UNIT 1

Database Systems

ORACLE SQL*PLUS

© Sue Stirk

School of Computing & Technology

University of Sunderland, 2002

Version 1 - Created July 2002
Version 2 – Created July 2003
Version 3 – Created February 2006 (Section 10 added © D Nelson)

Contents

1. THE WORKING ENVIRONMENT....1

1.1 INTRODUCTION....1
1.2 OBJECTIVES....2
1.3 GETTING STARTED....2
1.4 THE EDITOR....3
1.4.1 Correcting an SQL Command....3
1.4.2 Listing the Buffer Contents....4
1.4.3 Changing a Character to another Character....4
1.4.4 The 'C' command....5
1.4.5 Running Your SQL statement....5
1.4.6 Inserting a new line - The 'i' command....6
1.4.7 Appending to the end of a line - The 'a' command....6
1.4.8 Deleting a line - The 'del' command....6
1.5 ENTERING SQL COMMANDS THROUGH A TEXT EDITOR....7

2. SELECTING COLUMNS....8

2.1 INTRODUCTION....8
2.2 OBJECTIVES....8
2.3 SIMPLE QUERIES USING THE SELECT STATEMENT....8
2.3.1 Selecting all columns from a table....8
2.3.2 Selecting specific columns from a table....9
2.3.3 Selecting Unique Distinct Values....9
2.4 PERFORMING SIMPLE ARITHMETIC OPERATIONS ON COLUMNS....10
2.4.1 Using the Addition(+) and Subtraction (-) symbols....11
2.4.2 Using the Multiplication (*) and Division (/) symbols....11
Mathematical Precedence....11
EXERCISE 1....13

3. OUTPUT CUSTOMISATION....14

3.1 INTRODUCTION....14
3.2 OBJECTIVES....14
3.3 USING AN ALIAS....14
3.4 CONCATENATION (STRINGING COLUMN NAMES AND/OR LITERALS TOGETHER)....15
3.4.1 The Concatenate Operator....15
3.5 ORDERING ROWS RETRIEVED....16
3.5.1 The Order By Clause....17
EXERCISE 2....18

4. LIMITING ROWS SELECTED....20

4.1 INTRODUCTION....20
4.2 OBJECTIVES....20
4.3 THE WHERE CLAUSE....20
4.4 COMPARISON OPERATORS....20
4.5 LITERALS....21
4.5.1 Using Literals in a where clause....21
4.6 LOGICAI OPERATORS....22
4.7 SELECTING ROWS WITHIN A RANGE OF VALUES....23
USING 'BETWEEN'....23
4.8 SELECTING ROWS FROM A SET OF VALUES....23
4.8.1 Using 'IN'....23
4.9 SELECTING VALUES USING WILDCARDS....24
4.9.1 Using 'LIKE'....24
4.9.2 Use of NOT....24
EXERCISE 3....26

5. SQL FUNCTIONS....27

5.1 INTRODUCTION 27
5.2 OBJECTIVES 27
5.3 NUMERIC FUNCTIONS 27
5.3.1 The round function *27*
Null Values *27*
5.3.2 The nvl function *28*
5.4 STRING HANDLING FUNCTIONS 29
5.4.1 The substr Function *29*
5.5 CHARACTER HANDLING FUNCTIONS 30
5.5.1 The rpad Function *30*
5.5.2 The lower and upper Functions *31*
5.5.3 The initcap Function *32*
5.6 DATE FUNCTIONS 32
5.6.1 The sysdate Function *32*
5.6.2 The add_months Function *33*
5.6.3 The months_between Function *33*
5.7 GROUP FUNCTIONS 35
5.7.1 The avg Function *35*
5.7.2 The max and min Functions *35*
5.7.3 The Count Function *35*
5.7.4 The Sum Function *36*
5.8 GROUPING RESULTS - THE GROUP BY CLAUSE 36
5.9 RESTRICTING GROUPING - USE OF THE HAVING CLAUSE 37
EXERCISE 4 38

6. SELECTING DATA FROM MORE THAN ONE TABLE 39

6.1 INTRODUCTION 39
6.2 OBJECTIVES 39
6.3. EQUI-JOINS 39
6.4 JOINING MORE THAN TWO TABLES 41
EXERCISE 5 43

7. SUB QUERIES 44

7.1 INTRODUCTION 44
7.2 OBJECTIVES 44
7.3 USING A SUBQUERY TO SOLVE A PROBLEM 44
7.4 SUB QUERIES THAT RETURN ONE ROW 45
7.5 SUB QUERIES THAT RETURN MORE THAN ONE ROW 46
7.6 ORDERING DATA WITH SUB QUERIES 46
7.7. EXISTS AND NOT EXISTS 46
EXERCISE 6 48

8. MODIFYING DATA IN TABLES 49

8.1 INTRODUCTION 49
8.2 OBJECTIVES 49
8.3 INSERTING DATA 49
8.4 UPDATING DATA 50
8.5 DELETING DATA 50
EXERCISE 7 51

9. CREATING TABLES 52

9.1 INTRODUCTION 52
9.2 OBJECTIVES 52
9.3 CREATE TABLE COMMAND 52
9.4 DROPPING (DELETING) A TABLE 53
EXERCISE 8 54

9.5 CREATING A VIEW 55

EXERCISE 9 56

APPENDIX 1 61

1. The Working Environment

1.1 Introduction

SQL (Structured Query Language) has emerged from the development of the relational database model and nowadays many database management systems support SQL, running on different hardware platforms. This means that once you have learnt the basic SQL commands, then you can transfer those skills to other databases.

> You have already learnt many of the basic SQL commands in your Level 1 database module. Several parts of the early sections of this Unit will be revision for you. However, Microsoft Access uses many features of SQL which are not standard. This means that Oracle may implement some commands differently from those that you may have learnt already. Where these differences exist they will be pointed out.
>
> **Because of the differences between the MS Access and Oracle implementations of SQL, it is important that you read this guide. Although some of the early sections will be revision for you, DO NOT assume that you know all of this already!**

SQL can be split into two major parts

a) a Data Definition Language (DDL) - used to define database structures

b) a Data Manipulation Language (DML) - used to retrieve and to update data in a database

However, before learning Oracle's implementation of SQL you need to be able to use the Oracle SQL*Plus environment, which supports the development and running of SQL Programs. Part of this environment is the line editor that allows you to enter SQL commands interactively. The reason you need to learn how to use the line editor is that when you start typing in SQL commands you will almost certainly make typing errors. It is essential therefore that you learn to recognise basic errors, and know what to do when you make them. This section will introduce you to this basic working environment for SQL. This environment is quite different to that of MS Access.

1.2 Objectives

When you have completed this section you will be able to:

- Load the SQL*Plus line editor
- Enter SQL commands using the line editor
- Use the basic line editor commands to edit SQL code
- Run SQL commands from a text file

1.3 Getting Started

In order to control access to the database, i.e. to keep out unwanted guests, a system of 'username' and 'password' is used. You will be given separate instructions on what your username and password is, and how to enter SQL*Plus. Once you have successfully entered SQL you will see the SQL prompt:

SQL>

It is from this point that you can type in SQL commands. Whatever machine you work on, as long as you have this prompt you have successfully entered the SQL*Plus environment and can begin work.

Before you start you will need to create some tables and data. To do this you will need to run a file called ***property.sql***. You will be given separate instructions on how to obtain this file. Please note that the tables that will be created are not the same as those that you used in Level 1. To run the file, at the SQL> prompt type:

start property.sql

This will execute the SQL statements in the file and create the tables and data that you need. These tables will be created in your Oracle account, and you will only need to run this file once. You will **not** need to run it again at the start of every session.

1.4 The Editor

In Oracle SQL*Plus a line editor is used to enter SQL commands. The editor is a very simple line editor. All this means is that the editor uses single lines of text. It is NOT like a word processor. We will type lines of SQL code using this editor in order to demonstrate how it works. Do not worry about learning the commands at this stage. Concentrate on understanding how the editor works. When you type in an SQL statement it is stored in a buffer. The contents of this buffer are available until the next statement is input. The line editor uses a number of simple commands that are summarised below.

SUMMARY OF LINE EDITOR COMMANDS

Command	Abbreviation	Action
LIST	l (‘ell’ not ‘one’)	Lists all lines in the SQL buffer to the screen
LIST n	l*n*	Lists the whatever line number has been specified as *n* to the screen
CHANGE	c/old text/new text	Changes text in a line from old text to new text
RUN	r	Runs the SQL statement
INPUT	i	Insert lines of text *after* the current line
APPEND	a	Adds text to the end of a line
DELETE	del	Deletes a complete line

1.4.1 Correcting an SQL Command

In order to demonstrate how the editor works in practice, at the SQL prompt type in the following 3 lines of SQL. Type them exactly as they are shown, on 3 separate lines. This code is deliberately full of errors. Use the return, or enter key to move on to the next line.

```
SQL> select ploperty_no, substr(propty_no,3,2)
  2  where lower(prop_town) = 'sunderlan
..  3  and prop_rent_pm < 450;
```

ERROR:
ORA-01756: quoted string not properly terminated

When you press the return or enter key at the end of this statement, SQL will indicate that there is an error, and an error message will appear. Do not worry about this, as you will correct all of the errors as you proceed through this section. We will use this example to demonstrate each of the editor commands.

1.4.2 Listing the Buffer Contents

The I(ist) command allows you to list one or several lines of code that currently exist within the buffer. When you type I (ell, not one) the full SQL statement is listed on your screen.
Try the following command:

SQL> l

You should now see the full SQL statement that you have just typed in, displayed on your screen.

Notice that the semi-colon that you typed in on the last line is not displayed. This is because this semi-colon is not stored in the buffer. It is used to tell SQL that you have finished typing your SQL statement.

You will see that when you list the SQL statement, each line, except the first, has a line number. To display a specific line you simply type

SQL> l*n*

where *n* is the number of the line you want to display.

