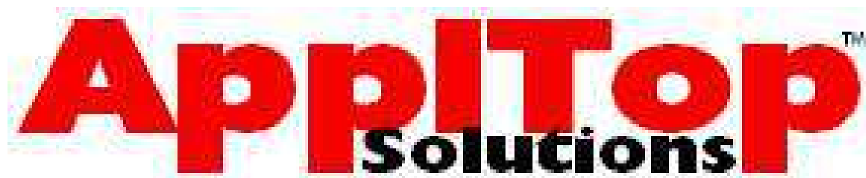
An



Training Guide

SQL & SQL*Plus for Beginners

Course Introduction

Copyright © 1999 – 2004 ApplTop Solutions Limited

Course Objectives

This course is designed to give each delegate a basic understanding/awareness of the following...

- Oracle SQL
- Oracle SQL*Plus

Copyright © 1999 – 2004 ApplTop Solutions Limited

Course Objectives

This course is designed to give each delegate a basic understanding of the following topics...

- Oracle SQL. You will learn how to use the industry standard tool for working with a Relational Database. You will also learn many other things, from constructing simple queries to creating your own tables.
- Oracle SQL*Plus. After the course you should have a good understanding of SQL*Plus, from starting it, to creating simple reports on it.

During the course there will be several exercises to complete.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Note that this guide is meant only as an introduction to SQL and SQL*Plus and therefore much of the newer, more advanced features available in Oracle databases 8i, 9i and 10g are not covered.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Course Contents

1 – Getting Started **11**
RDB – A quick refresher 12
What are SQL & SQL*Plus? 19

2 – Introduction to SQL & SQL* Plus **21**
Introduction to SQL 22
DML or DDL? 23
Starting SQL*Plus 24

Lab 1 26

Entering SQL Commands 27
Basic Query Block 28
Selecting Specific Columns 29
Arithmetic Operators 30
Column Aliases 31
Concatenation 32
Literals 33
`NULL` Values 34
Duplicate Rows 37
Ordering Data 38
Row Restriction 40
- Using Logical Operators 41
- Using SQL Operators 42
- Using `LIKE` 43
- Negating a Comparison 44
- Multiple Conditions & Operator Precedence 46

Basic SQL*Plus 50
Summary 55
Lab 2 57

Copyright © 1999 – 2004 ApplTop Solutions Limited

3 – Row & Group Functions **58**
Row functions 60
Character functions 62
LOWER, UPPER & INITCAP 63
LPAD & RPAD 64
SUBSTR 65
INSTR 66
LTRIM/RTRIM 67
LENGTH 68
TRANSLATE 69
REPLACE 70
Number functions 71
ROUND 72
TRUNC 73
SIGN 74
CEIL & FLOOR 75
Mathematical functions 76
Oracle Dates & Date Functions 77
MONTHS_BETWEEN 80
ADD_MONTHS 81
NEXT_DAY 82
LAST_DAY 83
ROUND & TRUNC 84
Conversion Functions 85
TO_CHAR & Common Format Masks 86
TO_NUMBER 89
TO_DATE 90
Functions that accept any kind of data 91
NVL 92
GREATEST & LEAST 93
DECODE 94

Copyright © 1999 – 2004 ApplTop Solutions Limited

Nesting Functions 95
Group Functions 96
Group functions 97
Grouping Data 98
Omitting Groups 100
Summary 101

Lab 3 103

4 - Querying More Than One Table 104

Joins 106
Product 107
Equi join 108
Table Aliases 109
Non-Equi join 110
Outer join 111
Self join 112
Set Operators 113
UNION 115
INTERSECT 116
MINUS 117
Rules 118
Subqueries 119
Single Row 120
Multiple Row 122
ANY/SOME Operator 123
ALL Operator 124
Correlated 126
EXISTS Operator 128
Summary 129

Lab 4 131

5 - Modifying Data & the Database 132
Inserting new data 134
Updating existing data 137
Deleting data 140
Transaction Processing 142
COMMIT 144
ROLLBACK 145
SAVEPOINT 146
Using DDL 148
Tables 150
Indexes 154
Synonyms 156
Privileges 157
Views 158
Sequences 160
Summary 163

Lab 5 165

6 – More SQL* Plus 166
SQL*Plus variables 168
Ampersand variables 169
Double Ampersand variables 170
Basic SQL*Plus Reporting 171
Adding a page title 172
Setting a BREAK point 173
Saving output to a file 174
Summary 175

Lab 6 177

Copyright © 1999 – 2004 ApplTop Solutions Limited

Answers to Exercises 178

Demo Tables 185

Copyright © 1999 – 2004 ApplTop Solutions Limited

Section One
Getting Started

Copyright © 1999 – 2004 ApplTop Solutions Limited

RDB – A Quick Refresher

RDB Constructs

Four basic constructs make up a Relational Database: -

- Tables
- Columns
- Rows
- Fields

In addition to the above there is the concept of key values: -

- Primary Keys
- Foreign Keys

Copyright © 1999 – 2004 ApplTop Solutions Limited

RDB Constructs

To understand Relational Databases, you need to understand the four basic constructs of an RDB and the concept of key values: -

Tables	A table can be thought of as a storage area, like a filing cabinet. You use tables to store information about anything: employees, departments, etc. A database may only contain one table, or it may contain thousands.
Rows	If table has been defined to hold information about (for example) employees, a row is a horizontal cross section into the table which contains the information about a single employee.
Columns	A column is the vertical cross section of a table, or in other words, a column defines each of the attributes about the data stored on a particular table. For example, if you have a table which holds employee information, you could have several columns which determine employee number, name, job…etc.
Fields	A field is where rows and columns intersect. A field points to a particular column on a particular row within a table.
Primary Keys	A primary key is a column that defines the uniqueness of a row. For example, with *employee number*, you would only ever want one employee with a number of 10001.
Foreign Keys	A foreign key defines how different tables relate to each other. For example, if you have defined two tables, one for employees and one for departments, there will be a foreign key column on the employee table which relates to the department code column on the department table.

Copyright © 1999 – 2004 ApplTop Solutions Limited

RDB Constructs

The EMP Table

A primary key column containing employee number

A normal column, not a key value

EMP	ENAME	JOB	MGR	HIREDATE	SAL	BONUS	DEPTNO
10001	HAMIL	PROGRAMMER	10005	10-JAN-1976	2,000.00	500.00	10
10002	FORD	ANALYST	10005	20-MAR-1976	3,000.00		10
10003	LUCAS	BIG BOSS		18-AUG-1976	10,000.00		20
10004	JONES	PROGRAMMER	10005	27-SEP-1976	2,100.00	1,500.00	10
10005	FISHER	TEAM LEADER	10003	14-APR-1976	4,000.00		20

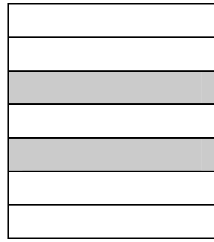
A single row representing a single employee

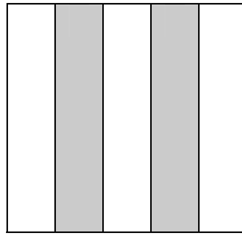
A field, found at the intersection of a row and column

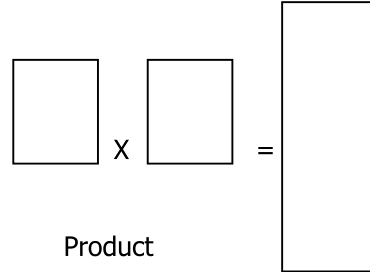
A foreign key column which links employee to department

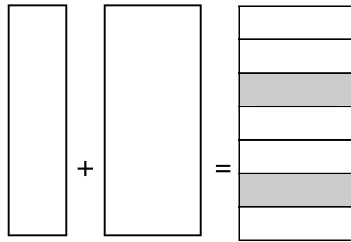
Copyright © 1999 – 2004 ApplTop Solutions Limited

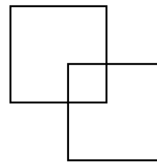
Relational Operators

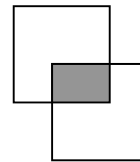


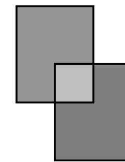












Copyright © 1999 – 2004 ApplTop Solutions Limited

Relational Operators

Relational Operators are used to extract and combine data for selection. They can be thought of as functions that can be performed on data held within a relational database.

Relation	Description
Restriction	Restriction is an operation that selects rows from a relation that meet certain conditions. There can be none, one or many conditions. This is sometimes referred to as a 'horizontal subset'.
Projection	This is an operation that only selects specified columns from a relation and is suitably referred to as the 'vertical subset'.
Product	The product operation is the result of selecting rows from two or more relations. The resulting set of rows returned is often very large.
Join	This operation is the result of selecting rows from two more relations using one or more specified conditions. Joins are often made via foreign key columns.
Union	This retrieves unique rows that appear in either or both of two relations. UNION ALL can be used to retrieve ALL rows from either or both tables.
Intersect	This retrieves all rows that appear in both of two relations.
Difference	This retrieves rows that appear in one relation only.

Copyright © 1999 – 2004 ApplTop Solutions Limited

RDB Properties

Database Properties: -

- Individual collection of tables
- User does not need to know how data is accessed
- Uses simple language called SQL for all actions
- Uses set operations, not row by row processing
- Can be modified online

Table Properties: -

- No duplicate columns or rows
- Row and column order is insignificant
- Field values are atomic, i.e. They cannot be broken into component parts

Copyright © 1999 – 2004 ApplTop Solutions Limited

RDB Properties

A Relational Database has the following properties: -

- It appears as a collection of individual tables to the user, even though the database may be contained in a single file.
- The user does not specify the access route and does not need to know how the data is physically stored.
- The user must know which tables can be accessed by table name
- The user queries and modifies the database using an English like, non-procedural 4GL, otherwise known as a Structured Query Language or SQL for short.
- The database provides the user with a set of operators for partitioning and combining relations via SQL. (see Relational Operators)
- Its structure can be modified easily

Table Properties

A table on a database has the following properties: -

- Each row on a table should be unique. Each row can be identified by a Primary Key column *
- There must be no duplicate column names
- Row order is insignificant, default order is the order in which rows are inserted into a table
- Column order is insignificant when the data is stored. Its order is defined when data is retrieved.
- All field values are atomic, or in other words they cannot be broken down into smaller components **

 * You may come across table definitions in Oracle that have no primary key column enforcing uniqueness. This is allowed, but under the hood Oracle maintains uniqueness using a special column called ROWID

** Traditionally, older database systems held rows as a single column, which was broken into its component parts by the programmer.

Copyright © 1999 – 2004 ApplTop Solutions Limited

What are SQL & SQL* Plus?

Okay, what exactly are SQL & SQL*Plus?.

SQL is the standard language used for querying Relational Databases. It allows you to view and change data held within an RDB as well as allowing you to actually modify the structure of the database.

SQL* Plus is Oracle's front-end interface to SQL. From SQL*Plus you can create SQL scripts. SQL*Plus also has its own set of commands which aid SQL script development, and it also allows you to produce simple reports.

Copyright © 1999 – 2004 ApplTop Solutions Limited

What is Next?

