*Sheffield* *Hallam* *University*



Faculty of Science, Technology and Arts

**SQL** **WORKBOOK**

**55-500998**

**Database** **Systems** **For**

**Software** **Applications**

**SECTION C**

**2020/2021**



*Sheffield* *Hallam* *University*

Faculty of Science, Technology and Arts

**STRUCTURED QUERY LANGUAGE** **(SQL)**

**2020/2021**

SQL Workbook 1 September 2020

SQL Workbook 2 September 2020

Introduction

**INTRODUCTION**

The purpose of this book is to provide practical exercises in the use of SQL to create, populate and maintain a relational database.

**SQL**

SQL (Structured Query Language) is an ISO and ANSI standard *database* *query* *language.* Most relation databases are SQL-compliant, but in spite of SQL being a standard, SQL code is not completely portable among different database management systems.

**Oracle Relational Database**

For this module we will be using Oracle Database 11g Enterprise Edition.

Students will need to create a user account for Oracle, - how to do this is shown in Appendix X. If you have an account from a previous module, it will still be valid – to reset the password or unlock an account, also see Appendix X

**Oracle SQL Developer**

Students will need to connect to their Oracle database account using SQL Developer which is a **free** graphical tool for database development. How to do this is shown in Appendix Y.

With SQL Developer, you can create, browse and manage database objects, execute SQL statements and SQL scripts, and import, manipulate, and export data.

**Sample Tables**

All new accounts have a set of default tables (EMP, DEPT and SALGRADE) already created and populated with data. These are referred to as the **PERSONNEL** **SYSTEM**. Many of the **exercises** throughout this workbook are based on the Personnel System tables.

Some lecture material and various **worked** **examples** in this text are based on part of a simple **BANK** **ACCOUNTING** **SYSTEM** and use the tables CUST, CUSTACC and ACC. These tables represent the fact that a bank customer may have many accounts, and that an account may be held jointly by more than one customer.

Details of these tables can be found in the following pages. If your Personnel System tables get changed/corrupted the instructions to delete and recreate them can be found at the bottom of page 5.

**Workflow**

Topics should be tackled in sequence. This is because exercises in later sections may rely on changes you make to your data or data structures in earlier sections to work properly. Topics must be completed before the scheduled session of the next topic.

SQL Workbook 3 September 2020

Introduction

**The Bank Accounting System**

CUST Owns CUSTACC Allocated ACC

Table: **CUST**

|  |  |  |  |
| --- | --- | --- | --- |
| REFNO | NAME | ADDRESS | AREA |
| A123 A124 B127 B128  C371 | J Doe J Smith R Best J Best  R Done | 1 High Street 2 West Street 4 East Row  4 East Row  23 Middle Avenue | Sheffield Sheffield Rotherham Rotherham  Barnsley |

Table: **CUSTACC**

|  |  |
| --- | --- |
| REFNO | ACCNO |
| A123 A123 B127  B128 | 1245890 1494315 5418490  5418490 |

Table: **ACC**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACCNO | BALANCE | BRANCH | OPENED | BONUS |
| 1245890 1494315  5418490 | 234.50 0.50  1789.40 | Broomhill Tinsley  Broomhill | 12 Nov 2003 15 Dec 1999  6 May 1988 | 100.00 0.00 |

SQL Workbook 4 September 2020

Introduction

**THE PERSONNEL SYSTEM**

Table: **EMP**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
| 7369 7499 7521 7566 7654 7698 7782 7788 7839 7844 7876 7900 7902  7934 | SMITH ALLEN WARD JONES MARTIN BLAKE CLARK SCOTT KING TURNER ADAMS JAMES FORD  MILLER | CLERK SALESMAN SALESMAN MANAGER SALESMAN MANAGER MANAGER ANALYST PRESIDENT SALESMAN CLERK CLERK ANALYST  CLERK | 7902 7698 7698 7839 7698 7839 7839 7566  7698 7788 7698 7566  7782 | 17-DEC-80 20-FEB-81 22-FEB-81 02-APR-81 28-SEP-81 01-MAY-81 09-JUN-81 09-DEC-82 17-NOV-81 08-SEP-81 12-JAN-83 03-DEC-81 03-DEC-81  23-JAN-82 | 800.00 1600.00 1250.00 2975.00 1200.00 2850.00 2450.00 3000.00 5000.00 1500.00 1100.00 950.00 3000.00  1300.00 | 300.00 500.00  1250.00  0.00 | 20 30 30 20 30 30 10 20  30 20 30 20  10 |