There are 4 errors in the SQL code you have just typed:

1) Line 1: the "**l**" in ploperty should be an "**r**"
2) Line 1 the "**er**" In the second property_no on the same line is also missing
3) Line 2: This line is missing completely. It should read "**from prop_for_rent**"
4) Line 3: The "**d'**" is missing from 'Sunderland'

We will now use the facilities of the line editor to correct this code.

1.4.3 Changing a Character to another Character

Here, we want to correct the first error in line one, i.e. change the 'l' in 'ploperty' to an 'r'.

To do this we need to

(1) Find the line that we want to change (line 1)

(2) Change it.

Use the *ln* command to display the line you want.
To actually make the change, you use the '**C**' command.

1.4.4 The 'C' command

The format of this command is

c/the letter(s) or word(s) that you want to change from/the letter(s) or word(s) that you want to change to.

For example:

SQL> c/Fred/Joe

will change Fred to Joe

This is called string replacement.

It is important when you replace characters that you check the line carefully. For example, if you typed in the command

SQL> c/l/r

you might think that this would change the 'l' in property to an ‘r’.

However, look again. If you did this, you would actually change the 'l' in select to serect. This is because SQL changes the first occurrence of the specified letter(s). You must ensure that you type in a letter or sequence of letters or words to uniquely identify what you want to change.

Correct the first and second error on line 1.

1.4.5 Running Your SQL statement

When you first type in an SQL statement, the code will be executed automatically when you press enter after the semi-colon has been typed in. However when you correct a line in your SQL statement you have to tell SQL that you want the whole statement in the buffer to be re-run. The 'r' command allows you to do this. Try it. Type

SQL> r

You will see that SQL shows you that there is an error by placing an asterisk near it. We need to correct the next error before our code will run correctly.

1.4.6 Inserting a new line - The 'i' command

We now need to add the missing line **from prop_for_rent** between lines 1 and 3. There are 4 steps to follow.

Use these steps to insert the missing line.

STEP 1
Go to the line that is before the line to be inserted (in this case line 1)

STEP 2
Type the letter **i** and press the **enter** key

STEP 3
Type in the text of the line or lines you want to insert (use the enter key whenever you want to move on to a new line)
STEP 4
Press the **Enter** key and then the **CTRL & C** keys to terminate the insertion procedure

1.4.7 Appending to the end of a line - The 'a' command

All that is left for you to do is to add the missing **d'** from 'sunderlan. The 'a' command allows you to do this. Use the following steps to do this.

STEP 1
Go to the line that you want to append to using the '**l*n***' command

STEP2
Type the letter **a** followed by a space (DO NOT press the enter key)

STEP3
Type in the text that is missing.

STEP4
Press the **enter** key to terminate the append procedure

1.4.8 Deleting a line - The 'del' command

The del command allows you to delete a line. All you need to do is to display the line you wish to delete by using the '**ln**' command. When the appropriate line is displayed type **del**. The line numbers of the remainder of the command will be automatically re-numbered.
You can also delete individual characters by replacing them with a space. For example, if you type

SQL> c/from prop_for_rent/

then the **from prop_for_rent** will be replaced with a space.

1.5 Entering SQL Commands through a Text Editor

The line editor is useful for entering short commands. However, if you type in long statements then this approach can be somewhat tedious. Statements can also be entered into a text file that you can edit easily without having to type the commands again. You can use any text editor that exists on your system to do this. For example you can use NOTEPAD or WORD if you are using MS Windows, or you can use the Text Editor supplied on the Sun's. However, your text file MUST be saved with the extension **.sql** and it **MUST** be saved as a **TEXT ONLY FILE NOT** as a Word Processing File.

Try this out. Using the text editor, create a new file and type in the statement that you used earlier. Make sure that you type it with all the errors in it. Save the file. Call the file **test.sql**

Now go back to the SQL prompt and run the file. To do this type in

SQL> start test

Because you have already named your file with a **.sql** extension you do not need to include this when you execute the file.

Your file will run and various error messages will be displayed. Now go back and re-open your file. Correct the errors, so that your statement reads

```
select property_no, substr(property_no,3,2)
from prop_for_rent
where lower(prop_town) = 'sunderland'
and prop_rent_pm < 450;
```

Save the file and run it again and the required information should be selected. One row should be selected.

PLEASE NOTE THAT IF YOU HAVE SAVED YOUR FILE IN A SUBDIRECTORY YOU WILL NEED TO GIVE THE FULL PATH NAME.

If your file does not run correctly then check the common errors listed below:

(1) Forgetting to use the extension .sql when you save your file
(2) Naming your file using upper case characters, and then trying to run it using lower case characters.
(3) Typing in the wrong filename.
(4) Forgetting to save the file as a text file.
(5) Typing in the wrong filename or incorrect path name.

2. Selecting columns

2.1 Introduction

SQL makes it easy to retrieve information by use of the **select** statement. SQL, like all languages, be it a programming language or French or German, has a basic grammar, including syntax, punctuation, and vocabulary. The SELECT statement is probably the most commonly used SQL statement. It is used to retrieve or extract data from a database.

> Although you have already met this command in MS Access, it is important to note some differences. Firstly, Oracle does not allow spaces in column names. The convention is to use an underscore instead of a space. This also means that the square brackets around column names are not required in Oracle.

2.2 Objectives

When you have completed this section you will be able to:

- Retrieve data from a table using the SELECT statement in SQL*Plus
- Retrieve distinct values from a table
- Perform arithmetic using the SELECT statement

2.3 Simple Queries Using the Select Statement

2.3.1 Selecting all columns from a table

Please note that a sample case study is provided in the appendix. You will need to refer to this for the table and column names. In Oracle you can also look up the tables and column names on-line.

SQL> **select * from tab;**

Will show you the names of tables that you have access to.

SQL> **describe** ***tablename (where tablename is the actual name of a table)***

Will give you the definition of the table you specify.

A simple form of the **select** statement is to specify which columns you want to retrieve, and which table you want to retrieve them from. To make it even easier, you can retrieve all columns in a table by using an asterisk (*), which just means all columns. An example statement, using the sample database in Appendix I, would therefore be

```
select *
from prop_for_rent;
```

Notice that the statement must be terminated with a semi-colon - just as in English you would end a sentence by using a full stop at the end. The SQL statement can be written using upper or lower case. It is not case sensitive. However, in this unit all statements will be written in lower case. If you typed in this statement you would see the entire table whizzing past you on the screen. Although this statement appears on two lines, it would be equally correct to type the whole statement on one line. This is just a matter of style.

2.3.2 Selecting specific columns from a table

If you don't want to display all of the columns, because you do not need to see all of the information, then you can name the specific columns that you want to see. Thus

```
select prop_town
from prop_for_rent;
```

will display only the town for each row in the table. If you want to retrieve more than one column, you must separate the column names with a comma.

```
select property_no, prop_town
from prop_for_rent;
```

2.3.3 Selecting Unique Distinct Values

When you selected the town from the prop_for_rent table, you may have noticed that the output was a little repetitious. A sample of the output you should have seen is shown overleaf.

```
PROP_TOWN
--------------------
Newcastle
Newcastle
Sunderland
Sunderland
Newcastle
Durham
Tyne and Wear
Durham
Sunderland
Houghton-Le-Spring
Durham
Newcastle

12 rows selected.
```

These are not duplicates because they all have different property numbers. However because only the town has been selected you can't see that. Repetition of names is not checked unless requested. In order to request this, and to display each value only once SQL uses the keyword distinct. Thus to display the towns only once we would use the following statement.

select distinct prop_town
from prop_for_rent;

Notice that there is no comma after the word distinct, there is a space instead. This is because it is not a column name.

2.4 Performing Simple arithmetic Operations on columns

Sometimes when we have numeric values in columns we want to be able to perform arithmetic on them. For example, we may want to calculate the price of a holiday as being the cost of the plane ticket plus the price of the hotel. An estate agent may wish to calculate its deposit. There are clearly many business applications that would require this type of calculation. SQL can be used to perform these simple calculations using the usual arithmetic symbols, i.e.

Operator	Meaning
*	multiply
/	divide
+	add
-	subtract

These are all referred to as arithmetic operators.

2.4.1 Using the Addition(+) and Subtraction (-) symbols

Adding columns together is very straightforward in SQL. Thus

select rent_pm + deposit_amount
from lease;

```
RENT_PM+DEPOSIT_AMOUNT
----------------------
                   700
                   700
                   800
                   550
                   650
                   600
                   550
                   600
                   700
                   700

10 rows selected.
```

will add together the rent per month and the deposit for each row on the table. As you can see, the select statement simply becomes an arithmetic expression.

Subtraction works in exactly the same way as addition.

2.4.2 Using the Multiplication (*) and Division (/) symbols

Again these both work in exactly the same way as addition and subtraction. Thus

select rent_pm * 12
from lease;

will multiply the rent_pm by 12 to give an annual amount.

Mathematical Precedence

When using arithmetic expressions it is important to remember that some calculations are automatically carried out before others. This is called mathematical precedence and just refers to the order, or sequence in which statements are carried out. Multiplication and division are carried out before addition and subtraction. However, you can override this by using brackets. Take the following example. If a local Council wanted to calculate all their tenants' annual rents by adding together the rent per week and their water rates to get a weekly amount to charge, and then to multiply this by the number of weeks in a year, then they would be seriously short of funds if they forgot about mathematical precedence. Consider the following:

select rent_pm + deposit_amount * 12
from lease;

where the value of the rent_pm is £250, and the value of the deposit is £100 then because multiplication is carried out before addition, this would mean that the deposit would be multiplied by 12 and then that value would be added to the rent_pm giving a total of 1450. What this figure should actually be is 7200. Quite a difference!

Exercise 1

Using the property case study found in Appendix 1, carry out the following exercises.

1. Retrieve all of the columns for all tenants.

2. Retrieve the surnames of all tenants

3. Retrieve the surnames and forenames of all tenants

4. Is the statement below correct?

   select from prop_for_rent

   If not, state why not below.