We've had a quick refresher on the concepts of a Relational Database and we've had a very brief description what SQL & SQL*Plus are. The next section jumps straight in and actually starts on basic SQL commands.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Section Two
Introduction to SQL

Introduction to SQL

Very briefly, before we get into actually using SQL, let's summarise what can be done with SQL: -

- Query the Database - SQL allows you to easily query data held on the database. Queries can be very simple and only take up a couple of lines or they can be quite complex and take up several pages of text.
- Change Data held within the Database - Changing data is just as easy as querying the data. Existing data can be modified, and you can remove data or insert new data.
- Change the Structure of the Database - SQL allows you to actually modify the structure of the database, meaning you can easily create new database objects such as tables, indexes, views, sequences…etc.

Copyright © 1999 – 2004 ApplTop Solutions Limited

DML or DDL?

Most SQL commands fall into one of three categories:

- Queries - You will use these the most. They are for retrieving data from the database, and they are neither DML nor DDL.
- Commands that allow you to modify the data held within the database - these commands are referred to as ***Data Manipulation Language*** commands or ***DML*** for short.
- Commands which allow you to modify the structure of the database - these commands are known as ***Data Definition Language*** commands or ***DDL*** for short

During this course we will learn how to use commands from all categories.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Starting SQL* Plus

Before you can use any SQL commands, you must log into the database using SQL*Plus. SQL*Plus can usually be found as an icon on your desktop (if you are running a Windows OS), or it is available whilst you are logged into a UNIX box.

- If you want to run SQL*Plus from the Windows desktop, simply double click on the icon. You will then be asked for a username and password.
- If you are wanting to start SQL*Plus whilst logged into a UNIX box, then simply type `sqlplus` from the command line. You will again be asked for a username and password.

Starting SQL* Plus

Before you can use any SQL commands, you must log into the database using SQL*Plus. SQL*Plus can usually be found as an Icon on your desktop (if you are running a windows OS), or, SQL*Plus can be accessed whilst logged into a UNIX box where the database server can be found.

- If you want to run SQL*Plus from Windows, simply double click on the SQL*Plus Icon, at this point you will be asked for an Oracle username and password.
- If you are wanting to start SQL*Plus from in your UNIX session, simply type `sqlplus` at the command line, you will now be asked for a Oracle username and password.

In either case, enter the username and password provided. You will now see the SQL*Plus command prompt:

```
SQL>
```

To quit out of SQL*Plus simply enter the following command:

```
SQL> quit
```

UNIX – An example

From the UNIX command line, simply enter:

```
sqlplus
```

or, you can provide the username and password from the command line:

```
sqlplus user/password
```

you can also provide the database instance name you wish to log onto:

```
sqlplus user/password@DEV
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Lab 1

1. Log into the database using SQL*Plus from the Windows desktop. Once you see the SQL*Plus prompt, quit out of SQL*Plus.
2. Log into the database using SQL*Plus from your UNIX session. Once you see the SQL*Plus prompt, quit out of SQL*Plus.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Entering SQL Commands

Once you are logged into the database using SQL*Plus, you can enter either SQL*Plus commands or SQL commands. There are a few things you should note before you start typing:-

- Commands may be on a single line, or many lines
- You should place different clauses on separate lines for the sake of readability - also make use of tabs and indents
- SQL Command words cannot be split or abbreviated
- SQL commands are not case sensitive
- All commands entered at the SQL*Plus prompt are saved into a command buffer
- You can execute SQL commands in a number of ways:
  - Place a semicolon (`;`) at the end of the last clause
  - Place a forward slash (`/`) at the SQL prompt
  - Issue the SQL*Plus `r[un]` command

Copyright © 1999 – 2004 ApplTop Solutions Limited

Basic Query Block

You now know how to log into the database; we have also covered how to enter basic commands. Let's now try to write our first SQL statement to query the database.

The basic query block is made up of two clauses:

- `SELECT` which columns?
- `FROM` which tables?

For example:

```
SELECT ename
FROM   emp;
```

The above statement will select the `ENAME` column from the `EMP` table. You can use a `*` to specify all columns:

```
SELECT *
FROM   emp;
```

Table Definition

To view the columns on a table use the `desc` SQL*Plus command:

```
desc emp
```

Selecting Specific Columns

You can select any columns in any order that appear on the table specified in the `FROM` clause.

- Use a comma (`,`) as a column separator
- Specify columns in the order you wish to see them in
- Data is justified by default as follows:-
  - Dates/characters to the left
  - Numbers to the right

For example,

```
SELECT    empno
     ,    ename
     ,    sal
FROM emp;
```

would produce output as follows:-

```
EMPNO      ENAME             SAL
-------    -----------    ------
100001     JONES          100.00
100002     SMITH           45.48
100010     WARD           234.89
100010     FORD           523.56
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Arithmetic Operators

At some point you may want to perform some arithmetic calculations based on the data returned by the SELECT statement. This can be achieved using SQL's arithmetic operators:

- Multiply *
- Divide /
- Add +
- Subtract -

Normal operator precedence applies - you can also use brackets to force precedence.

For example, to find the annual salary of all employees:

```
SELECT     empno
     ,     sal * 12
FROM       emp;
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Column Aliases

The heading SQL*Plus will give to columns is by default based on the column name, but what about situations as in the previous example?

```
SELECT empno
,      sal * 12
FROM   emp;
```

This SQL statement will give the following output:

```
EMPNO        SAL*12
-----        ------
100001       12500
100002       25000
```

Notice the second column, `SAL*12` - not very user friendly. Using column aliases you can change the headings, then simply follow the column name with a space and the column alias:

```
SELECT empno   employee_number
,      sal*12  annual_salary
FROM   emp
```

Column aliases must not contain any white space and the case is ignored. You can get around this by enclosing the alias in double quotes, as follows:

```
SELECT    empno     "Employee Number"
,         sal*50    "Annual Salary"
FROM emp;
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Concatenation

You can merge the output of two or more columns together using the concatenation operator (||). For example:

```
SELECT    'Name='||ename name
FROM emp;
```

This SQL statement will give the following output:

```
NAME
---------
Name=JONES
Name=SMITH
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Literals

A literal is a character/expression in the SELECT clause. For example,

```
SELECT    ename
     ,    'works in department'    literal
     ,    deptno
FROM emp;
```

gives the following output:

```
ENAME          LITERAL                  DEPTNO
---------      -------                  ------
SMITH          works in department      10
JONES          works in department      20
. . .          . . .                    . . .
```

Date and character literals must be enclosed in single quotes.

NULL Values

If a row contains a column which has no data in it, then its value is said to be NULL.

NULL is a value that is unavailable, unassigned, unknown or inapplicable.

- NULL is not the same as ZERO
- If NULL is part of an expression, then the result will ALWAYS be NULL

In our example `emp` table, we have the column `comm` which is only populated for Salesmen. If you were to perform the following:

```
SELECT    ename
     ,    sal*12 + comm remuneration
FROM emp;
```

then the `remuneration` column would be NULL wherever `comm` was NULL.

NVL Function

You can solve the problem of NULL values causing expressions to be NULL by using the NVL function. NVL simply puts a value where a NULL would otherwise appear.

- NVL can be used with date, character and number datatypes
- NVL takes two parameters:-
  - the column you are checking for NULL
  - the value you wish NVL to return if the first parameter is NULL

So, making use of NVL in the example on the previous page, we have:

```
SELECT surname
,      sal*12 + NVL(comm,0) remuneration
FROM emp;
```

The use of NVL in this example **always** ensures the value of the comm column is assumed to be 0 if it is NULL, thus ensuring the remuneration column is always calculated correctly.

Copyright © 1999 – 2004 ApplTop Solutions Limited

NVL Examples

Here are a few examples using the NVL function:-

```
NVL(a_string,'NOT SET')
NVL(a_date,SYSDATE)
NVL(a_number,-1)
```

Note

Both parameters passed to NVL must have matching datatypes.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Duplicate Rows

Whenever you execute a SELECT statement, by default, all rows are selected. For example,

```
SELECT    deptno
FROM emp;
```

will give something like:

```
DEPTNO
----------
10
20
20
30
10
40
50
50
```

You can prevent duplicate rows from being selected by using the DISTINCT keyword. Simply follow the SELECT keyword with DISTINCT; for example,

```
SELECT    DISTINCT  deptno
FROM      emp;
```

would give:

```
DEPTNO
----------
10
20
30
40
50
```

The DISTINCT keyword affects ALL columns in the SELECT clause.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Ordering Data

The order of rows returned by a SELECT statement is, by default, undefined. You can use the ORDER BY clause to sort the rows. For example,

```
SELECT    empno
FROM      emp
ORDER BY  empno;
```

will give something like:

```
EMPNO
----------
10001
10002
10003
10004
. . .
```

The ORDER BY clause is simply added to the end of your SELECT statement.

- Default order is ascending - use DESC after the column name to change order
- There is no limit on the number of sort columns
- There is no need to SELECT sort column
- You can sort using expressions and aliases
- NULL values are sorted high

Copyright © 1999 – 2004 ApplTop Solutions Limited

ORDER BY Examples

Here are a few examples of using the ORDER BY clause:

```
SELECT    empno
FROM      emp
ORDER BY  empno;

SELECT    ename
     ,    sal*12 + NVL(comm,0) renum
FROM      emp
ORDER BY  renum DESC;

SELECT    deptno
     ,    hiredate
     ,    ename
FROM      emp
ORDER BY  deptno
     ,    hiredate DESC
     ,    ename
```

Row Restriction

A simple SELECT statement, such as:

```
SELECT    empno
FROM      emp
ORDER BY  empno;
```

will return all rows from the emp table, but if you only want a list of employees who work in department 10, you would use the WHERE clause. The WHERE clause MUST appear after the FROM clause. You specify conditions in the WHERE clause that must be met if the row is to be returned. Conditions are basically comparisons of columns/literals using logical operators and SQL operators. Here is an example:

```
SELECT    empno
FROM      emp
WHERE     deptno = 10
ORDER BY  empno;
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Row Restriction using Logical Operators

The following logical operators are available:

Operator	Meaning
=	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

A `WHERE` clause is generally made up of three elements:

- Column name
- Comparison operator (logical operator)
- Column name/literal

Some examples:

```
SELECT    empno
FROM      emp
WHERE     hiredate <= '01-Jan-98';

SELECT    ename
     ,    empno
FROM      emp
WHERE     job = 'CLERK';

SELECT    empno
FROM      emp
WHERE     deptno = 10;

SELECT    *
FROM      emp
WHERE     comm > sal;
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Row Restriction using SQL Operators

The following SQL operators are available:

Operator	Meaning
BETWEEN	between two values (inclusive)
IN	match any in a list of values
IS NULL	is a NULL value
LIKE	match a character pattern

Some examples:

```
SELECT    empno
FROM      emp
WHERE     deptno BETWEEN 20 AND 50;

SELECT    ename
     ,    job
FROM      emp
WHERE     deptno IN (10,20,30,40,50);

SELECT    empno
FROM      emp
WHERE     comm IS NULL;
```

The LIKE operator is covered on the next page.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Row Restriction using LIKE

Sometimes you may not know the exact value to search for: for example, you may want a list of all employees whose name begins with the letter S.

You perform this kind of comparison using the LIKE operator.

The LIKE operator expects a wildcard search pattern. A wildcard is basically a way to specify parts of a string which you do not know.

Wildcard Symbols Available

Symbol	Represents
%	any sequence of zero or more characters
_ (underscore)	any single character

Some examples:

To list all employees whose names begin with S:

```
SELECT     *
FROM       emp
WHERE      ename LIKE 'S%';
```

To list all employees whose names have exactly 4 characters:

```
SELECT     *
FROM       emp
WHERE      ename LIKE '_ _ _ _';  (no spaces between _ )
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Negating a Comparison

Rather than writing a WHERE clause which says "select rows which meet a certain condition", it may well be easier to say "select rows which DO NOT meet a certain condition". You do this with a negate expression.