Table: **DEPT**

|  |  |  |
| --- | --- | --- |
| DEPTNO | DNAME | LOC |
| 10 20 30 40 | ACCOUNTING RESEARCH SALES OPERATIONS | NEW YORK DALLAS CHICAGO BOSTON |

Table: **SALGRADE**

|  |  |  |
| --- | --- | --- |
| GRADE | LOSAL | HISAL |
| 1 2 3 4  5 | 700.00 1201.00 1401.00 2001.00  3001.00 | 1200.00 1400.00 2000.00 3000.00  9999.00 |

Should data in the tables become corrupt, they may be restored to their original status by issuing each of the following statements for the appropriate table:

**DROP** **TABLE** EMP ;

**CREATE** **TABLE** EMP **AS** **SELECT** \* **FROM** EXAMPLE.EMP ;

SQL Workbook 5 September 2020

Introduction

This page is intentionally blank.

SQL Workbook 6 September 2020

Section C

**Dates, Times & NULLS** **Functions & Grouping Data**

**Using DATE Values**

Oracle stores dates as a number of days, and a date may be used just like any other number column. To match or enter a date the value must be enclosed in single quote marks, dates may be in one of these formats:

dd-mmm-yy dd/mmm/yy dd mmm yy dd-mmm-yyyy dd/mmm/yyyy dd mmm yyyy

eg '01-jan-01' eg '01/jan/01' eg '01 jan 01' eg '01-jan-2001' eg '01/jan/2001' eg '01 jan 2001'

When using two character year numbers, numbers less than 50 will become 20yy, 50 and over will become 19yy. Without formatting, all dates will be displayed in the two-character year format dd-mmm-yy.

Within Oracle, the ‘system date’ i.e. today’s date, is available as a function named **SYSDATE**. This can be used within the WHERE clause of a SELECT statement.

Every SELECT statement must include the SELECT and FROM clauses, but at times there will be no obvious table involved. Oracle provides the special table DUAL to be used in such situations.

To display tomorrow’s date:

To display the current user:

Or to display calculations:

**Examples** **SELECT** SYSDATE+1 **FROM** DUAL ;

**SELECT** USER **FROM** DUAL ;

**SELECT** 6 \* 365 **FROM** DUAL ;

**Exercises**

**C1** Display details of employees recruited before 1st March 1981, in hire date order.

**SELECT \* FROM EMP WHERE HIREDATE < ‘1/MAR/1981’ ORDER BY HIREDATE ASC;**

SQL Workbook 29 September 2020

Section C

**C2** Display details of employees who were recruited during 1983. Do NOT use wildcards.

**SELECT \* FROM EMP WHERE HIREDATE >= ‘01/JAN/1983’ AND HIREDATE < ‘01/JAN/1984’;**

**C3** Display, in chronological order, the name, hire date and job details of employees who have been with the company for more than approximately 29 years (remember that dates are held as a number of days). Ensure the statement will work equally well today and in a month's time. Do NOT pre-calculate any values you use.

**SELECT ENAME, HIREDATE, JOB FROM EMP WHERE (SYSDATE - HIREDATE) > (29\*365) ORDER BY HIREDATE ASC;**

**Storing & Retrieving TIME Values**

Columns defined as DATE types hold both date and time values but normally the time element is ignored and not presented by a SELECT statement. By default the time is set to 12.00 midnight.

To store a specific time in a date column we use the **TO\_DATE** function with a string format made up of those shown in the appendix of this workbook.

**Example** **INSERT** **INTO** ACC (ACCNO, OPENED)

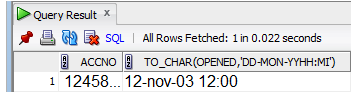
**VALUES** (1245890, TO\_DATE('12-NOV-2003 10:10', 'dd-mon-yy hh:mi') ) ;

To extract the time element we use the **TO\_CHAR** function with a string format made up of those shown in the appendix of this workbook.

**Example** **SELECT** ACCNO, **TO\_CHAR**(OPENED, 'dd-mon-yy hh:mi')

**FROM** ACC

**WHERE** ACCNO = 1245890;

This example will will display:

SQL Workbook 30 September 2020

Section C

**Aggregate Functions**

SQL provides some simple aggregating functions that enable us to derive information about the rows in a table.