5. The following SQL statement has three syntax errors. Circle the errors and then correct and execute the statement

   ```
   select tenant_Id  t_surname,
   from tenant;
   ```

6. Retrieve all towns that properties are situated in.

7. List the monthly rents for all properties. Ensure that you are not repeating data.

8. Add up the monthly rent, and the deposit amount

9. List the property_no, rent per month and the rent per month increased by 5%. (Use 1.05)

10. List the property_no, rent per month and the rent per month increased by 5%. Also list the annual rent including the poll tax amount.
    Assume that the poll tax is already an annual amount.

3. Output Customisation

3.1 Introduction

As you have seen the output that you see displayed on the screen is not always very meaningful. A column name such as prop_rent_pm is reasonably straightforward, and you could probably guess that this means rent per month. However sometimes column names are less meaningful. In SQL, it is possible to tailor what appears as a column name on the screen to make it more useful. This is achieved by using an alias.

3.2 Objectives

By the end of this section you will be able to:

- Give a field a different name when it is displayed
- Concatenate fields
- Use a literal
- Sort the records retrieved from a table

3.3 Using an Alias

An alias in English simply means an assumed name. It means exactly the same in SQL. It is just a string of characters that is used as the heading for a column or arithmetic expression that is displayed on the screen, i.e. it is an assumed name for the column heading. You saw earlier that when you use a mathematical calculation in a select statement, the column name that appears on the screen is simply the arithmetic expression, e.g. rent_pm + deposit_amount. It would be much better to display this as for example total_paid. In this case total_paid would be the alias. The SQL statement to display this alias would be

select property_no, rent_pm + deposit_amount as total_paid
from lease;

Several column headings can be given an alias in the same select statement. For example:

select property_no, prop_rent_pm as rent, prop_rooms as rooms from prop_for_rent;

In this case ***rent*** is an alias, or alternative name for prop_rent_pm, and ***rooms*** is an alias for prop_rooms.

> In Oracle you do not have to use the ***as*** part of the statement. Instead, you can just leave a space:
>
> **select property_no, prop_rent_pm rent, prop_rooms rooms**
> **from prop_for_rent;**
>
> Either method is correct and you can use whichever you prefer
> In the rest of this guide, the ***as*** will not be used.

3.4 Concatenation (Stringing column names and/or literals together)

Sometimes it is desirable to combine several columns together under one heading when it is displayed. For example you may want to place the surname and forename of a tenant together. Or you may want to place all of the address columns - street, area, town, and postcode - under one heading called address. SQL allows you to do this by using what is called the concatenate operator.

3.4.1 The Concatenate Operator

> In Microsoft Access the concatenate operator is the & (ampersand) symbol. If you use this in Oracle you will have problems.

The concatenate operator in standard SQL is simply the use of the double pipe character ||. This is used to string columns and literals together. For example, if we want to string together street and town with a comma in between, and give it a title of address when displayed, we can use the following statement.

select prop_street ||', '|| prop_town address
from prop_for_rent;

This would appear as

```
ADDRESS
-----------------------------------------------
The Cedars, Newcastle
Belmont Road, Newcastle
Bromsmere Court, Sunderland
Lassiter Drive, Sunderland
Stretton Avenue, Newcastle
Sharp Rise, Durham
The Oaks, Tyne and Wear
Chatham Road, Durham
Prudhoe Road, Sunderland
Charles Drive, Houghton-Le-Spring
Chatham Road, Durham
Stretton Avenue, Newcastle

12 rows selected.
```

We have strung together street and town, placed a comma between them, and given the whole thing an alias called address.

Text can be added to appear on the screen with a column name, by using a literal. For example

select 'Rent per month is', prop_rent_pm
from prop_for_rent;

will display the following output

```
'RENTPERMONTHIS'  PROP_RENT_PM
----------------- ------------
Rent per month is          650
Rent per month is          550
Rent per month is          450
Rent per month is          450
Rent per month is          450
Rent per month is          550
Rent per month is          300
Rent per month is          600
Rent per month is          400
Rent per month is          650
Rent per month is          600
Rent per month is          450

12 rows selected.
```

3.5 Ordering Rows Retrieved

The order in which information appears on the screen can sometimes be very important, and indeed, helpful. For example it is useful to be able to view records in alphabetical order, or to see rents in descending order so that the most expensive is at the top of the list. In SQL it is possible to change the order in which records appear using the order by clause.

3.5.1 The Order By Clause

The order by clause is optional and the words ***asc*** or ***desc*** are used to determine whether the sort will be in ascending or descending order. For example

```
select property_no, prop_street
from prop_for_rent
order by prop_street desc;
```

will place the output in descending street order.

N.B. If you do not add desc or asc to the order by statement, then by default the order will be ascending.

Data can also be sorted using more than one column. When you do this, the first column you name in the **order by** clause means that your data will be sorted first on this column and then within that, on the next column you name. For example:

```
select property_no, prop_street, prop_pcode
from prop_for_rent
order by prop_street desc, property_no asc;
```

will sort the data by ascending property_no within descending street. If you try out this statement you will see the following output.

```
PROPERTY_NO PROP_STREET               PROP_PCO
----------- ------------------------- --------
       1007 The Oaks                  NE8 3HS
       1001 The Cedars                NE76 7YU
       1005 Stretton Avenue           NE8 3LK
       1012 Stretton Avenue           NE8 3LB
       1006 Sharp Rise                DH5 9LT
       1009 Prudhoe Road              SR5 4TB
       1004 Lassiter Drive            SR6 12K
       1008 Chatham Road              DH6 3HT
       1011 Chatham Road              DH6 3HV
       1010 Charles Drive             DH7 6LB
       1003 Bromsmere Court           SR2 5BK
       1002 Belmont Road              NE12 7BY

12 rows selected.
```

You can also use an alias in the order by clause. For example

```
select property_no, prop_street address
from prop_for_rent
order by address desc;
```

will order the output in descending street order.

Exercise 2

In questions I to 4 write down whether the SQL statements are correct. Where they are correct, name the alias. Where they are incorrect, state why.

1. select property_no, prop_rent_pm as rent
   from prop_for_rent;

2. select property_no, prop_rent_pm rent
   from prop_for_rent;

3. select distinct prop_street as address
   from prop_for_rent;

4. select property_no, rent_pm+deposit_amount as total charge
   from lease;

5. Using SQL, output the property_no, prop_rent_pm and the prop_rent_pm increased by 5% for each property. In addition, display the column heading for the existing rent as old_rent and the column heading for the increased rent as new_rent.

6. Using SQL, display the surnames and forenames for each tenant in a single column with a column heading of 'FULLNAME'.

7 Write the SQL which will produce output in the following format

```
Tenant     3003  started the lease for property      1001  on  12-JUN-1995
Tenant     3002  started the lease for property      1002  on  12-JUN-2000
Tenant     3003  started the lease for property      1002  on  12-JUN-1996
Tenant     3004  started the lease for property      1003  on  12-MAR-1994
Tenant     3005  started the lease for property      1004  on  21-MAY-2001
Tenant     3006  started the lease for property      1005  on  30-JUL-1993
Tenant     3007  started the lease for property      1006  on  01-MAY-1997
Tenant     3008  started the lease for property      1005  on  21-APR-1995
Tenant     3003  started the lease for property      1008  on  15-MAR-1996
Tenant     3003  started the lease for property      1010  on  12-JUN-1995
```

8. Write the SQL to produce output in the following format:

   The tenant's surname is Riddell
   The tenant's surname is Peters

   N.B. You will select all surnames and so will have several more lines than are shown here.
   (HINT: To use a literal which has an apostrophe in it you use two single quotes to indicate that this is an apostrophe to be used in the literal, and not just the end of the literal. E.g. 'sue''s age is unknown')

Is the syntax for questions 9 and 10 correct? If not, circle the errors and correct and execute the code.

9.
```
select property_no, prop_street
from prop_for_rent;
order by descending property_no;
```

10.
```
select property_no, prop_pcode as postcode
from prop_for_rent
order postcode desc;
```

11. Write the SQL to display the property_no, the street, the rent_pm and the poll tax in the order ascending rent_pm within ascending poll tax.

12. Write the SQL to display the property_no, the street and town separated by a space and given a title of address, and the total annual rent. Order the output by the first column in descending order

4. Limiting Rows Selected

4.1 Introduction

When you select information from the database sometimes it is desirable to limit the rows that you select. For example if there were 5000 rows of data in the prop_for_rent table and you wanted to look at the details of only one property then you would not want to have to select all 5000 rows. You would only want to retrieve the details of the property you were interested in. You can reduce the number of rows selected by use of the **where** clause.

4.2 Objectives

When you have completed this section you will be able to:

- Retrieve specific records using the WHERE clause
- Retrieve specific records by using comparison and logical operators
- Retrieve specific records in a range of values
- Retrieve values from a set of values
- Retrieve values using wildcard features

4.3 The where clause

The **where** clause is part of the select statement and is optional. The where clause limits the number of rows retrieved according to some criteria. So

```
select *
from prop_for_rent
where prop_rent_pm  =  450;
```

would display only those properties with a monthly rent of £450.

4.4 Comparison Operators

The = symbol used in the above select statement is called a comparison operator and is used in the normal way, i.e. to test if one thing equals another thing. There are other comparison operators which you can use with the where clause.

The full list is

Operator	Meaning
=	is equal to
<	is less than
>	is greater than
<> *or* !=	is not equal to
<=	is less than or equal to
>=	is greater than or equal to

For example you could select all rows which had a rent per month which was more than £450. Thus

```
select *
from prop_for_rent
where prop_rent_pm > 450;
```

4.5 Literals

Apostrophes are used to enclose any character based 'constant', whenever they are used in a command. The apostrophes, or quotes, tell SQL that whatever is inside them is to be displayed EXACTLY, or literally as it appears.

In Microsoft Access you can use either double or single quotes for text literals. In Oracle SQL*Plus only single quotes can be used.

Numeric literals, e.g. 90, should NOT be enclosed in quotes. Dates in Oracle should be enclosed in single quotes and should not be enclosed between hash(#) symbols as used in Access.