Negating Logical Operators

Operator	Meaning
<>	Not equal to
!=	Same as <>
NOT column =	Same as <>
NOT column >	Not greater than

Negating SQL Operators

Operator	Meaning
NOT BETWEEN	not between two values
NOT IN	not in a list of values
NOT LIKE	not in a character pattern
IS NOT NULL	is not a NULL value

Copyright © 1999 – 2004 ApplTop Solutions Limited

Examples of Negating a Comparison

To select all rows where the department is not 10, 20 or 30:

```
SELECT    *
FROM      emp
WHERE     deptno NOT IN(10,20,30);
```

To select all rows where the name does not begin with S:

```
SELECT    *
FROM      emp
WHERE     ename NOT LIKE 'S%';
```

To select all rows where the `comm` column is not null:

```
SELECT    *
FROM      emp
WHERE     comm IS NOT NULL;
```

To select all rows where the `hiredate` is not during 1998:

```
SELECT    *
FROM      emp
WHERE     hiredate NOT BETWEEN '01-jan-99'
                   AND '31-jan-99;
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Multiple Conditions

A WHERE clause is not restricted to a single condition - it can contain any number of conditions. Multiple conditions used together are referred to as a compound logical expression.

You can use the AND and OR keywords to create WHERE clauses with multiple conditions.

- A condition using the AND keyword is true if BOTH conditions are true
- A condition using the OR keyword is true if EITHER condition is true

Some examples:

To select all rows where the department is 10 AND the salary is greater than 1000:

```
SELECT    *
FROM      emp
WHERE     deptno = 10
AND       sal >= 1000;
```

To select all rows where the department is 10 OR the salary is greater than 1000:

```
SELECT    *
FROM      emp
WHERE     deptno = 10
OR        sal >= 1000;
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Multiple Condition Examples

To select all employees who were hired before July 1997 whose commission is more than £1,000 but less than £1,500:

```
SELECT    *
FROM      emp
WHERE     hiredate < '01-jul-99'
AND       NVL(comm,0) BEWTEEN 1001 AND 1499;
```

To select all employees who are not clerks but do work in departments 10, 40 or 50:

```
SELECT    *
FROM      emp
WHERE     job <> 'CLERK'
AND       deptno IN (10,40,50);
```

To select employees who work in department 10 who earn more than £10,000 per annum, or employees who work in department 30 who earn more than £15,000 per annum:

```
SELECT    *
FROM      emp
WHERE     (deptno = 10 AND sal*12 > 10000)
OR        (deptno = 30 AND sal*12 > 15000);
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Operator Precedence

When constructing a `WHERE` clause you need to be aware of operator precedence. This determines how a condition is evaluated, and it can greatly affect the results. Operators are evaluated in a strict order, as follows:

1. All Logical and SQL operators
2. `NOT`
3. `AND`
4. `OR`

Default Operator Precedence can be overruled by using parentheses.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Operator Precedence Examples

To select all managers in any department AND all clerks in department 10:

```
SELECT *
FROM   emp
WHERE  deptno = 10
AND    job='CLERK' OR JOB ='MANAGER' ;
```

The following would produce different results:

```
SELECT *
FROM   emp
WHERE  deptno = 10
AND    (job='CLERK' OR JOB ='MANAGER');
```

The second statement says "select all rows where deptno is 10 and job is either CLERK or MANAGER" - this would only give rows where department is 10 AND job is MANAGER OR CLERK.

Tip

If you are unsure as to exactly how a condition will be evaluated, feel free to use parentheses.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Basic SQL* Plus

As we saw earlier, the SQL language is actually accessed from within a tool called SQL*Plus. Before we finish this section of the course with the exercises, we need to take a look at some of the basic SQL*Plus commands you will need to use when working with SQL. We will quickly cover the following:

- Editing and executing SQL in the buffer
- Saving, loading and executing SQL files

Copyright © 1999 – 2004 ApplTop Solutions Limited

SQL* Plus Buffer

When you enter a SQL command at the SQL*Plus prompt, it is stored in the SQL buffer and it will remain there until you enter a new command. If you press `[RETURN]` before completing a line, SQL*Plus will prompt you for a line number where you can continue to enter the command. You terminate the buffer by pressing `[RETURN]` again.

A semicolon, forward slash or entering `RUN` will terminate and execute a SQL statement.

For example:

```
SQL> select *
  2  from emp;
```

or:

```
SQL> select *
  2  from emp
  3  /
```

or:

```
SQL> select *
  2  from emp
  3  [RETURN]
SQL> run
```

All of the above examples do the same thing.

Editing SQL

Once you have entered some SQL, you may want to execute the same code but with a few changes. You could simply re-type the code with the changes, or you could use some basic SQL*Plus commands to edit what is stored in the SQL Buffer.

Editing Commands

Command	Abbreviation	Purpose
`APPEND text`	`A text`	Adds text to end of current line
`CHANGE`	`c/old/new/`	Changes old top new on current line
`CHANGE`	`c/text/`	Deletes text from current line
`CLEAR BUFFER`	`cl buff`	Deletes all lines from buffer
`DEL`		Deletes current line
`INPUT`	`i`	Inserts an indefinite number of lines
`INPUT`	`i text`	Inserts a line consisting of text
`LIST`	`l`	Lists all lines in buffer
`LIST n`	`l n (or just n)`	Lists one line specified by n
`LIST m n`	`l m n`	Lists from line m to n
`ED`		Invokes editor with contents of SQL Buffer
`RUN`	`r`	Displays and runs what is in buffer
`/`		Runs what is in buffer

Copyright © 1999 – 2004 ApplTop Solutions Limited

Editing SQL Examples

Assume you have the following statement in the SQL buffer:

```
SQL> SELECT     ename
   2        ,    job
   3 FROM        emp;
```

If you wanted to change this SQL to also select the `hiredate` column after the `job` column, you would enter the following:

```
SQL> 2
  2* , empno
SQL> input , hiredate
```

You would now have:

```
SQL> SELECT     ename
   2        ,    job
   3        ,    hiredate
   4 FROM        emp;
```

To remove line 2:

```
SQL> 2
  2* , job
SQL> del
```

To change `hiredate` to `job`:

```
SQL> 2
  2* , hiredate
SQL> c/hiredate/job/
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Saving, Loading and Executing SQL

As well as using the SQL buffer to store SQL commands, you can also store your SQL in files. These files can be edited with your own editor; you can then re-call and run these files as if you had just typed the SQL commands directly into the SQL buffer. Use the following commands from the SQL*Plus prompt:

Command	Description
`SAVE filename`	Saves the current contents of the SQL buffer to a file
`GET filename`	Loads the contents of a file into the SQL buffer
`START filename`	Runs a file (can also use @file)
`ED filename`	Invokes an editor to edit the file
`EXIT`	Quits SQL*Plus

Copyright © 1999 – 2004 ApplTop Solutions Limited

Summary

To summarise, we have seen the basic syntax for the SELECT statement:

```
SELECT     [DISTINCT]{*,COLUMN [ALIAS]….}
FROM       TABLE
WHERE      CONDITION(S)
ORDER BY   {COLUMN|EXPRESSION}[ASC|DESC]};
```

- Use TABS for clarity
- Always place different clauses on different lines

We have covered the following:

- Basic SELECT clause
- The DISTINCT keyword
- Column Aliases
- The ORDER BY clause
- The WHERE clause
  - Single/Multiple conditions
  - Logical/SQL Operators
  - Negating conditions
  - Multiple conditions
  - Operator Precedence
- Basic SQL*Plus commands

Copyright © 1999 – 2004 ApplTop Solutions Limited

What is Next?

Having covered the most commonly used type of SQL statement, the `SELECT` statement and also briefly looked at SQL*Plus commands, we are now ready to take a look at some of the more complex features of SQL.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Lab 2

1. Select all rows from the salgrade table.
2. Select all rows from the emp table.
3. Select all employees who have a salary between 1600 and 3000.
4. List department number and department name in name order from the dept table.
5. Display all the different job types, in reverse order.
6. List the names and hiredate of all clerks in department 20.
7. List all employees whose name begins with S.
8. Display the name, job, mgr and sal for all employee who have a manager. Sort the list by sal descending.
9. List the name and total remuneration of all employees.
10. Display the name, salary, annual salary and commission of all salespeople whose monthly salary is less then their commission. The output should be sorted by salary, highest first.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Section Three
Row & Group Functions

Copyright © 1999 – 2004 ApplTop Solutions Limited

Row & Group Functions

So far, we have seen how to use the most commonly used command in SQL, the `SELECT` statement. Now that we have this basic knowledge we can begin to cover more complex features of SQL. In this section we will cover the following:

- Row Functions
- Group Functions & Grouping Data

Copyright © 1999 – 2004 ApplTop Solutions Limited

Row Functions

Row functions are basically pre-defined or custom built commands which can be used to modify the data in a SQL statement. Row functions have the following properties:

- They require arguments - these can be constants, variables, column names or expressions
- Functions return a single value
- Functions act on each row returned by the query
- They can be used in `SELECT`, `WHERE` and `ORDER BY` clauses
- They can be nested

Row Functions

There are many different kinds of row functions available. They are:

- Character Functions
- Number Functions
- Oracle Dates, Date Arithmetic & Date functions
- Conversion functions & format Masks
- Functions that accept any kind of datatype
- Nesting Functions

We will take a brief look at some of the more commonly used functions.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Character Functions

We will look at the following character functions:

- LOWER
- UPPER
- INITCAP
- LPAD & RPAD
- SUBSTR
- INSTR
- LTRIM & RTRIM
- LENGTH
- TRANSLATE
- REPLACE

Copyright © 1999 – 2004 ApplTop Solutions Limited

LOWER

Converts all characters to lower case

Syntax

```
LOWER(argument)
```

Example

```
SELECT     LOWER('ORACLE') …    = oracle
```

UPPER

Converts all characters to upper case

Syntax

```
UPPER(argument)
```

Example

```
SELECT     UPPER('oracle') …    = ORACLE
```

INITCAP

Forces the first letter of each word to be in upper case

Syntax

```
INITCAP(argument)
```

Example

```
SELECT     INITCAP('oracle') … = Oracle
```

LPAD & **RPAD**

Pads string to the left or right with a specified character until a specified length is reached

Syntax

```
LPAD(string,len,pstring)
RPAD(string,len,pstring)
```

Arguments

`string`	the string to be padded
`len`	the length of the final string
`pstring`	the string to use for padding

Example

```
SELECT    LPAD('ORACLE',10,'-') … = ----ORACLE
```

Notes

If `string` is longer than `len` then `string` is truncated to `len` characters.

Copyright © 1999 – 2004 ApplTop Solutions Limited

SUBSTR

The `SUBSTR` function is used to extract a portion of a string.

