sql (see Figure A-26) sets the SQL\*PLUS environment for displaying this number at the screen.

```
SET PAUSE OFF
SET LINESIZE 80
SET PAGESIZE 24
SET NEWPAGE 1
SET ECHO OFF
```

Figure A-26: Screen2.sql

**Defined variables: getcolmn.sql**

To compute report line size, you must specify the data columns by table owner, table name, and data column name. The program getcolmn.sql (see Figure A-27) prompts for these columns and accepts the response.

```
SET VERIFY OFF
SET TERMOUT ON
PROMPT Enter data columns: owner.table_name.column_name
PROMPT Use single quotes around entries and enclose within ().
PROMPT Use uppercase only.
PROMPT
ACCEPT ColumnName PROMPT 'Enter column names: '
SET TERMOUT OFF
```

Figure A-27: Getcolmn.sql

**Defined variables: defline.sql**

Data columns with a LONG datatype display by default at the smaller of the LONG or LONGCHUNKSIZE system variables. The program defline.sql sets each of these system variables to 80 characters. It also retrieves the NLS parameters in effect for the session and determines the display width needed for DATE data columns.

```
-- set LONG and LONGCHUNKSIZE so know display width
-- for LONG datatype
SET LONG 80
SET LONGCHUNKSIZE 80
-- determine length of date format being used
COLUMN datelength NEW_VALUE xDate_length NOPRINT
SELECT
  LTRIM(length(value),' ') "datelength"
FROM NLS_SESSION_PARAMETERS
WHERE parameter = 'NLS_DATE_FORMAT';
```

Figure A-28: Defline.sql