4.5.1 Using Literals in a where clause

You can use literals as your criteria for selection. For example

```
select property_no, prop_street, prop_town
from prop_for_rent
where prop_town = 'Sunderland';
```

will retrieve all properties in Sunderland.

4.6 Logical Operators

SQL also allows you to retrieve information from a table based on more than one column at a time. This involves using what are called logical operators to connect the columns that are to be used in the where clause. The logical operators are:

Operator	Meaning
AND	All criteria must be met in order to retrieve rows, i.e. column1 AND column2 AND colurnn3 etc
OR	Only one of the specified criteria must be met, i.e. column 1 OR column2 OR column3 etc
NOT	specified criteria is NOT valid

There can only be one where clause in a select statement, so if you want to combine ***and*** and ***or*** operators, you must use brackets to identify the precedence of selection criteria. This is just the same as using brackets to show which part of a mathematical calculation is carried out first. For example

```
select property_no, prop_town, prop_rent_pm
from prop_for_rent
where prop_town = 'Sunderland'
and prop_rent_pm > 400;
```

will select rows which meet both criteria. However

```
select property_no, prop_town, prop_rent_pm, prop_poll_tax
from prop_for_rent
where prop_town = 'Sunderland'
and (prop_rent_pm >400
or prop_poll_tax >100);
```

will select all properties in Sunderland which either have a monthly rent that exceeds £400 or have a poll tax that exceeds £100. The brackets mean that the OR part of the statement is executed before the AND part of the statement.

4.7 Selecting Rows Within a Range of Values

USING 'BETWEEN'

The where clause can be phrased so that you can select a row within a range of values. For example, if you wanted to select all property numbers that have a rent of between £200 and £500 you could write the statement

```
select property_no, prop_rent_pm
from prop_for_rent
where prop_rent_pm >= 200
and prop_rent_pm <= 500;
```

However, SQL provides the between operator to specify a range. For example

```
select property_no, prop_rent_pm
from prop_for_rent
where prop_rent_pm between 200 and 500;
```

will select properties which have a prop_rent_pm between 200 and 500 inclusive.

4.8 Selecting Rows From a Set of Values

4.8.1 Using 'IN'

Another type of query that can be used is when rows are to be selected from a set of possible values. For example

If the details of tenants with 3 possible surnames are required then the following select statement could be used.

```
select *
from tenant
where t_surname = 'Jackson'
or t_surname = 'Stones'
or t_surname = 'Riddell';
```

An alternative to this is to use the in operator. Thus

```
select *
from tenant
where t_surname in ('Jackson', 'Stones', 'Riddell');
```

4.9 Selecting Values Using Wildcards

4.9.1 Using 'LIKE'

It is useful in any language to be able to retrieve data, even when you do not know the exact value of what you are looking for. For example, you may remember that a person's surname begins with S, but not remember any more than that. If you wanted to select all tenants who had a surname which began with an S, then the SQL statement would be

```
select *
from tenant
where t_surname like 'S%';
```

The wildcard % (percentage sign) tells SQL to match on any sequence of characters that follow the letter S.

Note that the wildcard character used in SQL*Plus is different from that of Microsoft Access. In Access you use an asterisk (*) whereas in Oracle you use the percentage (%) symbol).

You can also use an underscore to mean any single character. For example if you are unsure whether someone's name is Stones or Slones, you could use the following statement

```
select *
from tenant
where t_surname like 'S_ones';
```

The like operator is used whenever you want to match characters using the % or _ characters.

4.9.2 Use of NOT

You can also use **NOT** as the negation of several commands - such as NOT LIKE, NOT BETWEEN etc.

For example

```
select *
from tenant
where t_surname not like 'S%';
```

will retrieve all rows which do not start with an S

Exercise 3

For question I & 2 identify which of the SQL statements are correct. If statements are incorrect, circle the errors, and then correct and execute the statements using SQL.

1.
```
select property_no, prop_street, prop_town
from prop_for-rent
where town = SUNDERLAND;
```

2.
```
select surname
from tenant
where t_max_rent = 300
```

3. Display the streets which are not in Sunderland. AIso include the town in your query so that you can check that it is correct.

4. Display the details of all tenants with the name 'Barrymore'.

5. Display all properties that have a rent per month that is more than £600. (Use the prop_for_rent table)

6. Display all properties in Newcastle which have a rent of more than £350 but which also have a poll tax of more than £200.

7. Display details of leases where the deposit is £100 and the payment method was by Standing Order (Entry on the table will be S)

8. Display details of leases where payments were made by Cheque (Entry will be C) and where the lease started after 21-MAY-1995.

9. Display details of all properties in the East Mickley, Grangetown and Sulgrove areas.

10. Using Not, display all properties which are not in East Mickley or Grangetown.

5. SQL Functions

5.1 Introduction

Many of you will have used a spreadsheet before, and will have used spreadsheet functions such as SUM, AVERAGE, COUNT etc. Functions can be used in SQL statements in a similar way. In SQL, functions can be used to operate on both text and numbers. Only some of the more common functions will be described here. You can use the on-line help to discover more functions for yourself. It is expected that you will do this!

5.2 Objectives

By the end of this section you will be able to:

- Use some string handling functions
- Use some character handling functions
- Use some group functions
- Use some date functions

5.3 Numeric Functions

5.3.1 The round function

The round function shortens a number to a specified number of decimal places. So for example

```
select round(prop_rent_pm + prop_poll_tax,1)
from prop_for_rent;
```

will output the accumulated prop_rent_pm and poll tax to one decimal place. The 1 in this statement indicates the number of decimal places. If this is not specified it will default to the nearest whole number, i.e. no decimal places.

Null Values

Sometimes the value of a column in a table may not be known. For example, a person's first name may not be known, and this may be perfectly legitimate and

allowable. In the property database, for example, some properties may not have an area in their address. In order to allow values to be unknown a column on a database can be set up as NULL when creating the database. A NULL value means that it is allowable to have columns in the database which have no values stored in them. This is not the same as zero or spaces. It simply means not known. There are cases where we would NOT want to allow values to be unknown. For example, we would never want a column which was the primary key to be not known as this would defeat the purpose of having a primary key - if there was no value how could it uniquely identify a row? When columns MUST have a value they have to be defined when creating the database. Such columns are defined as NOT NULL.

5.3.2 The nvl function

If a value has been set up on the database as NULL, then it cannot be used in mathematical calculations. The nvl function enables a null value to be converted. For example, if the deposit_amount on the lease table had been set up as NULL we can ensure that a value is returned using the following statement

```
select property_no, nvl(deposit_amount,0)
from lease;
```

This will mean that if SQL finds a NULL value in the deposit_amount column, it will replace it with a 0 when it is displayed. If the deposit_amount column did have a value in it, then it would simply return that value.
N.B. The lease table has no null values in it, so these examples are for illustration purposes only.

You can also use IS NULL (or IS NOT NULL) as part of select statement. For example

```
select property_no, deposit_amount
from lease
where deposit_amount is not null;
```

would retrieve all properties where the deposit_amount contained a value. Conversely

```
select property_no, deposit_amount
from lease
where deposit_amount is null;
```

would retrieve all properties where the deposit_amount does not contain a value.

5.4 String Handling Functions

A string of characters is simply a sequence of characters, and there are a number of functions which handle strings.

Please note that the string handling function in SQL*Plus is quite different to that of Access. In Access there are three main functions (Left, Mid and Right). In Oracle there is only one function.)

5.4.1 The substr Function

The word **substr** stands for sub-string, so as you might expect it deals with parts of a 'string' of characters. It returns whatever part of the string you specify. An example statement might be

```
select t_surname, substr(t_forenames, 1,1) first_initial
from tenant
order by t_surname;
```

will give the following output

```
T_SURNAME                 F
------------------------- -
Barrymore                 H
Carling                   B
Jackson                   J
Karping                   A
Peter                     M
Riddell                   J
Stones                    J
Stones                    J

8 rows selected.
```

Let us look at this function in more detail by breaking down the statement

substr(t_forename,1 ,1)

There are three items inside the brackets, these are called arguments:

a) The first argument is the name of the column that you want the **substr** function to operate on - in this case forename.

b) The second argument is the start position of the string you want to be output - in this case the first number 1 tells SQL that you want to start with the first character of the forename.

c) The third argument is the length of the string. Again, in this example, the length is 1. The full SQL statement above therefore displays the surname and the first character of the forename. If the third argument had been a 4, then it would have output the first 4 characters of the forename.

Notice as well that the alias first_initial cannot be fully displayed because the column is only one character wide. Therefore only the first character is shown (F).

COMMON ERRORS

Typing in the third argument as the end position of the substring within the original string is a very common error. For example, the property_no is defined on the table as 5 characters long. If you wanted to select the last two characters of the property_no, you would use the statement

select substr(property_no,4,2)
from prop_for_rent;

So if a property_no was 10012, this would output 12

The mistake that is made is to write the statement like this
select substr(property_no,3,5) from property;

The 5 is used in error because 5 is the last character position of the original string, i.e. it is the total length of the property_no. This statement would mean that SQL would start at position 3, and then count 5 from that position. As there are only 2 characters left in the string, this would be incorrect. What we want to do is select the length of the substring.

Remember, the numbers are (a) start position of the sub-string (b) length of the substring.

BRAIN TEASER

N.B. There is a further problem with using a sub-string function on the property_no column, due to the way the property_no has been defined. Try to spot the problem. If you can't, then ask your tutor!

5.5 Character Handling Functions

5.5.1 The rpad Function

The word rpad stands for right pad, and this function does just that. It pads out the right end of a string of characters, with whatever you specify, up to a particular length.