Syntax

```
SUBSTR(string,pos,len)
```

Arguments

`string`	the string to be extracted from
`len`	starting position to extract
`pstring`	length of extraction

Example

```
SELECT    SUBSTR('ORACLE',2,3) …    = RAC
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

INSTR

Returns the starting position of string within another string.

Syntax

```
INSTR(string,search)
INSTR(string,search,pos,n)
```

Arguments

`string`	The string to be searched
`search`	The search string
`pos`	Start position of search
`n`	Finds the *nth* occurrence

Example

```
SELECT    INSTR('Oracle','cle') …   = 4
```

LTRIM & RTRIM

The ltrim and rtrim functions remove portions of a string from the left or right

Syntax

```
LTRIM(string,rem)
RTRIM(string,rem)
```

Arguments

`string`	The string you wish to modify
`rem`	The string to be removed. All occurrences are removed

Example

```
SELECT    LTRIM('OOracle,'O') … = racle
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

LENGTH

The length function returns the number of characters in a string.

Syntax

```
LENGTH(string)
```

Arguments

`string` The string you want the length of

Example

```
SELECT    LENGTH('Oracle) …   = 6
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

TRANSLATE

The TRANSLATE function searches through a string for a character, and replaces it with another.

Syntax

```
TRANSLATE(string,from,to)
```

Arguments

string	The string you wish to modify
from	Searches for this character
to	Replaces with this character

Example

```
SELECT    TRANSLATE('hekko','k','l') … = hello

SELECT    TRANSLATE('gekko','gk','hl') … = hello
```

REPLACE

The `REPLACE` function searches through a string for another string and replaces all occurrences of it with another string

Syntax

```
REPLACE(string,search,replace)
REPLACE(string,search)
```

Arguments

`string`	The string you wish to modify
`search`	Searches for this string
`replace`	Replaces with this string. If `replace` is omitted then `search` is removed from `string`

Example

```
SELECT REPLACE('orafred','fred','cle') … = oracle

SELECT    REPLACE('oracleab','ab') … = oracle
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Number functions

We will look at the following number functions:

- ROUND
- TRUNC
- SIGN
- CEIL & FLOOR
- Mathematical functions

Copyright © 1999 – 2004 ApplTop Solutions Limited

ROUND

The ROUND function rounds a number to a specified number of decimal places.

Syntax

```
ROUND(number,n)
```

Arguments

number	The number you want to round
n	The number of decimal places: if n is negative then the number to the left of the decimal point is rounded

Example

```
SELECT ROUND(10.25,1) …  = 10.3

SELECT ROUND(10.25,-1) … = 10
```

TRUNC

The `TRUNC` function truncates a number to a specified number of decimal places.

Syntax

```
TRUNC(number,n)
```

Arguments

`number`	The number you want to truncate
`n`	The number of decimal places

Example

```
SELECT TRUNC(10.25,1) …  = 10.2
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

SIGN

The `SIGN` function returns -1 if a number is negative, 0 if a number is zero and +1 if a number is positive.

Syntax

```
SIGN(number)
```

Example

```
SELECT SIGN(10),SIGN(-100) …  = 1 and -1
```

CEIL & FLOOR

The CEIL and FLOOR functions return the largest or smallest integers which are greater or smaller than a specified number.

Syntax

```
CEIL(number)
FLOOR(number)
```

Example

```
SELECT CEIL(10.25) …      = 11

SELECT FLOOR(10.25) …     = 10
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Mathematical Functions

There are more number functions available. Here is a list of some of them:

Function	Arguments	Returns
`power`	m n	Raises m to the power n
`mod`	m n	Returns remainder of m divided by n
`abs`	m	Returns absolute value of m
`sqrt`	m	Square root on m
`log`	m n	Logarithm, base m of n
`sin`	n	Sine of n
`sinh`	n	Hyperbolic sine of n
`tan`	n	Tangent of n
`tanh`	n	Hyperbolic tangent on n
`cos`	n	Cosine of n
`cosh`	n	Hyperbolic cosine of n
`exp`	n	e raised to the nth power where e=2.71828183
`ln`	n	Natural logarithm on n, where n is greater than zero

Copyright © 1999 – 2004 ApplTop Solutions Limited

Oracle Dates

Dates in Oracle are stored as a number which represents the following:

- Century
- Year
- Month
- Day
- Hours
- Minutes
- Seconds

The default display format is `DD-MON-RR` (or `DD-MON-YY`) which represents a 2 digit day, followed by a 3 character month, and ending in a 2 digit year: for example, 10-JUL-99.

Current Date & Time

There is a special pseudo column available in Oracle called `SYSDATE`. This returns current date and time, for example:

```
SELECT SYSDATE …
```

Date Arithmetic

Arithmetic operators can be performed on dates. The following table describes how this works:

Operation	Result Type	Description
date + number	date	Adds number of days to date
date - number	date	Subtracts number of days from date
date - date	number of days	Subtracts one date from another
date + number/24	date	Adds number of hours to date

Date Arithmetic Examples

To find the date 10 days ago:

```
SELECT SYSDATE-10
FROM dual;
```

The above example SELECTs from a dummy table called "dual" - this is simply a table containing a single column and a single row. It is useful to SELECT from when you want a single row to be returned to allow you to use a function or pseudo column.

Date functions

We will look at the following Date functions:

- MONTHS_BETWEEN
- ADD_MONTHS
- NEXT_DAY
- LAST_DAY
- ROUND
- TRUNC

MONTHS_BETWEEN

This function returns the number of months between two dates. The non-integer part of the result represents a portion of a month.

Syntax

```
MONTHS_BETWEEN(date1,date2)
```

Example

```
SELECT    MONTHS_BETWEEN(sysdate,sysdate-30) … = 1
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

ADD_MONTHS

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

Syntax

```
ADD_MONTHS(date,mon)
```

Arguments

`date`	The date you are adding too
`mon`	The number of months to add. `mon` can be negative

Example

```
SELECT    ADD_MONTHS(sysdate,2) …
```

would add 2 months to the current date

Copyright © 1999 – 2004 ApplTop Solutions Limited

NEXT_DAY

The `NEXT_DAY` function is used to find the next date of the specified day after a specified date.

Syntax

```
NEXT_DAY(date,day)
```

Arguments

`date`	The starting date
`day`	A string representing the day you are looking for, i.e. Monday, Tuesday, etc. Can also be given as a number where 1 is Sunday and 7 is Saturday.

Example

```
SELECT    NEXT_DAY(SYSDATE,'Monday') …
```

would return the first Monday after today's date.

Copyright © 1999 – 2004 ApplTop Solutions Limited

LAST_DAY

The LAST_DAY function is used to find the last day of the month which contains a specified date.

Syntax

```
LAST_DAY(date)
```

Example

```
SELECT    LAST_DAY(sysdate) …
```

would find the last day of the current month

Copyright © 1999 – 2004 ApplTop Solutions Limited

ROUND (date version)

The `ROUND` function is used to round dates to the nearest month or year.

Syntax

```
ROUND(date,what)
```

Arguments

`date` The date to be rounded

`what` Can either be MONTH or YEAR - if omitted then time element is set to 12:00:00am (useful for comparing dates with different times).

Example

```
SELECT    ROUND(sysdate,'YEAR') …
```

will return the first day of the current year

TRUNC

Very similar to the `ROUND` function, but `TRUNC` effectively always rounds down.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Conversion Functions

When you are selecting columns from a table, specifying literals or using the results of functions, you are working with specific datatypes. There will be times when you need to mix and match datatypes, and you do this using conversion functions. We will cover the following conversion functions:

- TO_CHAR
- TO_NUMBER
- TO_DATE

Copyright © 1999 – 2004 ApplTop Solutions Limited

TO_CHAR

The TO_CHAR function is used convert a value into a char, with or without a specified format.

Syntax

```
TO_CHAR(number)
TO_CHAR(number,format)
TO_CHAR(date)
TO_CHAR(date,format)
```

Arguments

number	The number you want to convert to a char
date	A date you want to convert to a char
format	The format mask you wish to apply to the resulting char. Many format masks are available.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Common Format Masks

Date Format Masks

Format Mask	Meaning
YYYY,YYY,YY,Y	Displays year in 4, 3, 2 or 1 digits
RR	Returns a year according to the last two digits of the current year and the 2 digit year passed to the to_char function
MON,MONTHS,MM	3 digit spelled month, full month spelling or 2 digit month number
Q	Quarter of year
DY,DAY,DDD,DD,D	3 letter spelled day, fully spelled day, day of year, day of month or day of week
WW,W	Week of month or year

Time Format Mask

Format Mask	Meaning
HH,HH12,HH24	Hour of day, Hours 1-12 or Hours 1-24
MI	Minute
SS	Second
SSSSS	Seconds since midnight

Common Format Masks

Number Format Masks

Format Mask	Meaning
9	Numeric position, number of 9's determine width
0	Same as 9 except leading 0's are displayed
$	Floating dollar sign
.	Decimal point position specified

Examples of TO_CHAR

Convert a number to a char:

```
TO_CHAR(10)
```

Convert a date to a char, and display as DD-MON-YYYY:

```
TO_CHAR(SYSDATE,'DD-MON-YYYY')
```

Convert a number to a char and display as a 5 digit char:

```
TO_CHAR(number,'99999')
```

Convert a date to a char and display only the time:

```
TO_CHAR(SYSDATE,'HH:MI:SS')
```

List employee salaries as a char with some leading text:

```
SELECT 'Salary='||TO_CHAR(sal,'9990.00') …
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

TO_NUMBER

The `TO_NUMBER` function is used convert a char into a number.

Syntax

```
TO_NUMBER(string)
```

Example

```
SELECT TO_NUMBER('10')  …
```

TO_DATE

The TO_DATE function is used convert a char into a date.

Syntax

```
TO_DATE(string)
TO_DATE(string,format)
```

Arguments

string	The string to be converted
format	The format mask you wish to apply to the input string: this ensures that the string is in a correct date format. If format is omitted then the default date format (usually DD-MON-RR) is used.

Example

```
SELECT TO_DATE('10-JUL-1999','DD-MON-YYYY') …
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Functions That Accept Any Kind of Datatype

We will look at the following functions:

- `NVL`
- `GREATEST` & `LEAST`
- `DECODE`

Copyright © 1999 – 2004 ApplTop Solutions Limited

NVL

The `NVL` function returns a specified value if another is NULL.

Syntax

```
NVL(value,new value)
```

Arguments

`value`	The value you wish to check for null
`new value`	Returns this if `value` is null

Examples

```
SELECT     NVL(mgr,'No manager') …

SELECT     NVL(comm,0) …
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

GREATEST & LEAST

These two functions return either the greatest or least from a list of values.

Syntax

```
GREATEST(value1,value2, …)
LEAST(value1,value2, …)
```

Arguments

`valuen`	Makes up list of values

Examples

```
SELECT     GREATEST(10,20,50,40) …
```

will return 50, whereas

```
SELECT     LEAST(10,20,50,40) …
```

will return 10

Copyright © 1999 – 2004 ApplTop Solutions Limited

DECODE

The DECODE function is very powerful. It works like an IF statement, but can be embedded within a SQL statement.