**COUNT**(\*)

**COUNT** (col\_name) **MAX**(col\_name) **MIN**(col\_name) **AVG**(col\_name)

**SUM**(col\_name)

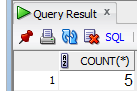
returns the number of rows in the table

returns the number of rows where col\_name is not NULL returns the largest value in this col\_name

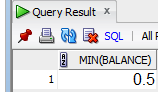
returns the smallest value in this col\_name works out the average value for this col\_name

works out the total value for this col\_name

This example returns a single value, - the number of rows in the CUST table.

**Example** **SELECT** COUNT(\*) **FROM** CUST ;

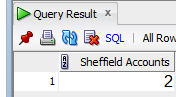
This displays the lowest value held in the BALANCE column from all the rows in the ACC table.

**Example** **SELECT** **MIN(BALANCE)**

**FROM** ACC ;

Any of these can be in an SQL statement that includes constraints.

This example tells us how many customers are based in Sheffield.

**Example** **SELECT** COUNT(\*) AS "Sheffield Accounts" **FROM** CUST

**WHERE** AREA **=** ‘Sheffield' ;

(Note the double quotes needed to delimit a column alias which contains spaces.)

SQL Workbook 31 September 2020

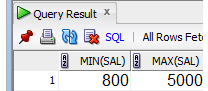
Section C

**Exercises**

For the following exercises you should check your results carefully against the table listings at the front of the workbook.

**C4** What are the lowest and highest basic salaries within the company?

**SELECT MIN(SAL), MAX(SAL) FROM EMP;**



**C5** How many people have a salary greater than £2000?

**SELECT** COUNT(SAL) **FROM** EMP **WHERE SAL > 2000;**

**C6** How many people are there in department 10?

**SELECT COUNT(DEPTNO) FROM EMP WHERE DEPTNO = 10;**

SQL Workbook 32 September 2020

Section C

**NULL Values**

If a column has no value for any particular row, it is referred to as having a NULL value. This is **not** the same as a zero in a numeric column, nor a space in a character column. NULL can be used in the same way as a numeric or character value.

**SELECT** \* **FROM** CUST **WHERE** AREA **IS** **NULL** ;

**SELECT** \* **FROM** CUST **WHERE** AREA **IS** **NOT** **NULL** ;

**INSERT** **INTO** ACC **VALUES** (1234567, 1.00, **NULL**, **NULL**, **NULL**) ;

**Exercises**

**C7** Run the following two statements and identify the difference in the results. Why is this ? (look at the data)

**SELECT** **COUNT(\*)** **FROM** **EMP;**

**SELECT** **COUNT(DEPTNO)** **FROM** **EMP;**

A function that uses (\*) includes all the rows (that satisfy any conditions in the Select statement), whereas a function that uses (column) only includes rows which have non-null values in that column.

**C8** List the employees with no commission recorded in their details.

**SELECT \* FROM EMP WHERE COMM IS NULL;**

Hint: if you check in the EMP table you should see that there are 10 such employees. One employee has a zero commission, but that is a recorded value, and therefore should not be included in the count as it is not NULL.

SQL Workbook 33 September 2020

Section C

**NVL Function**

A further problem is that any calculation that includes a NULL value will always produce a NULL value. Take for example, the expression SAL + COMM.

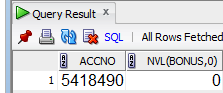
If SAL = 800 and COMM = 0 (zero), this will evaluate to 800+0 = 800; but if COMM is null, it will evaluate to 800 + NULL = NULL – which may be very misleading.

The NVL (NulL Value) function is provided to help avoid this problem. It can be used to convert a null into a specified default value. It will return the actual value of the specified column if it has a real value, or the specified default value if the column is null.

This is often necessary when performing calculations, or formatting for output - since otherwise, null values will be totally ignored by many functions and operations. This may be applied to a column of any data type provided the replacement value is of the same type as the column.

The syntax is:

**NVL**(column, replacement\_value)

**Example** **SELECT** ACCNO, **NVL**(BONUS, 0) **FROM** ACC **WHERE** BONUS **IS** **NULL** ;

**Exercises**

**C9** List the name, job and total income (salary + commission) for all employees. Ensure the total income is shown for all employees.

**SELECT ENAME, JOB, SAL + NVL(COMM, 0) AS INCOME FROM EMP;**

**C10** What are the highest and lowest incomes (including commission) for all employees?

**SELECT MAX(SAL + NVL(COMM, 0)) AS HIGHEST, MIN(SAL + NVL(COMM, 0)) AS LOWEST FROM EMP;**

SQL Workbook 34 September 2020

Section C

**The GROUP BY clause**

The **GROUP** **BY** clause is used in conjunction with the aggregate functions (COUNT, MIN, MAX, etc). It splits the result set into groups based on the values of specified grouping columns, returning one summary row for each group.