For example

select rpad(t_surname, 30, '*') padded_surname

from tenant
order by t_surname;

will set the length of the surname to 30 characters and pad out any blank characters with asterisks. It will also place the output in surname order. The output would look like this:

```
PADDED_SURNAME
------------------------------
Barrymore**********************
Carling************************
Jackson************************
Karping************************
Peter**************************
Riddell************************
Stones*************************
Stones*************************

8 rows selected.
```

5.5.2 The lower and upper Functions

These functions force the whole of a string of characters to be displayed as either lower or upper case as appropriate. So

select lower(t_surname)
from tenant;

will display the surname in lower case characters, no matter how it appears in the database.

```
LOWER(T_SURNAME)
-------------------------
riddell
peter
jackson
stones
karping
stones
barrymore
carling

8 rows selected.
```

The 'upper' function will display the output in capital letters. Thus

select upper(t_surname)
from tenant;

will give the following output:

```
UPPER(T_SURNAME)
-------------------------
```

```
RIDDELL
PETER
JACKSON
STONES
KARPING
STONES
BARRYMORE
CARLING
```

5.5.3 The initcap Function

This works very much like the upper and lower functions, but it forces the first letter of each word to be displayed as upper case.

select initcap(t_surname)
from tenant;

```
INITCAP(T_SURNAME)
-------------------------
Riddell
Peter
Jackson
Stones
Karping
Stones
Barrymore
Carling
```

5.6 Date Functions

The ability to manipulate dates can often be very important in a commercial environment. For example you can compare today's date with date columns to calculate overdue payments, or you can subtract one date from another to find out how many months there are in between them - useful for clients requiring a visit at six monthly intervals. SQL provides several functions to manipulate dates.

5.6.1 The sysdate Function

The word sysdate stands for system date. This function allows you to compare a date held in a table with the system date (which is usually today).

This is the equivalent function to ‘Now’ in Access.

For example

select *
from lease

where enddate = sysdate;

would display all leases that are due to end today (if any)

5.6.2 The add_months Function

This function adds a specified number of months to a date. For example

select stdate, add_months(stdate, 6) visit_due
from lease;

will add 6 months on to the start date of the lease agreement. The output would be as follows:

```
STDATE      VISIT_DUE
----------- -----------
12-JUN-1995 12-DEC-1995
12-JUN-2000 12-DEC-2000
12-JUN-1996 12-DEC-1996
12-MAR-1994 12-SEP-1994
21-MAY-2001 21-NOV-2001
30-JUL-1993 30-JAN-1994
01-MAY-1997 01-NOV-1997
21-APR-1995 21-OCT-1995
15-MAR-1996 15-SEP-1996
12-JUN-1995 12-DEC-1995

10 rows selected.
```

You can see from this though that some of the dates are not actually very useful because they include dates in the past. To ensure that you only display visit dates that are relevant you could add to the query:

select stdate, add_months(stdate, 6) visit_due
from lease
where add_months(stdate,6) > sysdate;

```
STDATE      VISIT_DUE
----------- -----------
21-MAY-2001 21-NOV-2001
```

N.B. If you run this query then your output will be different, since it will depend on the day on which you run the query as time always moves on!

5.6.3 The months_between Function

The months_between function will display the number of months between two dates. In the example below, the time that has elapsed between the start date of the lease

and today will be displayed in months. The output will also be rounded to the nearest whole number.

select property_no, round(months_between(sysdate,stdate)) from lease;

```
PROPERTY_NO ROUND(MONTHS_BETWEEN(SYSDATE,STDATE))
----------- -------------------------------------
       1001                                    71
       1002                                    11
       1002                                    59
       1003                                    86
       1004                                    -1
       1005                                    93
       1006                                    48
       1005                                    72
       1008                                    62
       1010                                    71

10 rows selected.
```

You can see from the data that one start date has a negative figure. This is because one start date is actually AFTER todays date (i.e. the date on which this query was run). If you run this query for yourself you will, of course, find a different number of months displayed because you are running your query on a different date!

5.7 Group Functions

When you select information from the tables in the database you normally retrieve several rows. However, sometimes it is useful to be able to show a single value such as a total value for all rows. For example, when you add up the value of a column in a spreadsheet, the result is just one value, i.e. the total. There are a number of group functions in SQL which allow you to do this.

5.7.1 The avg Function

This enables you to calculate an average. Thus

select avg(rent_pm)
from lease;

will display the average amount paid by tenants as per the following example.

```
AVG(RENT_PM)
------------
         540
```

5.7.2 The max and min Functions

These functions are probably self-evident

select max(rent_pm)
from lease;

will display a single value for the maximum rent.

```
MAX(RENT_PM)
------------
         600
```

Using min will produce the opposite result.

select min(rent_pm)
from lease;

```
MIN(RENT_PM)
------------
         450
```

5.7.3 The Count Function

The count function is very useful if you want to know how many rows exist. The format of this is as follows:

select count(property_no)
from prop_for_rent;

This would count the number of properties held on the prop_for_rent table.

```
COUNT(PROPERTY_NO)
------------------
                12
```

5.7.4 The Sum Function

The sum function is used to give a total figure. For example

select sum(prop_poll_tax)
from prop_for_rent;

```
SUM(PROP_POLL_TAX)
------------------
           3560.12
```

5.8 Grouping results - the GROUP BY clause

The **group by** clause enables the rows retrieved using a group function to be divided into smaller groups. For example, if you calculated the average, minimum and maximum salaries this would look like the kind of totals that you might have at the end of a report. However, it is often useful to have sub totals. The **group by** clause is used to do this. There are one or two rules for the use of the **group by** clause:

- When the group by clause is used, each item in the select statement must be single valued per group.
- Columns used in the select statement must appear in the group by clause, unless they are only being used in a group function.

For example, to find the number of staff working in each branch and the total of their salaries you would use

select branch_no, count(staff_no), sum(staff_salary)
from staff
group by branch_no;

This would give the following output:

```
BRANCH_NO COUNT(STAFF_NO) SUM(STAFF_SALARY)
--------- --------------- -----------------
```

101	2	37350
102	2	32500
103	1	35000
104	2	34750

Without the group by clause you would retrieve only a single value with the group functions and so you would not be able to list the branch number. If you tried to do this you would obtain an error message:

select branch_no, count(staff_no), sum(staff_salary)
from staff;

select branch_no, count(staff_no), sum(staff_salary)
*****
ERROR at line 1:
ORA-00937: not a single-group group function

5.9 Restricting Grouping - use of the HAVING clause

The having clause is designed for use with the **group by** clause to restrict the groups that appear. This is very similar to the **where** clause, except that the where clause filters individual rows, whereas the **having** clause filters groups. N.B. The where clause cannot be used with group functions. If you wanted to display the number of staff, and the average salaries of staff for each branch with more than one member of staff working there you could use the following query:

select branch_no, count(staff_no), avg(staff_salary)
from staff
group by branch_no
having count(staff_no) > 1;

```
BRANCH_NO COUNT(STAFF_NO) AVG(STAFF_SALARY)
--------- --------------- -----------------
      101               2             18675
      102               2             16250
      104               2             17375
```

Exercise 4

1. Output the property number and the start and end dates for those tenants currently still occupying a property.

2. Output the property number, lease start and end dates for those tenants who have vacated a property.

3. Output the average poll tax amount.

4. The last two characters of the property_no column provide the property number for a particular property. Output this number and the street, town and postcode of all properties. Order the output in descending property number within ascending street order.

5. Display the property number (as per question 4), street, town, and postcode where the street name begins with a P.

6. Add together the rent per month and the deposit amount rounded to two decimal places for each leased property.

7. Using the lease table, display the total amount of rent, the number of rents and the average of those rents.

8 Display all tenants who prefer a flat and who do not wish to pay more than £400 in rent. Output the name of the tenant in capital letters.

9 How many rows are there in the branch table?

10. Display the branch number, and the number of properties dealt with by each branch.

6. Selecting Data from More Than One Table

6.1 Introduction

You have already been introduced to the concept of primary and foreign keys in your Level 1 database Module. You will use that knowledge in this section to learn more about joining tables together.

So far, all of the select statements that you have been using, have retrieved data from only one table. Often it is necessary to select information from more than one table. In order to do this the tables have to be joined. The SQL join operation combines information from two tables by forming pairs of related rows from the two tables. The pairs that are formed are those where the matching columns in each table have the same values. For example in our case study, the branch_no in the staff table has the same values as the branch_no in the branch table. To perform a join we simply include more than one table in the **from** clause and use the **where** clause to specify the matching columns.

6.2 Objectives

By the end of this section you will be able to:

- Write an SQL statement to join two tables
- Write an SQL statement to join more than two tables

6.3. Equi-joins

The most popular form of joining tables is by joining them according to two values in two different tables that are the same as each other, This is called an equi-join in standard SQL - because the two values are equal.

Note that the Access syntax using INNER JOIN does not exist in Oracle. Oracle uses much more standard SQL than Access.

For example, if we want to list the branch_no, branch_town, staff_no, and staff_surname for all branches, a join is required. If you look carefully at the case study you will see that this is because some of these columns are in one table and some are in another. The name of staff for example, is in the staff table, whereas the branch town is in the branch table.

BRANCH

BRANCH_NO	BR_TOWN
101	Sunderland
102	Newcastle
103	Whitburn
104	Durham

Primary Key

STAFF

BRANCH_NO	STAFF_NO	STAFF_SURNAME
101	201	Stewart
101	202	Jones
102	203	Murphy
102	204	Farrell
103	205	Kilburn
104	206	Shaw
104	206	Sheldon

Foreign Key

The SQL to perform this join would be

```
select branch.branch_no, br_town, staff_no, staff_surname
from branch, staff
where branch.branch_no = staff.branch_no;
```

```
BRANCH_NO BR_TOWN               STAFF_NO STAFF_SURNAME
--------- -------------------- --------- -------------------------
      101 Sunderland                 201 Stewart
      101 Sunderland                 202 Jones
      102 Newcastle                  203 Murphy
      102 Newcastle                  204 Farrell
      103 Whitburn                   205 Kilburn
      104 Durham                     206 Shaw
      104 Durham                     207 Sheldon

7 rows selected.
```

If you examine this statement you will see that we have selected from two tables - the branch table and the staff table. In the where clause, we have prefixed the name of the columns with the name of the tables. It has also been necessary to qualify the branch_no in the select clause with the table name. The branch_no could have come from either table because the column name is the same.

If you try to select a column which has the same name in more than one of the tables that you are selecting from you will see the following error.