Syntax

```
DECODE(   value,
     ,    search1, result1
    [,    search2, result2 . . .]
     ,    default)
```

Arguments

value	The value to be evaluated
search	The value to search for
result	Returns value if a match is found
default	Returns this if no match is found

Examples

To display a percentage based on salary grade:

```
SELECT DECODE( salgrade
          ,    1,'15%'
          ,    2,'10%'
          ,    3,'8'
          ,    '5%' ) bonus …
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Nesting Functions

There will be times when you need to perform two or more functions on a single value. The second function may depend on the result of the first, and so on: you can do this kind of thing by nesting functions.

As a simple example, let's say that you want to list all employees, and that you want the manager column to contain some readable text if it is null. You might at first try:

```
SELECT NVL(mgr,'NO MANAGER') ..
```

This would produce an error because the datatypes do not match (`mgr` is a number, 'NO MANAGER' is a char). The solution would be to convert `mgr` to a char first:

```
SELECT NVL(TO_CHAR(mgr),'NO MANAGER')) …
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Group Functions

Single row functions act upon each row returned from a query. Group functions, on the other hand, act on sets of rows returned from a query. This set can be the whole table or the table split into smaller groups.

A table is split into smaller groups with the `GROUP BY` clause. This appears after the `WHERE` clause in a `SELECT` statement.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Group Functions

There are many group functions available:

Function	Value Returned
AVG(n)	Returns average on n, ignoring nulls
COUNT(n|*)	Returns number on non-null rows using column n. If * is used then all rows are counted
MAX(expr)	Maximum value of expr
MIN(expr)	Minimum value of expr
STDDEV(n)	Standard deviation of n, ignoring nulls
SUM(n)	Sum of n, ignoring nulls
VARIANCE(n)	Variance of n, ignoring nulls

Notes

n can be prefixed with the keyword `DISTINCT` - this will make the group function only work on unique values of the column specified by `n`.

Examples of using group functions

To find total paid in salaries for all employees:

```
SELECT SUM(sal) …
```

To find highest, lowest and average salary:

```
SELECT MAX(sal), MIN(sal), AVG(sal) …
```

To find total paid in salaries for all employees in department 20:

```
SELECT SUM(sal)
FROM emp
WHERE department = 20;
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Grouping Data

You can split the data in a table into smaller groups, you can then use Group Functions to return summary information about each group. You split a table using the GROUP BY clause.

The GROUP BY clause instructs the query to return rows split into groups determined by the specified columns. GROUP BY generally takes the following form:

```
SELECT     job
     ,     AVG(sal)
FROM       emp
WHERE      deptno = 20
GROUP BY   job;
```

The above statement will return the average salary for each job for employees who work in department 20. The data has been grouped by the job column, the AVG group function has then returned summary data based on all rows in the table that are in the current group.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Grouping Data

When grouping data, you should be aware of the following:

- You are not restricted to a single column. You can group by as many columns as you like, as long as the columns you are grouping by are in the `SELECT` clause
- Rows can be omitted from the grouped data by using the `WHERE` clause
- Groups can be omitted from the results by using the `HAVING` clause
- When grouping data and using group functions, you must ensure all columns in the `SELECT` clause that do not use group functions are included in the `GROUP BY` clause, otherwise an error will occur

Copyright © 1999 – 2004 ApplTop Solutions Limited

Grouping Data - Omitting Groups

You can omit groups returned from a query which use a GROUP BY clause by using the HAVING clause. HAVING generally takes the form:

```
SELECT     job
     ,     AVG(sal)
FROM       emp
WHERE      department = 20
GROUP BY   job
HAVING     AVG(sal) > 1000;
```

The above statement will return the average salary for each job for employees who work in department 20 where the average salary is greater than 1000.

The HAVING clause can include anything that appears in the SELECT clause. You generally include group functions in the HAVING clause rather than just column names, because if you had a HAVING clause as follows:

```
HAVING job <> 'CLERK'
```

this would correctly omit all CLERKS; but this is a very inefficient way to do it, and by the time the HAVING clause is evaluated, the rows with CLERKS have already been retrieved. It would be much better to omit CLERKS using the WHERE clause.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Summary

In this section we have covered:

- Row Functions
  - Row functions act on each row returned
  - Character functions
  - Number functions
  - Oracle dates & date functions
  - Conversion functions & format masks
  - Functions that accept any datatype
  - Nesting functions
- Group Functions & Grouping Data
  - Group functions act on a group of rows
  - Many different group functions
  - Grouping Data
  - Omitting groups

Copyright © 1999 – 2004 ApplTop Solutions Limited

What is Next?

In the next section we take a look how to select data from more than one table at a time. We cover joins, set operators and subqueries.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Lab 3

1. List all employees, give each a 15% pay increase and display new salary always rounded down to the nearest whole number. Display name, old salary and new salary.
2. List all employee names in upper and lower case, also display the name with the first character in upper and the rest in lower.
3. Create a list that displays the first 2 digits of the employee name, followed by the empno, followed by the rest of the name, display as a single column.
4. Generate the following output:

```
EMPLOYEES
-----------------------
Smith  is 5 digits long
Allen  is 5 digits long
Ward   is 4 digits long
Jones  is 5 digits long
Martin is 6 digits long
Blake  is 5 digits long
. . . . .
```

5. Display the name, hiredate and number of whole months each employee has been employed. Show the highest first.
6. List each employee name along with the salary if it is more than 1500, if it is 1500 print 'On Target', if it is less than 1500 print 'Below 1500'.
7. Write a query that will display the day of the week for the current date.
8. List the maximum, minimum and average salaries for all employees.
9. List the total salary bill for each job type.
10. Display a count of how many CLERKS there are.
11. Find all departments that have more than 3 employees.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Section Four
Querying More Than One Table

Copyright © 1999 – 2004 ApplTop Solutions Limited

Querying More Than One Table

There will be times when you need to select data from more than one table at a time. This section covers all the basics of doing this, and we will cover:

- Joins
- Set Operators
- Subqueries

Copyright © 1999 – 2004 ApplTop Solutions Limited

Joins

So far, any queries we've seen have been from a single table at a time - but SQL allows you to query many tables at the same time through the use of joins. We will now cover some of the basics of joining tables within a `SELECT` statement. We will look at the following types of join:

- Product
- Equi join
- Non-equi join
- Outer join
- Self join

The `WHERE` clause is used to construct a join.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Joins - Product

If you construct a `SELECT` statement which contains information from two or more tables without specifically linking any of the columns from one table to the next, the resulting query would be what is known as a product (sometimes referred to as a Cartesian join). This basically means that ALL rows from ALL tables are returned in EVERY combination. So for example, lets say we have two tables - "emp" with 14 rows and "dept" with 4 rows - and we entered the following statement:

```
SELECT    dname,ename
FROM      emp,dept;
```

The above query would return all rows from both tables in all combinations, resulting in a total of 56 rows being returned (14 * 4).

You will very rarely need to perform this kind of query, but it is mentioned so that you are aware of the result of a product join. If you unintentionally create one then the results could be very different to what you might expect: imagine 2 tables, with over 1,000,000 rows each!!

Copyright © 1999 – 2004 ApplTop Solutions Limited

Joins - Equi

An equi join is a join which directly links columns from one table to another, or in other words, an equi join joins tables where a column on one table is equal to a column from another table. As an example, let's say you have the following statement:

```
SELECT    ename,deptno
FROM      emp;
```

This is okay, but what if instead of displaying the department number (deptno), you wanted to display the department name. You would have to create an equi join from the emp table to the dept table:

```
SELECT    emp.ename,dept.dname
FROM      emp,dept
WHERE     dept.deptno = emp.deptno;
```

The above statement joins the emp and dept tables using the deptno column, and in English the statement reads: select the ename column from the emp table and get the dname column from the dept table, only select dept rows where the deptno on dept is the same as the deptno on the emp table.

Table Aliases

As a rule, when selecting data from more than one table, you should qualify the column names you use with the table name. This removes any ambiguity if there are duplicate column names across different tables: you do this by prefixing a column name with the table name followed by a dot (.):

```
SELECT TABLE.COLUMN …
```

A better method of this is to use a table alias. A table alias is very similar a column alias, in that it is a method of renaming a table only for the purposes of the query - you can then use this alias within your query as if the table were actually called by the alias. This can save a lot of typing, make SQL easier to read and allowing for self joins (covered later). The syntax for a table join is simple: just follow the table name with the alias:

```
SELECT    e.ename,d.dname
FROM      emp e,dept d
WHERE     e.deptno = d.deptno;
```

Qualifying column names can also improve performance of your code because you are telling the system exactly where to find the column.

Joins - Non Equi

A non-equi join is used where a value (column) is within a range of values rather than equal to a specific value. As an example, the relationship between the emp and salgrade table is an equi join, in that no column on the emp table corresponds directly to a column on the salgrade table. The link between the two tables is that the sal column on emp must be BETWEEN two values found on salgrade.

```
SELECT    e.ename,e.sal.s.grade
FROM      emp e,salgrade s
WHERE     e.sal BETWEEN s.losal AND s.hisal;
```

The statement reads: select the ename and sal columns from the emp table and get the grade column from the salgrade table, only select salgrade rows where the sal column on emp is between the losal and hisal columns on salgrade. As a general rule, when joining tables, you need 'number of tables minus 1' join conditions, so to join three tables you would require at least two join conditions.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Joins - Outer

An outer join allows you to join tables together and still return rows even if one side of a condition is not satisfied. For example, the `dept` table has 4 departments (10, 20, 30 and 40), and the `emp` table has employees in all departments except 40, so if you were to write some SQL to join the two tables together using a standard equi-join, the row from `dept` which does not appear in `emp` would not be in the returned rows. You can use an outer join to get around this problem.

```
SELECT    e.ename,d.dname
FROM      emp e,dept d
WHERE     d.deptno = e.deptno(+);
```

The `(+)` in the above statement creates an outer join. It basically says: still return a row from `dept` even if the join condition fails. When creating outer joins, you must put the `(+)` on the side of the condition where no data will be found - in this case, when SQL has retrieved department 40 from `dept`, it will not find any rows on `emp` for department 40, but the `dept` row is still returned.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Joins - Self

By using a self join with table aliases you can join a table to itself. A self join basically allows you to select from the same table more than once within the same SQL statement - this is very useful if a table has rows on it which relate to other rows on the same table. For example, the `emp` table holds employees, and each employee has a manager (except the big boss). This manager is stored on the same table: so, you would need a self join if you wanted to create a statement that listed all employee names along with their manager name.