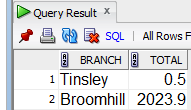
The syntax for the GROUP BY clause is:

SELECT **gpng\_col\_name,** **aggr\_function(**column\_name**)** FROM table\_name

WHERE column\_name operator value

**GROUP** **BY** **gpng\_col\_name;**

The following will list the sum of the account balances for each branch.

**Example** **SELECT** BRANCH, SUM(BALANCE) **AS** TOTAL **FROM** ACC

**GROUP** **BY** BRANCH ;

Any columns specified in the **SELECT** clause which are not arguments of aggregate functions must also appear in the GROUP BY clause.

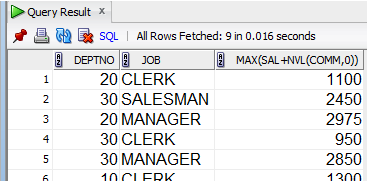
(It is not necessary to have any grouping columns in the SELECT clause, but normally ALL grouping columns should be specified, otherwise the meaning of the output may be misleading.)

We can specify groups as combinations of columns, by specifying multiple columns to group by, as in:

**Example** **SELECT** DEPTNO, JOB, MAX(SAL + NVL(COMM,0))

**FROM** EMP

**GROUP** **BY** DEPTNO, JOB

Note that the groups are not necessarily in a useful order, but that can be addressed using the ORDER BY clause, - see later.

SQL Workbook 35 September 2020

Section C

**Exercises**

**C11** What will be the result of the following query? Before you actually run the code work out the expected result.

**SELECT** **AREA,** **COUNT(\*)** **FROM** **CUST**

**GROUP** **BY** **AREA** **;**

**C12** How many people are there in each department?

**SELECT** DEPTNO, COUNT(DEPTNO) **FROM** EMP **GROUP** **BY DEPTNO;**

**C13** How many people are there in each type of job within each department?

**SELECT DEPTNO, JOB, COUNT(JOB) FROM EMP GROUP BY DEPTNO, JOB;**

**C14** For **each** department, find the average salary and the total salary bill excluding commission.

**SELECT DEPTNO, AVG(SAL), SUM(SAL) FROM EMP GROUP BY DEPTNO;**

**C15** For each department, find the maximum commission earned, and the number of people in that department.

**SELECT DEPTNO, MAX(NVL(COMM, 0)), COUNT(DEPTNO) FROM EMP, GROUP BY DEPTNO;**

SQL Workbook 36 September 2020

Section C

**The HAVING clause**

The **HAVING** clause is used to filter out GROUPS which are not required by applying selection criteria to the results of grouping.

The syntax for the HAVING clause is:

SELECT gpng\_col\_name, gpng\_col\_name, **aggr\_function(column\_name)** FROM table\_name

WHERE column\_name operator value GROUP BY gpng\_col\_name, gpng\_col\_name ;

**HAVING** **aggr\_function(column\_name)** **operator** **value;** ORDER BY gpng\_col\_name, gpng\_col\_name ;

**Example** **SELECT** AREA, COUNT(REFNO) Customers **FROM** CUST

**GROUP** **BY** AREA

**HAVING** COUNT(REFNO) > 1 ;

This is a modified version of exercise C11.

What will the output be ?

**NB:** The WHERE clause restricts which **rows** are input to the full SELECT statement, whereas the HAVING clause restricts which **groups** are shown in the output.

The WHERE clause is processed first to select rows, then the GROUP BY clause calculates the function values for each group, before the HAVING clause determines which groups to retain for output.

The output sequence of the groups may not be predictable, so it is quite valid (and normal) to control the sequence of selected rows by using an ORDER BY clause in addition to a GROUP BY clause. The ORDER BY clause is applied to the result of the GROUP BY and HAVING clauses.

**Exercises**

**C16** Display the department number and number of employees in departments with 5 or more employees.

**SELECT** DEPTNO, COUNT(DEPTNO) **FROM** EMP **GROUP** **BY** DEPTNO **HAVING COUNT(DEPTNO) > 5;**

**C17** Display in descending value, the average salary for those jobs held by two or more people.

**SELECT JOB, AVG(SAL) FROM EMP GROUP BY JOB HAVING COUNT(JOB) >= 2;**

SQL Workbook 37 September 2020