ORA-00918: column ambiguously defined

You can also use aliases for the table name - which can give you less typing to do. The same query could have been written:

```
select b.branch_no, br_town, staff_no, staff_surname
from branch b, staff s
where b.branch_no = s.branch_no;
```

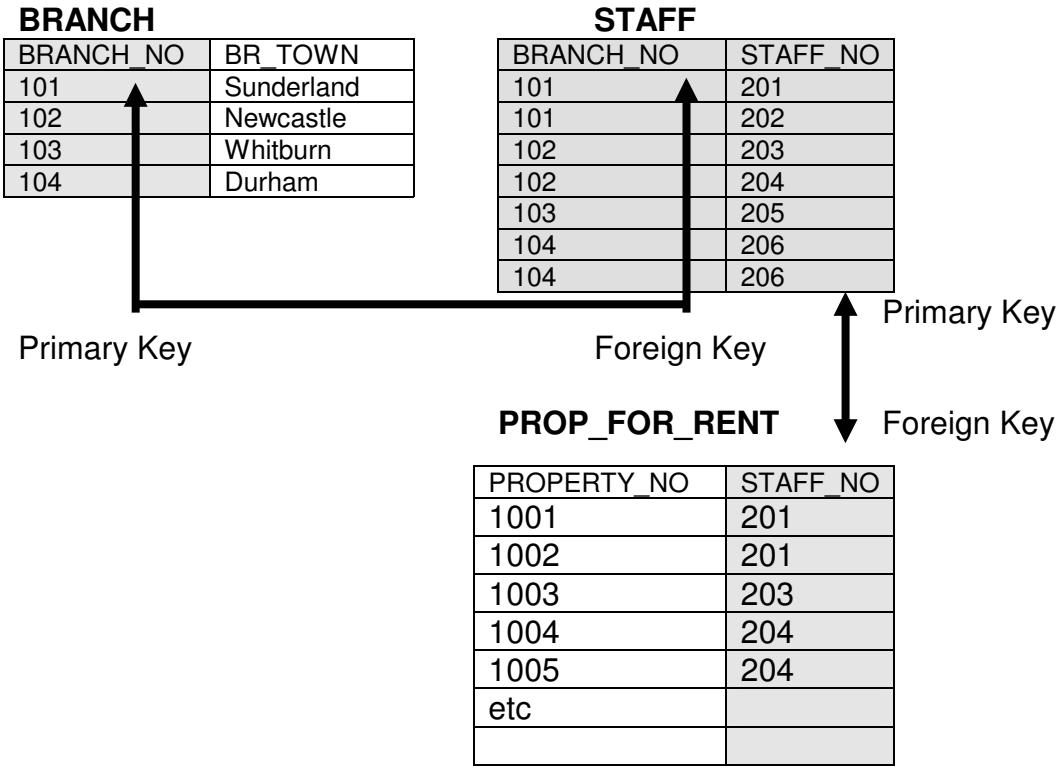
The table branch has been given an alias of **b** and the staff table has been given an alias of **s**. The alias can then be used to qualify a column that may be ambiguously defined. If an alias is used then it must be used everywhere the table name would be used.

COMMON ERROR

It is common to forget to define the abbreviation in the from clause and then try to use it in the where clause. The abbreviation b.branch_no in the select statement is only necessary if you are selecting a column which has the same name in another table that you are selecting from. A common error is to forget to do this when it is necessary.

6.4 Joining More Than Two Tables

Joins can be used to select information from many tables at the same time, making it possible to perform very complex queries. The AND and OR logical operators are used to do this. For example if we wanted to display for each branch the staff who manage properties, including the town in which the branch is located and the properties that they manage we would need to retrieve information from the branch table, the staff table and the prop_for_rent table. Thus



select b.branch_no, br_town, s.staff_no, property_no
from branch b, staff s, prop_for_rent p
where b.branch_no = s.branch_no and
s.staff_no = p.staff_no;

```
BRANCH_NO BR_TOWN               STAFF_NO PROPERTY_NO
--------- -------------------- --------- -----------
      101 Sunderland                 201        1001
      101 Sunderland                 201        1002
      102 Newcastle                  203        1003
      102 Newcastle                  204        1004
      102 Newcastle                  204        1005
      104 Durham                     206        1006
      103 Whitburn                   205        1007
      104 Durham                     206        1008
      104 Durham                     206        1009
      102 Newcastle                  204        1010
      104 Durham                     206        1011
      102 Newcastle                  204        1012

12 rows selected.
```

The branch and staff details are joined using the branch_no (where b.branch_no = s.branch_no) and the staff and prop_for_rent tables are joined using the staff_no (s.staff_no = p.staff_no) to link staff to the properties they manage.

Cartesian Product

If you join tables incorrectly, or if you leave out the join condition completely, you will display all combinations of rows. In the above example if you had left out the where clause which joins the tables you would have displayed 4 (branches) x 7 (staff) x 12 (properties), i.e. 336 rows. In a large database this would take hours to perform the query and possibly result in millions of (incorrect) rows being retrieved. This is called a cartesian product and is best avoided!

Exercise 5

1. Display the tenant name, and property address for all leases where the tenancy start date is between 1st January 1995 and 1st January 1997 inclusive.

2. Display the surname and initial of owners of properties that are currently the subject of a lease agreement, i.e. where a tenant still occupies the property.

3. Display the property number, and inspection date, of all properties which have been inspected. Also show the name of the member of staff who did the inspection.

7. Sub Queries

7.1 Introduction

Some SQL statements can have a complete select statement embedded in them. The results of this inner select are used within the outer statement to determine the final result. This is called a sub query. A subquery is simply a SELECT statement that is nested within another SELECT statement. This section will show you how to write sub queries.

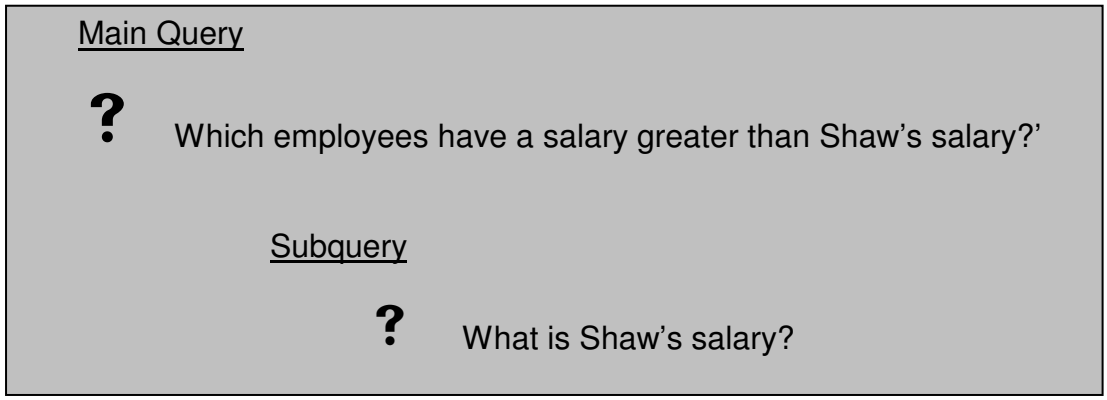
7.2 Objectives

At the end of this section you will be able to:

- Describe the types of problem that sub queries can solve
- Write single-row subqueries
- Write multiple row sub queries

7.3 Using a Subquery to solve a problem

Suppose you want to know who has a salary greater than Shaw. To solve this type of problem you need two queries, one query to find out how much Shaw earns, and another to find out who earns more than that amount. You can solve this problem by combining two queries, one inside the other.



The format of the query would be:

```
select column 1, column2
from table
where column =
      (select column
```

```
        from table
        where condition);
```

The sub query is sometimes called a sub-select. Usually, the sub-select is executed BEFORE the main outer select, because the answer to the sub-select is needed for the outer one.

7.4 Sub queries that return one row

For example, to display the staff who work in the Newcastle branch you would use the following query

```
select staff_no,staff_forenames, staff_surname
from staff
where branch_no =
        (select branch_no
        from branch
        where br_town = 'Newcastle');
```

STAFF_NO	STAFF_FORENAMES	STAFF_SURNAME
203	Tina	Murphy
204	George	Farrell

The inner select statement (select branch_no from branch) finds the branch number that corresponds to the branch with the town 'Newcastle'. Having obtained this branch number, the outer select statement then retrieves the details of all staff who work at this branch. In other words, the inner select statement creates a temporary table that contains the branch_no (102) so that the outer select statement becomes

```
select staff_no,staff_forenames, staff_surname
from staff
where branch_no = 102;
```

The important difference between a simple query and a sub query is that the value of the inner query is NOT KNOWN until the query is run - If you did know it then you would use a simple query.

In the above example, the branch number '102' is a single value, i.e. there is only one branch in Newcastle. The sub query that returned the value of 102 is called a single row sub query. When a sub query returns only one row, a single row operator should be used,e.g, =, >, . If there had been more than one branch in Newcastle you would have obtained an error message saying that the single row sub query returns more than one row.

7.5 Sub queries that return more than one row

The following query will display staff with the lowest salary in each branch

```
select staff_surname, staff_forenames, staff_salary, branch_no
from staff s
where staff_salary in
      (select min(staff_salary)
      from staff
      where branch_no = s.branch_no);
```

```
STAFF_FORENAMES           STAFF_SURNAME              STAFF_SALARY BRANCH_NO
------------------------- -------------------------- ------------ ---------
Stephanie                 Sheldon                           12300       104
John                      Stewart                           12350       101
George                    Farrell                           14500       102
Michael                   Kilburn                           35000       103
```

7.6 Ordering Data with Sub queries

When you use a sub query you cannot use the ORDER BY clause within the inner select statement. If order by is used it must appear as the last statement in the main (outer) SELECT statement. The statement would then look like this:

```
select staff_no,staff_forenames, staff_surname
from staff
where branch_no =
      (select branch_no
      from branch
      where br_town = 'Newcastle')
 order by staff_no desc;
```

```
STAFF_NO  STAFF_FORENAMES            STAFF_SURNAME
--------- ------------------------- --------------------------
      204 George                    Farrell
      203 Tina                      Murphy
```

7.7. EXISTS and NOT EXISTS

Exists and not exists are designed for use with sub queries. They simply produce a true or false result. For example if we wanted to find all staff who work in the Sunderland branch we could use

```
select staff_no, staff_forenames, staff_surname
from staff s
where exists
      (select *
      from branch b
      where s.branch_no = b.branch_no and
      br_town = 'Sunderland');
```

```
STAFF_NO STAFF_FORENAMES            STAFF_SURNAME
-------- -------------------------- --------------------------
     201 John                        Stewart
     202 Janice                      Jones
```

The same query could have been written using a join.

```
select staff_no, staff_forenames, staff_surname
from staff s, branch b
where s.branch_no = b.branch_no and
br_town = 'Sunderland';
```

What does this tell you about the use of the subquery?