```
SELECT     e.ename    employee_name
     ,     m.ename    manager_name
FROM       emp e,emp  m
WHERE      m.empno  = e.mgr;
```

The above statement says: select the employee name from `emp`, and call it `employee_name`, then select the employee name again and call it `manager_name` from `emp` where the employee number (`empno`) is the same as the manager (`mgr`) stored on the first record.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Set Operators

We covered the concepts of Set Operators in the introduction to Relational Databases at the start of this course. So far we have covered:

- restriction with the `WHERE` clause
- projection with the `SELECT` clause,
- joins
- product

We will now cover:

- `UNION`
- `INTERSECT`
- `MINUS`

Copyright © 1999 – 2004 ApplTop Solutions Limited

Set Operators

`UNION`, `INTERSECT` and `MINUS` set operators are used when you need to construct two or more queries and you want to see the results as if it were a single query.

The queries you use could be totally different, i.e. different tables, or they could be using the same table but be using different `WHERE` or `GROUP BY` clauses.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Set Operators - UNION

The UNION set operator combines the results of two or more queries and returns all distinct rows from all queries. It takes the form:

```
SELECT    job
FROM      emp
WHERE     deptno=10
UNION
SELECT    job
FROM      emp
WHERE     deptno=30;
```

The above statement would, first of all, select all jobs from emp where the department is 10, then select all jobs from emp where the department is 30. The results of both these queries are combined and only distinct rows are returned.

UNION ALL

Instead of using UNION, you could use UNION ALL which would return ALL rows from both queries.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Set Operators - INTERSECT

The INTERSECT set operator combines the results of two or more queries and returns only rows which appear in BOTH queries. It takes the form:

```
SELECT    deptno
FROM      dept
INTERSECT
SELECT    deptno
FROM      emp;
```

The above statement would first of all select all rows from dept, then all rows from emp, and only where the data is found in BOTH queries would the data be returned. This is effectively saying: select all department numbers where employees can be found

Set Operators - MINUS

The MINUS set operator combines the results of two or more queries and returns only rows that appear in the first query and not the second. It takes the form:

```
SELECT    deptno
FROM      dept
MINUS
SELECT    deptno
FROM      emp;
```

The above statement would first of all select all rows from dept, then all rows from emp - data would be returned if it was found in the first query and not the second. This is effectively saying: select all department numbers which have no employees.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Set Operators - Rules

There are a number of rules you should follow when using Set Operators:

- You must `SELECT` the same number of columns in each query
- All corresponding columns MUST be of the same datatype
- Duplicate rows are ALWAYS eliminated (except when using `UNION ALL`)
- Column names are derived from the first query
- Queries are executed from top to bottom
- Can use multiple set operators at the same time
- Can include an `ORDER BY` at end of last query. A useful way to specify columns in an `ORDER BY` is by using the column position rather than the name, for example, to sort the output from a Set Operation by the first column:

```
SELECT deptno FROM dept
UNION
SELECT deptno FROM emp
ORDER BY 1;
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Subqueries

A sub query is basically a `SELECT` statement within another `SELECT` statement; they allow you to select data based on unknown conditional values. A subquery generally takes the form:

```
SELECT column(s)
FROM   table(s)
WHERE  column(s)  = (SELECT column(s)
                     FROM table(s)
                     WHERE condition(s) );
```

The subquery is the part in bold and in brackets: this part of the query is executed first, just once, and its result is used in the main (outer) query.

Single Row Subqueries

A single row subquery is the simplest form of sub query. It returns a single row to the outer query that in turn uses the result to complete itself.

For example, to find all employees who earn the lowest salary in the company we could use a subquery. If we think about what is needed first:

- we need to determine what the lowest salary is
- we need to select all employees who earn this amount

The finished query would be as follows:

```
SELECT ename,sal
FROM   emp
WHERE  sal = (SELECT MIN(sal)
              FROM   emp);
```

The above statement executes the subquery first to find the lowest salary and then it uses the single row result of that query to find all employees who earn that amount.

Single Row Subqueries

As another example, let's say we wanted to find all employees who have the same job as BLAKE:

```
SELECT ename,job
FROM   emp
WHERE  job = (SELECT job
              FROM emp
              WHERE ename = 'BLAKE');
```

The above statement executes the subquery first to find what job BLAKE has, and then it uses the single row result of that query to find all employees who have the same job.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Multiple Row Subqueries

A subquery can return more than one row, but you must use a multi-row comparison operator (such as IN) in the outer query or an error will occur:

```
SELECT ename,sal,deptno
FROM   emp
WHERE  (deptno,sal)
            IN (SELECT  deptno,MIN(sal)
               FROM     emp
               GROUP BY deptno);
```

The above statement executes the subquery first to find the lowest salary in each department (by using a GROUP BY), then it uses each row returned from that query to find all employees who earn that amount in each department.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Multiple Row Subqueries

ANY/SOME Operator

The ANY (SOME) operator compares a value to EACH row returned from the subquery.

```
SELECT ename,sal,job,deptno
FROM   emp
WHERE  sal > ANY
            (SELECT DISTINCT SAL
             FROM   emp
             WHERE  depton = 30);
```

The above statement executes the subquery first to find all distinct salaries in department 30, and the > ANY part of the outer query says where the sal column in greater than ANY of the rows returned by the subquery. This effectively says: list all employees whose salary is greater than the lowest salary found in department 30.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Multiple Row Subqueries

ALL Operator

The ALL operator compares a value to ALL rows returned from the subquery.

```
SELECT ename,sal,job,deptno
FROM   emp
WHERE  sal > ALL
            (SELECT DISTINCT SAL
             FROM   emp
             WHERE  deptno = 30);
```

The above statement executes the subquery first to find all distinct salaries in department 30, the > ALL part of the outer query says where the sal column is greater than ALL of the rows returned by the subquery. This effectively says: list all employees who earn more than everyone in department 30.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Subqueries

When constructing subqueries you should be aware of the following:

- ORDER BY can appear in the outer query only
- You can nest subqueries to a level of 255 (you never would!)
- Subqueries can be used with the HAVING clause, for example, to list the departments which have an average salary bill greater than the average salary bill for department 30:

```
SELECT    deptno,AVG(sal)
FROM      emp
GROUP BY  deptno
HAVING    AVG(sal) >
               (SELECT AVG(sal)
                FROM   emp
                WHERE  deptno=30);
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Correlated Subqueries

A correlated subquery is a way of executing a subquery once for each row found in the outer query. A correlated subquery works as follows:

1. Get a candidate row from outer query
2. Execute subquery using candidate row data
3. Use values from subquery to either include or exclude candidate row
4. Continue until no more candidate rows are found in the outer query.

A correlated subquery is identified by the use of an outer query column within the subquery. They are useful when you need the subquery to return different results based on the outer query.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Correlated Subqueries

As an example, let's say we wanted to list all employees who earn a salary greater than the average for their department. A standard subquery would not work because the rows returned from the subquery will not be related (correlated) to the rows in the outer query. Using a correlated subquery you would have something like:

```
SELECT empno,ename,sal,deptno
FROM emp e
WHERE sal > (SELECT avg(sal)
             FROM emp
             WHERE deptno = e.deptno);
```

The above query says: select all employees as candidate rows, then find the average salary for the current department, using this average, either include or exclude the candidate row.

The key to this working is the table alias used in the outer query - we need to ensure the emp table is called something different in both queries, otherwise we would not be able to link the columns together.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Correlated Subqueries

The EXISTS Operator

You can use the EXISTS operator with a correlated subquery: this is used to determine if any rows are returned from the subquery. If any are returned then the condition is true and the row in the outer query is returned. For example, to select all employees who manage someone,…

```
SELECT empno,ename,job,deptno
FROM   emp e
WHERE  EXISTS (SELECT 'FOUND A ROW'
               FROM   emp
               WHERE  emp.mgr = e.empno);
```

The above query says: select all employees as candidate rows, then find an employee whose manager is the employee in the outer query, and if an employee was found in the subquery then return the row in the outer query.

NOT EXISTS

You can use NOT EXISTS operator to check if NO rows are returned from the subquery. If you used a NOT EXISTS in the above example, you would get all employees who do not manage anyone.

Summary

In this section we have covered a lot of ground: we have looked at some of the more complex types of queries. We briefly covered ways in which you can select data from more than one table at a time.

- Joins - Product, Equi, Non-equi, Outer and Self
- Set Operators - `UNION`, `INTERSECT` and `MINUS`
- Subqueries - Single Row, Multi Row and Correlated

Copyright © 1999 – 2004 ApplTop Solutions Limited

What is Next?

In the next section we take a quick look at using DML and DDL to modify data within the database, and how to change the structure of the database.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Lab 4

1. List all employee names along with the name of the department they work in.
2. List all employee names along with the department name, department number and the location.
3. Produce a list of employees with their salary and salary grade.
4. Display departments with no employees.
5. List all employees, along with their managers name.
6. The query produced for question 5 will probably not have listed employees who have no manager, so change the query to show these. Display 'NO MANAGER' if no manager is found.
7. Find all employees who earn the highest for each job type.
8. List all employees with a * against the mostly recently hired one (use the UNION operator).

Copyright © 1999 – 2004 ApplTop Solutions Limited

Section Five
Modifying Data & the Database

Copyright © 1999 – 2004 ApplTop Solutions Limited

Modifying Data & the Database

So far, all of the SQL we have looked at has been to do with querying data, using the `SELECT` statement. Now we are ready to take a look at other types of SQL commands, DML and DDL. DML is SQL's Data Manipulation Language: it is used to modify data held within the database. DDL or Data Definition Language is used to modify the structure of the database. In this section we cover the following:

- Using DML
    - Inserting new data
    - Updating existing data
    - Deleting data
    - Transaction Processing
- Using DDL
    - Tables
    - Indexes
    - Synonyms
    - Privileges
    - Views & Sequences

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DML
Inserting New Data

If you need to create new data within the database, you use the INSERT statement. This allows you to create new rows on any table (as long as you have the correct privileges).

INSERT

SQL statement used to insert rows into the database

Syntax

```
INSERT INTO table   [ (column,column,……) ]
     VALUES         (value,value,…..);
```

Example

```
INSERT INTO dept (deptno,dname,loc)
     VALUES      (50,'MARKETING','DAVENTRY');
```

The above statement says: insert a new row into the dept table, set the deptno column to 50, dname to MARKETING and loc to DAVENTRY.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Inserting New Data

The INSERT statement can be used in a number of different ways:

- Specify only required columns - If you want to insert a row but do not want to insert values into certain columns, then simply omit the column name from the `INSERT` statement (unless there is a `NOT NULL` constraint on the column).
- Specify no columns at all - If you want to always insert a value into all columns, then you do not need to give the column names - just ensure the `VALUES` clause matches the table exactly (not recommended).
- You can insert rows into a table based on rows from another table.
- Almost anything that can appear in a `SELECT` clause of a `SELECT` statement can also appear in the `VALUES` clause of the `INSERT` statement.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Inserting New Data - Examples

To insert a row into the `dept` table only specifying 2 columns:

```
INSERT INTO dept (deptno,loc)
     VALUES      (60,'DAVENTRY');
```

To insert a row using rows from another table:

```
INSERT INTO dept(deptno,dname)
     (SELECT deptno,dname
      FROM   old_dept);
```

To insert a row using rows from another table and utilising a row function:

```
INSERT INTO dept (deptno,dname)
     (SELECT deptno,INITCAP(dname)
      FROM   old_dept);
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DML
Updating Existing Data

If you need to change some data within the database, you use the UPDATE statement. This allows you to change a single row or many rows at the same time (as long as you have the correct privileges).

UPDATE

SQL statement used to update rows in the database

Syntax

```
UPDATE table [alias]
SET    column [,column…] =
       {expression,subquery}
[WHERE condition];
```

Example

```
UPDATE emp
SET    sal = sal * 1.1
WHERE  job = 'CLERK';
```

The above statement says: find all employees whose job is CLERK and set their salary to itself multiplied by 1.1 - or in other words, give all clerks a 10% pay increase.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Updating Existing Data

When using the `UPDATE` statement, you should be aware of the following:

- If the `WHERE` clause is omitted then ALL rows on the table will be updated.
- The `WHERE` clause can contain anything that would normally appear in the `WHERE` clause for the `SELECT` statement.
- It is possible to use subqueries and correlated subqueries in the `SET` clause.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Updating Existing Data - Examples

To give the employee SCOTT a new job, change of department and a pay increase:

```
UPDATE emp
SET    job = 'SALESMAN'
,      sal = sal * 1.25
,      deptno = 40
WHERE  ename = 'SCOTT';
```

To set the NUMBER_OF_EMPLOYEES on the dept table (using a correlated subquery):

```
UPDATE    dept d
SET  number_of_employees =
               ( SELECT count(*)
                 FROM   emp e
                 WMERE  e.deptno = d.deptno)
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DML
Deleting Data

If you need to delete data within the database, you use the DELETE statement. This allows you to delete a single or many rows at once (as long as you have the correct privileges).

DELETE

SQL statement used to delete rows from the database

Syntax

```
DELETE [FROM] table
[WHERE condition];
```

Example

```
DELETE emp
WHERE  job = 'MANAGER';
```

The above statement says: delete all rows from the emp table where the job column is MANAGER - or in other words, give all manager the boot.

It is also possible to use subqueries with the DELETE statement.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DML
Deleting Data

Another way to remove all the data from a table is using the `TRUNCATE TABLE` command.

TRUNCATE TABLE

SQL statement used to remove ALL rows from a table

Syntax

```
TRUNCATE TABLE table
```

Example

```
TRUNCATE TABLE emp;
```

The `TRUNCATE TABLE` command effectively flags the table internally as empty without actually deleting any rows. It is VERY fast and is useful for clearing out tables with many hundreds of thousands or million of rows.

Truncates are DDL, not DML. A truncate moves the High Water Mark of the table back to zero. No row-level locks are taken, no redo or rollback is generated. All extents except those defined in the MINEXTENTS parameter are de-allocated from the table.

By resetting the High Water Mark, the truncate prevents reading of any table's data, so they it has the same effect as a delete, but without the overhead. There is, however, one aspect of a Truncate that must be kept in mind. Because a Truncate is DDL it issues a COMMIT before it acts and another COMMIT afterward so no rollback of the transaction is possible.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Transaction Processing

Oracle ensures the consistency of data through the use of transactions. Transactions give you more control when changing data and will ensure data consistency in the event of a system failure.

A transaction can be thought of as a single consistent change to the database which may directly relate to some kind of business functionality: this change to the database may consist of a single or multiple DML statements. For example, in a banking system, a transfer funds transaction would involve the transfer of funds out of one account and then further transfer into another account - this would require two DML statements but is considered a single transaction. Transaction processing allows you to perform all the DML you need before committing the changes to the database. This ensures that both of DML statements are always complete and in the event of a failure, none of the DML statements would complete.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Transaction Processing

There are two types of transaction:

- DML - holds a number of DML statements
- DDL - holds a single DDL statement

A transaction begins when:

- A DDL command is issued
- First DML statement issued after a `COMMIT`

A transaction ends when:

- A `COMMIT` or `ROLLBACK` command is issued
- DDL command is issued
- You exit SQL*Plus
- System Failure (auto-rollback)

Copyright © 1999 – 2004 ApplTop Solutions Limited

Transaction Processing
The COMMIT command

Whenever you issue a DML statement which changes the data held within the database, you are not actually changing the database. You are effectively putting your changes into a buffer, and to ensure this buffer is flushed and all your changes are actually in the database for others to see, you must first commit the transaction. You can do this with the COMMIT statement.

Syntax

```
COMMIT [WORK];
```

Example

```
UPDATE dept
SET dname = initcap(dname);
COMMIT;
```

The above code will update all dept rows then commit the changes to the database.

After the COMMIT command has run, the following is true:

- Your changes are in the database and permanent
- The current transaction has ended
- Any locks are released

Copyright © 1999 – 2004 ApplTop Solutions Limited

Transaction Processing
The ROLLBACK command

If you have started a transaction by issuing a number of DML statements, but you then decide you want to abort the changes and start again, you need to use the ROLLBACK statement.

Syntax

```
ROLLBACK [WORK];
ROLLBACK [WORK] TO SAVEPOINT_NAME;
```

Example

```
UPDATE dept
SET dname = initcap(dname);

ROLLBACK;
```

The above code will update all dept rows then rollback the changes: this will effectively be as if you had never issued the UPDATE statement.

After the ROLLBACK command has run, the following is true:

- All of your changes will be lost
- The current transaction has ended
- Any locks are released

We cover the SAVEPOINT part next.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Transaction Processing
The SAVEPOINT command

You can split a transaction up into smaller portions using the SAVEPOINT command. SAVEPOINT allows you to specify markers within a transaction, and these markers can be easily rolled back to if needed.

Syntax

```
SAVEPOINT savepoint_name;
```

Example

```
UPDATE dept
SET dname = initcap(dname);
SAVEPOINT done_dept;
UPDATE emp
SET sal = sal * 1.1;
ROLLBACK TO done_dept;
COMMIT;
```

The above code will update all dept rows, create a savepoint and then update all emp rows. Then a ROLLBACK is issued, only back to the last SAVEPOINT - this will effectively discard your changes to the emp table, leaving only the changes to the dept table; the final command will COMMIT any changes left to do - in this case, just the dept changes.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Transaction Processing

Keep in mind the following about transaction processing:

- Other database users will not see your changes until you `COMMIT` them.
- An uncommitted transaction is one which holds locks on the data, for example, if you update a single emp row without committing, other people will still see the unchanged emp row but will not be able to change it themselves until you either issue a `COMMIT` or `ROLLBACK`.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL

DDL is a subset of SQL commands that allows you to make changes to the structure of the database. We will briefly cover the following kinds of database objects:

- Tables
- Indexes
- Synonyms
- Privileges
- Views
- Sequences

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL

When working with any kind of object within the database, there are usually three basic commands that apply to all objects. These are:

- `CREATE` - used to create an object
- `ALTER` - used to alter/change an object
- `DROP` - used to remove an object

Column Types/ Datatypes

All columns on a table must be given a datatype, as this determines what kind of data the column can hold. A few of the more common data types are:

Datatype	Purpose
NUMBER	Holds number data of any precision
NUMBER(w)	Holds number data of w precision
NUMBER(w,s)	Holds number data of w precision and s scale, i.e. 10,2 is a number upto 10 digit in length, with 2 digits after the decimal point.
VARCHAR2(w)	Holds variable length alphanumeric data upto w width.
CHAR(w)	Holds fixed length alphanumeric upto w with.
DATE	Holds data/time data

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL - Tables
The CREATE TABLE Command

To create a new table within the database, you use the `CREATE TABLE` command. In its most basic form, the `CREATE TABLE` command has the following form:

CREATE TABLE

Command used to create tables

Syntax

```
CREATE TABLE table-name
(    column_name type(size)
,    column_name type(size) . . . );
```

Example

```
CREATE TABLE dept
(    deptno     NUMBER
,    dname      VARCHAR2(12)
,    loc        VARCHAR2(12));
```

The above statement will create a table called `dept`, with 3 columns: a number column called `deptno`, a 12 digit `VARCHAR2` column called dname and another 12 digit `VARCHAR2` column called `loc`.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL - Tables
The CREATE TABLE Command

When creating a table, you can ensure that no NULL values are stored in a column by using the `NOT NULL` constraint - for example, to create the dept table and to ensure no NULL values are ever inserted into the deptno column:

```
CREATE TABLE dept
(     deptno     NUMBER     NOT NULL
,     dname      VARCHAR2(12)
,     loc        VARCHAR2(12));
```

DESCRIBE - SQL* Plus command

You can list columns on a table from SQL*Plus using the `describe` command (or `desc`) - for example:

```
SQL> desc dept
Name                      Null?    Type
------------------------- -------- ----
DEPTNO                    NOT NULL NUMBER(2)
DNAME                              VARCHAR2(14)
LOC                                VARCHAR2(13)
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL - Tables
The DROP TABLE Command

A table can be removed from the database using the DROP TABLE command.

```
DROP TABLE dept;
```

Be aware that once a table has been dropped, it cannot be recovered. Also, ALL data on the table is removed.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL - Tables
The ALTER TABLE Command

A table can be altered using the ALTER TABLE command. This command allows you to do many things - we will only look at how you can add extra columns to a table. Let's say you want to add the column 'comments' to the emp table:

```
ALTER TABLE emp
     ADD    (COMMENTS VARCHAR2(80));
```

This will add the column COMMENTS to the emp table.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL - Indexes

An index is a data structure within the database that allows you to provide quick access to data on a table via a particular column or columns. It can also serve as a method of ensuring no duplicate records can be stored on a table. For example, a common column to use when querying the emp table is empno: this column should also be unique since no two employees should have the same number.

When you query the emp table (for example) using the empno column, if no index was present on this column then the database would have to sequentially go through all records on the table until the required one was found. An index would provide a faster method of querying this table.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL - Indexes
The CREATE INDEX Command

You create indexes with the CREATE INDEX command. In it most basic form, it is as follows:

```
CREATE [UNIQUE] INDEX index_name
     ON table_name
     (column [,column . . .]);
```

Examples

To create a unique index called emp_idx01 on the emp table using the empno column:

```
CREATE UNIQUE INDEX emp_idx01
     ON emp
     (empno);
```

To create a non-unique index on the emp table using the ename column:

```
CREATE UNIQUE INDEX emp_idx01
     ON emp
     (empno);
```

To create a non-unique index on the emp table using the ename and hiredate columns:

```
CREATE UNIQUE INDEX emp_idx01
     ON emp
     (empno,diredate);
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL - Synonyms
The `CREATE SYNONYM` Command

Within an Oracle Database, there is an object called user. This is effectively a space within the database for a particular user. The user of that database user would create tables belonging to this user. If another user wanted to access these tables, they would normally have to qualify the table name with the user name first - for example, if user Bob creates a table called `emp`, and user `Dave` wanted to list all rows on `Bob's emp` table, he would have to enter:

```
SELECT *
FROM   bob.emp;
```

A much better method of accessing the table would be to only have to specify the table name. You can do this if a synonym under the `Dave` user exists.

```
CREATE SYNONYM emp
     FOR bob.emp;
```

Now `dave` can see `bob's` emp table by just entering:

```
SELECT *
FROM   emp;
```

Using DDL - Privileges
The GRANT/REVOKE Commands

As with synonyms, if Bob has a table called emp and Dave wants to query it - or even insert, update or delete it - Dave must first be given the correct privileges to allow him to do this. Many privileges exist, and the most common ones are SELECT, INSERT, UPDATE and DELETE - so to give Dave SELECT and INSERT privileges on Bobs' emp table, Bob would have to enter:

```
GRANT SELECT,INSERT ON emp TO dave;
```

A privilege can be removed with the REVOKE command, to remove Dave's INSERT privilege on Bobs' emp table:

```
REVOKE INSERT ON emp FROM dave;
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL - Views
The CREATE VIEW Command

Let's say you have constructed a very useful and complex query and another user says, "I could do with the same kind of query". You could just give them the code and let them run it themselves, but a better method is to create a database view based on your query. A view is basically a virtual table which is made up of the rows that your query returns. For example, if you have a query that lists employee names along with department names, you could create a view as follows:

```
CREATE VIEW emp_dept
AS
     SELECT e.ename employee_name
          , d.dname department_name
     FROM   emp e
     ,      dept d
     WHERE  d.deptno = e.deptno;
```

The above statement would create a view called `emp_dept` which would contain two columns, `employee_name` and `department_name`. You could now access this view as if it were a table:

```
SELECT *
FROM emp_dept;
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL - Views
The DROP VIEW Command

A view can be removed with the `DROP VIEW` command. For example:

```
DROP VIEW emp_dept;
```

The above statement would remove a view called `emp_dept`.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL - Sequences
The CREATE SEQUENCE Command

Assume that you have a new table; it contains a column called sequence_number, and you always want to populate this column the a sequential number. A good way to do this is to create a sequence, then reference the sequence in your INSERT statement. A sequence is simply an object within the database that returns a number, usually the next in sequence. To create a sequence, use the CREATE SEQUENCE command. In its most basic form, it has the following syntax:

```
CREATE SEQUENCE sequence_name
     INCREMENT BY n
     START WITH m;
```

So, to create a sequence called my_seq01 which starts at 100 and increases by 10 each time:

```
CREATE SEQUENCE my_seq01
     INCREMENT BY 10
     START WITH 100;
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Using DDL - Sequences
Referencing a Sequence

You have created a sequence called `my_seq01`: you can reference it in two ways, the first being by using the `NEXTVAL` pseudo column to get the next number:

```
SELECT my_seq01.NEXTVAL
FROM   dual;
```

This would return 100; if you ran this SQL again it would return 110, and so on.