Exercise 6

1. Find the member of staff who earns the lowest salary (the lowest salary is unknown).

2. Find the employees who earn the maximum salary in their branch. Display the result in ascending order of salary.

3. Find the members of staff who work in the same branch as Tina Murphy.

4. Show the following details for any member of staff who earns a salary less than the average for their branch.

STAFF_SURNAME	STAFF_FORENAMES	STAFF_SALARY	BRANCH_NO
Stewart	John	12350	101
Farrell	George	14500	102
Sheldon	Stephanie	12300	104

5. Display the following information for the branch with the highest salary bill.

BRANCH_NO	SUM(STAFF_SALARY)
101	37350

8. Modifying data in tables

8.1 Introduction

There are three commands that allow data in tables to be modified. These are:-

INSERT
UPDATE
DELETE

In this section you will learn how to update tables usiing SQL statements.

8.2 Objectives

By the end of this section you will be able to:

- Insert data into an existing table
- Update data in an existing table
- Delete data from an existing table

8.3 Inserting data

Data can be inserted into a new empty table, or into a table which already has data in it. There are one or two rules though.

a) There must be an insert statement for every row
b) The data type of the values must correspond to the data type of the column.
c) The values that you insert are in the order that the columns appear on the table.

An example of the insert statement is

insert into branch
values (105,'25 Sunny Street',NULL,'Sunnyside','SR4 5TH', '0191 5362528');

insert into branch
values (106,'121 Dunhelm Road',NULL,'Durham','DH8 5SH', '0191 2332512');

8.4 Updating data

The values in a table can be updated using the SQL update statement

An example statement would be

update prop_for_rent
set prop_rent_pm = prop_rent_pm * 1.1
where lower(prop_town) = 'newcastle';

When using the update statement the name of the table comes before the names of the columns. The where part of the statement identifies the rows that are to be updated, and the set part of the statement identifies the column(s) to be changed.

8.5 Deleting Data

You can either delete all rows or a single row from a table. For example, the command

delete from prop_for_rent
where property_no = 1008;

would delete the details for this property_no. If there was no **where** clause, and we had just stated

delete from prop_for_rent;

This would completely empty the table - **so take care.**

Exercise 7

1. Increase the **prop_poll_tax** held on the table by 5% for all properties in Durham

2. Insert three new rows of data into the viewing table (make up the data but ensure that the property number and the tenant number both exist))

3. A mistake has been made on the viewing table, the details of the viewing for property 1006 on 11th June 1997 should not have been entered. Delete this row.

9. Creating tables

9.1 Introduction

So far in this Unit, we have looked at ways of selecting, creating and updating data in tables that already exist. In this section we will look at how you create your own tables.

9.2 Objectives

By the end of this section you will be able to

- Create a table
- Drop a table
- Create a view

9.3 Create table command

Using the text editor, we will create a new table to be added to the property database. We will call the new table advert. Each property will be advertised in a particular newspaper. A table and its columns must be created before you can put data into it. When you create a table the definition of the table is ready to be stored in the data dictionary. When you insert data into these tables, the data is not made permanent until you **commit** it. To commit data, you simply use the **commit** command at the end of the file.

commit;

Before a commit takes place , it is possible to undo the creation of data by rolling it back. The command to do this is

rollback;

Using the rollback command enables you to undo all database changes since the last commit statement was used.

In order to illustrate the creation of tables we will look at one of the create table commands used in the property.sql file.

create table viewing
(property_no number(5) not null references prop_for_rent(property_no),

tenant_no number(5) not null references tenant(tenant_no),
date_view date,
comments varchar2(50),
primary key(property_no,tenant_no));

There are several important points to note. Firstly, you have to type in the word table as well as the name of the table. The second important point is that you can use constraints when you create tables. For example, notice the 'references' constraint placed on this table. This ensures that the property number on the viewing table can only be inserted if the property number exists on the property table. This ensures the integrity of data, i.e. no one can view a property that does not exist! The primary key of the table is also set up when the table is created. In this case it is a compound key made up of the property number and the tenant number.

There are several data types that can be used in Version 8 of Oracle:-

Datatype	**Description**
VARCHAR2(size)	Variable length character data. A maximum size must be specified
CHAR(size)	Fixed-length character data
NUMBER	Numeric data
DATE	Date and time data
LONG	Variable length character data - Can be up to 2 Gigabytes
CLOB	Single byte character data - up to 4 Gigabytes
BLOB	Binary data up to 4 Gigabytes
BFILE	Binary data stored in an external file - can be up to 4 Gigabytes

The most usual ones used are the first four in the table.

9.4 Dropping (deleting) a Table

To remove the definition of a table from the data dictionary, you have to use the **drop table** command. The format of this is

drop table tablename;

When you drop a table, this deletes not only the definition of the table, but all of the data in it too. It is usual to place the drop table command at the beginning of the text file.

Exercise 8

1. Using a text file write the SQL statement to create the **advert** table.
   The table consists of the following columns

Column name	Data Type	Size	Null Allowed?
advert_id	number	8	No
newpaper_name	varchar2	25	No
property_no	number	5	No

   The primary key is the advert_id and the property_no together.
   A constraint should be placed on the table to ensure that an advert can only be placed for a property that exists.

   Ensure that a drop table command has been placed at the start of the file and that there is a commit at the end of it.

   NOTE: When you run your file the commands will be executed sequentially, so that if the table already exists it will be dropped first, and then created. If the table does not already exist, then SQL will attempt to execute the drop statement, but will give an error message saying that the table was not found. This will not cause a problem as it will then continue to be executed and the table will be successfully created - provided that you have not made any errors in your CREATE TABLE statement.

2. Save your file. Call it **advertcreate.sql**
   Run the file using the start command, i.e.

   start advertcreate

   When you have done this, you should see a message saying that the table has been created. If you get an error message go back to your file and correct the errors, then run the file again.

   Remember that you will need to state the full path name of the file.

3. Insert some values into the advert table. (Make up your own data - but remember the constraint placed on the table!)
   Remember as well that you need an insert statement for every row.

   Your text file should now consist of

   a drop table statement
   a create statement
   an insert statement (several)
   a commit command

4. Save and rerun your file to insert the data.

9.5 Creating a view

A view is a subset of the database presented to the user. There are several advantages to using a view.

- It promotes security by restricting access to limited data.
- It can make complex queries easier
- It allows you to present different views of the data to different people

A view can be queried in the same manner as a real table. Creating a view is very similar to creating a table.

```
create view staff_salaries as
        select staff_no, staff_surname, staff_forenames, staff_salary
        from staff;
```

Notice that not all of the columns from the staff table have been included in the view. This is a very useful security feature as it can be used to prevent users from seeing sensitive information.
All the usual parts of a select statement that you have learned so far can be used when creating a view. The where clause and table joins can also be used.
You select information from a view in exactly the same way that you select it from a table. For example, to select information from the above view:

```
        select *
        from staff_salaries;
```

Exercise 9

1. Create a view, which shows the lease_no, property_no and rent_pm for each property that has a monthly rent of more than £500. Display a new column heading for the rent per month as 'expensive'.

2. Have a guess at the statement that would drop the view you have just created. Write down the statement here - then try it out.

10. Relational Algebra

10.1 Introduction

In this section we will illustrate how relational algebra queries can be converted into SQL.

10.2 Objectives

By the end of this section you will be able to:

- Write equivalent SQL queries for simple relational algebra operations
- Use some of the relational algebra operators in SQL
- Write complex relational algebra operations and their equivalent in SQL

10.3 Project Operation

The relational algebra operation PROJECT specifies which columns (i.e. attributes) to return in the result of a query. For example, the following relational algebra query:

TENANT [t_surname, t_forenames]

would be written in SQL as:

```
SELECT t_surname, t_forenames
FROM tenant;
```

10.4 Restrict Operation

The relational algebra RESTRICT returns all rows (i.e. tuples) matching the given condition. For example, the following relational algebra query:

PROP_FOR_RENT WHERE prop_poll_tax > 300;

would be written in SQL as:

```
SELECT *
FROM prop_for_rent
WHERE prop_poll_tax > 300;
```

10.5 Join Operation

The relation algebra JOIN combines data from two or more tables based on a given join predicate. For example, the following relational algebra query:

BRANCH JOIN (branch.branch_no = staff.branch_no) STAFF

would be written in SQL as:

```
SELECT *
FROM branch, staff
WHERE branch.branch_no = staff.branch_no;
```

Exercise 10

For the following relational algebra operations, produce the equivalent SQL query in Oracle.

1. List the owner number, surname and forenames of all owners:

   OWNER [owner_no, ow_surname, ow_forenames]

2. List all information about branches which are based in Sunderland:

   BRANCH WHERE br_town = 'Sunderland';

3. List all information about properties for rent, including the full description of their property type:

   PROP_FOR_RENT JOIN (prop_for_rent.prop_type = prop_type.prop_type) PROP_TYPE

10.6 Combining Relational Algebra Operators

To be able to perform complex queries needs to be able to combine relational algebra operations. For example, the previous join operation would return all attributes from the two relations, thus we need to be able to PROJECT the result over the required attributes, and specify an extra condition to only include results where the staff's salary is greater than 10000. In the relational algebra, we would write this as follows:

((BRANCH JOIN (branch.branch_no = staff.branch_no) STAFF) WHERE staff_salary > 10000) [staff.branch_no, staff.staff_surname, staff.staff_forenames, branch.br_street, branch.br_town]

Notice the use of brackets to nest the queries. This would be written in SQL as:

```
SELECT staff.branch_no, staff.staff_surname,
staff.staff_forenames, branch.br_street, branch.br_town
FROM branch, staff
WHERE branch.branch_no = staff.branch_no
AND staff.staff_salary > 15000;
```

10.7 Difference, Intersect and Union

SQL contains direct support of the relational algebra operations difference, intersect and union. The relational algebra operations and their equivalent in SQL are given in the following table.

Relational Algebra Operator	SQL Operator
UNION	UNION
INTERSECT	INTERSECT
DIFFERENCE	MINUS

Use of difference, intersect and union requires the input relations to be union compatible, so we need to combine these operations with PROJECT. For example, the following relational algebra query returns all towns in which staff AND branches reside:

(BRANCH [br_town]) INTERSECT (STAFF [staff_town])

This would be written in SQL as:

```
SELECT br_town
FROM branch
INTERSECT
SELECT staff_town
FROM staff;
```

Exercise 11

Rewrite the following queries, firstly in relational algebra, and then convert the algebra to SQL.

1. List the lease number, property number, tenant number, rent per month, tenant's surname and forename, for tenant's whose preferred rental type is Flat.

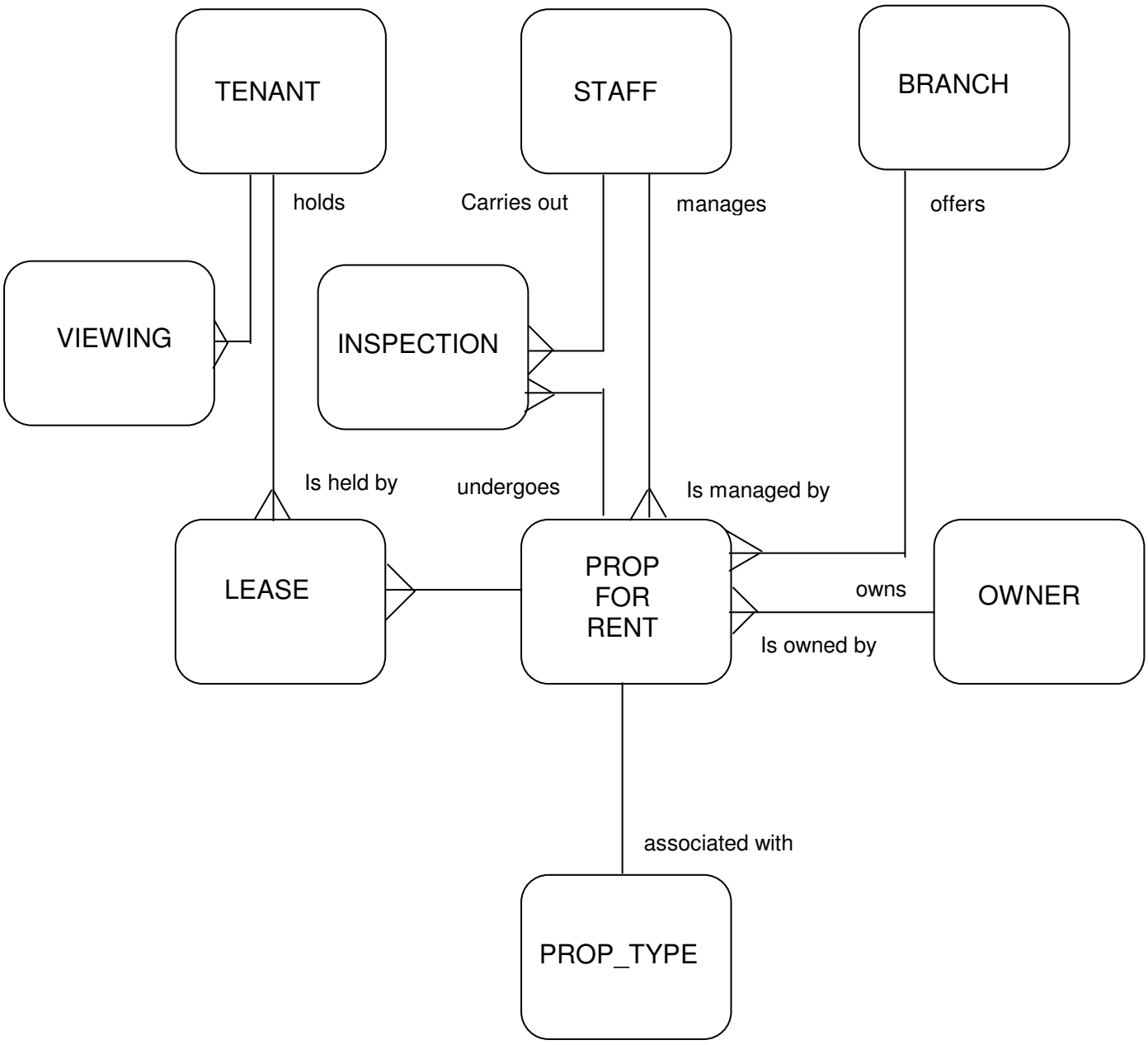
2. List all property numbers which are for rent, but which are NOT subject to a lease agreement. HINT: think about how you would use the DIFFERENCE operation for this query.

3. For all inspections which have been carried out since 1st May 1997, list the property_number, inspection date, comments, staff surname and staff forename.

4. List the surnames and forenames of all tenants and staff.

APPENDIX 1

PROPERTY RENTAL SERVICES
CASE STUDY



LEASE

Name	Null?	Type
LEASE_NO	NOT NULL	NUMBER(5)
*PROPERTY_NO	NOT NULL	NUMBER(5)
*TENANT_NO	NOT NULL	NUMBER(5)
RENT_PM		NUMBER(6,2)
PAYMENT_METHOD	NOT NULL	CHAR(1)
DEPOSIT_AMOUNT		NUMBER(6,2)
DEPOSIT_PAID		CHAR(1)
STDATE	NOT NULL	DATE
ENDDATE		DATE

VIEWING

Name	Null?	Type
PROPERTY_NO	NOT NULL	NUMBER(5)
TENANT_NO	NOT NULL	NUMBER(5)
DATE_VIEW		DATE
COMMENTS		VARCHAR2(50)

TENANT

Name	Null?	Type
TENANT_NO	NOT NULL	NUMBER(5)
T_SURNAME	NOT NULL	VARCHAR2(25)
T_FORENAMES	NOT NULL	VARCHAR2(25)
T_STREET	NOT NULL	VARCHAR2(25)
T_AREA		VARCHAR2(20)
T_TOWN	NOT NULL	VARCHAR2(20)
T_POSTCODE	NOT NULL	VARCHAR2(8)
T_TELNO		VARCHAR2(12)
T_PREF_TYPE		CHAR(1)
T_MAX_RENT		NUMBER(6,2)

INSPECTION

Name	Null?	Type
PROPERTY_NO	NOT NULL	NUMBER(5)
STAFF_NO	NOT NULL	NUMBER(5)
INSPECT_DATE	NOT NULL	DATE
COMMENTS		VARCHAR2(50)

PROP_FOR_RENT

Name	Null?	Type
PROPERTY_NO	NOT NULL	NUMBER(5)
PROP_STREET	NOT NULL	VARCHAR2(25)
PROP_AREA		VARCHAR2(20)
PROP_TOWN	NOT NULL	VARCHAR2(20)
PROP_PCODE	NOT NULL	VARCHAR2(8)
PROP_TYPE	NOT NULL	CHAR(1)
PROP_ROOMS	NOT NULL	NUMBER(2)
PROP_RENT_PM	NOT NULL	NUMBER(7,2)
PROP_POLL_TAX		NUMBER(6,2)
*OWNER_NO	NOT NULL	NUMBER(5)
*STAFF_NO		NUMBER(5)
*BRANCH_NO	NOT NULL	NUMBER(3)
AVAILABLE		CHAR(1)
COMMENTS		VARCHAR2(150)

PROP_TYPE

Name	Null?	Type
PROP_TYPE	NOT NULL	CHAR(1)
TYPE_DESC	NOT NULL	VARCHAR2(15)

OWNER

Name	Null?	Type
OWNER_NO	NOT NULL	NUMBER(5)
OW_SURNAME	NOT NULL	VARCHAR2(25)
OW_FORENAMES	NOT NULL	VARCHAR2(25)
OW_STREET	NOT NULL	VARCHAR2(25)
OW_AREA		VARCHAR2(20)
OW_TOWN	NOT NULL	VARCHAR2(20)
OW_PCODE	NOT NULL	VARCHAR2(8)
OW_FEE		NUMBER(6,2)

STAFF

Name	Null?	Type
STAFF_NO	NOT NULL	NUMBER(5)
*BRANCH_NO		NUMBER(3)
STAFF_SURNAME	NOT NULL	VARCHAR2(25)
STAFF_FORENAMES	NOT NULL	VARCHAR2(25)
STAFF_STREET	NOT NULL	VARCHAR2(25)
STAFF_AREA		VARCHAR2(20)
STAFF_TOWN	NOT NULL	VARCHAR2(20)
STAFF_PCODE	NOT NULL	VARCHAR2(8)
STAFF_TELNO		VARCHAR2(12)
STAFF_GENDER		CHAR(1)
STAFF_SALARY		NUMBER(8,2)

BRANCH

Name	Null?	Type
BRANCH_NO	NOT NULL	NUMBER(3)
BR_STREET	NOT NULL	VARCHAR2(25)
BR_AREA		VARCHAR2(20)
BR_TOWN	NOT NULL	VARCHAR2(20)
BR_PCODE	NOT NULL	VARCHAR2(8)
BR_TELNO		VARCHAR2(12)