You can also retrieve a sequence's current value without increasing its value with the `CURRVAL` pseudo column:

```
SELECT my_seq01.CURRVAL
FROM   dual;
```

Using DDL - Sequences
The DROP SEQUENCE Command

A sequence can be removed with the DROP SEQUENCE command. For example:

```
DROP SEQUENCE mu_seq01;
```

The above statement would remove a sequence called my_seq01.

Summary

In this section we have covered both how to use DML and DDL, and specifically we have seen:

- Insert, Update and Delete data
- Transaction Processing
- Creating, deleting and altering Tables
- Indexes
- Synonyms
- Privileges
- Views
- Sequences

Copyright © 1999 – 2004 ApplTop Solutions Limited

What is Next?

As far as SQL is concerned, we have pretty much finished: all that is left to do now is take a quick look at some features of SQL*Plus.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Lab 5

1. Write a SQL statement to give all employees a 25% pay increase, commit your changes and view the table.
2. Give all employees who work in NEW YORK an additional 5% pay increase, rollback your changes and view the table.
3. Create a new department called I.T located in ENGLAND.
4. Remove the department you created in question 3.
5. Create a sequence called EMPINFO_SEQ, start it at 10 and increment it by 1.
6. Create a new table called EMPLOYEE_INFO, using the following table as a guide:

Column Name	Datatype & Size
INFO_ID	NUMBER
EMPNO	NUMBER
INFO_DATE	DATE
INFO	VARCHAR2 80

7. Insert a row into the new table, set INFO_ID to be the next value from the sequence you created, set EMPNO to a valid employee number, INFO_DATE to today's date and enter some text into INFO.
8. Create a synonym called einfo for your new table.
9. Select all rows from your new table using the synonym;
10. Remove the new synonym and table.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Section Six
More SQL*Plus

Copyright © 1999 – 2004 ApplTop Solutions Limited

More SQL* Plus

We have already seen a little of SQL*Plus: we will now cover a little more on how to use SQL*Plus and what it can do. We will briefly cover:

- SQL*Plus Variables
- Basic SQL*Plus Reporting
- Saving Output to a file

Copyright © 1999 – 2004 ApplTop Solutions Limited

SQL* Plus Variables

A SQL*Plus variable is basically a way of storing a value in a temporary space. This value can then be referenced from within a SQL statement. We will take a very quick look at:

- Ampersand substitution variables
- Double ampersand substitution variables

There are other types of SQL*Plus variables available which are not covered on this course.

Copyright © 1999 – 2004 ApplTop Solutions Limited

SQL* Plus Variables
Ampersand Variables

A single ampersand substitution variable is a method of asking the user for input before some SQL executes, then substituting the value entered for the reference to the variable within your SQL. Simply prefix a variable name with a single ampersand and SQL*Plus will prompt you to enter a value at runtime. For example:

```
SELECT *
FROM emp
WHERE deptno = &department_number;
```

If you run this SQL, before anything is done, you will be asked the following:

```
Enter value for department_number:
```

You now enter a valid value, and SQL*Plus will substitute &department_number for whatever you entered.

The value entered into a single ampersand variable is not remembered - this means if you execute the same SQL again, you will be asked for a value again.

Copyright © 1999 – 2004 ApplTop Solutions Limited

SQL* Plus Variables
Double Ampersand Variables

A double ampersand substitution variable is almost the same as a single ampersand variable. The only difference is that the value is remembered, so you only need enter a value once.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Basic SQL* Plus Reporting

The results of a query are displayed in a very simple and plain manner. SQL*Plus allows you to add basic formatting to this output to create simple, but effective reports. SQL*Plus reporting is quite a complex topic and beyond the scope of this course, therefore we will only cover the most basic areas; this should be enough to give you a feel for what can be achieved with SQL*Plus. We will cover:

- Adding a title to the page
- Setting break points

Copyright © 1999 – 2004 ApplTop Solutions Limited

Basic SQL* Plus Reporting
Adding a Page Title

You can use the SQL*Plus TTITLE command to add a title to a page. Simply enter

```
SQL> TTITLE 'my title'
```

before you enter your SQL, and the report will be headed with whatever you set the title to.

You can also add a footer to a page with the BTITLE command. For example:

```
SQL> BTITLE 'my footer'
```

<u>Example</u>

```
SQL> TTITLE 'Employee List'
SQL> SELECT * FROM emp;
Sun Jul 18                  page    1

             Employee List

    EMPNO ENAME

--------- ----------

     7369 SMITH

     7499 ALLEN

     7521 WARD

     7566 JONES
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Basic SQL* Plus Reporting
Setting A BREAK point

If you have a query that selects some repeating data, or you want to section out different rows, you can set up a break point to do this. For example, if you have a query which simply selects the deptno and ename columns from emp, you may want to section out rows for each department, and also only print the department each time it changes.

<u>Example Before</u>

```
DEPTNO ENAME
--------- --------
       10 CLARK
       10 KING
       10 MILLER
       20 SMITH
       20 ADAMS
```

If you first entered:

```
     SQL> BREAK ON deptno SKIP 1
```

You would have:

```
DEPTNO ENAME
--------- --------
       10 CLARK
          KING
          MILLER

       20 SMITH
          ADAMS
```

The BREAK command says: set up a BREAK point on deptno and skip a line each time it changes.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Basic SQL* Plus Reporting
Saving Output to a File

You can save the output from SQL*Plus using the `spool` command: you simply enter

```
SQL> spool <file>
```

before you execute your SQL statement - <file> can be any valid filename, and if you omit the file extension then it will default to `.lst`. Once you have entered the `spool` command, ALL SQL*Plus output is saved to the file - to switch off spooling without leaving SQL*Plus, just enter:

```
SQL> spool off
```

Copyright © 1999 – 2004 ApplTop Solutions Limited

Summary

SQL*Plus is a very powerful tool, and it can do much more than described on this course: but, you should now have an idea as to what can be achieved using SQL*Plus. We briefly covered:

- SQL*Plus variables
- Basic SQL*Plus Reporting
- Saving Output to a File

Copyright © 1999 – 2004 ApplTop Solutions Limited

What is Next?

This is the end of the course; we have covered an awful lot of topics in such as short space of time. Don't be concerned if you haven't quite taken it all in, as this course was really designed to give you a taster of SQL and SQL*Plus. You should be equipped with enough knowledge to make it easy to further progress your knowledge in these areas.

I suggest you get to work with SQL and SQL*Plus as soon as you can, read any documentation you can on the subject, and you will be a fully fledged SQL and SQL*Plus scripter/programmer in no time.

This course has also given you good grounding for one of the follow-up courses:

PL/SQL, Packages, Procedures and Triggers

This course covers Oracle's procedural extension to SQL in the form of PL/SQL, a fully functional programming language.

Good Luck!!

Lab 6

1. Create a query that lists all employees. Prompt for a job type and only list employees in that job.
2. Ask for a column list, (columns separated by commas) then list all employees, showing only those columns you entered when asked.
3. Create a simple SQL*Plus report, it should have a title of 'Employees By Job' and list each job type with all employees for that job, skip 1 line between jobs. Show job type, employee number, employee name and hiredate.

Copyright © 1999 – 2004 ApplTop Solutions Limited

Answers To Exercises

Answers To Exercises

Lab	Exercise	Answer
1	1	Double Click the SQL*Plus icon on the desktop. Login to the database using the username and password supplied
	2	From the UNIX prompt enter, sqlplus <username>/<password>

Copyright © 1999 – 2004 ApplTop Solutions Limited

Answers To Exercises

Lab	Exercise	Answer
2	1	`SELECT *` `FROM   salgrade;`
	2	`SELECT *` `FROM   emp;`

	3	`SELECT *`
		`FROM   emp`
		`WHERE  sal BETWEEN 1600 AND 3000;`

	4	`SELECT   deptno`
		`,        dname`
		`FROM     dept`
		`ORDER BY dname;`

	5	`SELECT   DISTINCT job`
		`FROM     emp`
		`ORDER BY job DESC;`

	6	`SELECT ename`
		`,      hiredate`
		`FROM   emp`
		`WHERE  job = 'CLERK'`
		`AND    deptno = 20;`

	7	`SELECT *`
		`FROM   emp`
		`WHERE  ename LIKE 'S%';`

	8	`SELECT   ename`
		`,        job`
		`,        mgr`
		`,        sal`
		`FROM     emp`
		`WHERE    mgr IS NOT NULL`
		`ORDER BY sal DESC;`

	9	`SELECT ename`
		`,      sal + NVL(comm,0) "Remuneration"`
		`FROM   emp;`

	10	`SELECT ename`
		`,      sal`
		`,      sal*12 "Annual Salary"`
		`,      comm`
		`FROM   emp`
		`WHERE  sal < NVL(comm,0)`
		`ORDER BY sal DESC`

Copyright © 1999 – 2004 ApplTop Solutions Limited

Answers To Exercises

<table>
<tr><th>Lab</th><th>Exercise</th><th>Answer</th></tr>
<tr><td>3</td><td>1</td><td>SELECT ename
, sal
, ROUND(sal*1.15) new_sal
FROM emp;</td></tr>
<tr><td></td><td>2</td><td>SELECT ename
, LOWER(ename)
, INITCAP(ename)
FROM emp;</td></tr>
<tr><td></td><td>3</td><td>SELECT SUBSTR(ename,1,2)
|| TO_CHAR(empno)
|| substr(ename,3,LENGTH(ename)-2)
FROM emp;</td></tr>
<tr><td></td><td>4</td><td>SELECT RPAD(INITCAP(ename),7,' ')
|| ' is '||TO_CHAR(LENGTH(ename))
|| ' digits long'
FROM emp</td></tr>
<tr><td></td><td>5</td><td>SELECT
ename,hiredate,FLOOR(MONTHS_BETWEEN(SYSDATE,hiredate
)) months
FROM emp
ORDER BY months DESC;</td></tr>
<tr><td></td><td>6</td><td>SELECT ename
, DECODE(SIGN(1500-sal)
, 1,'BELOW 1500'
, 0,'On Target'
, sal) salary
FROM emp;</td></tr>
<tr><td></td><td>7</td><td>SELECT TO_CHAR(SYSDATE,'DAY') Day
FROM dual;</td></tr>
<tr><td></td><td>8</td><td>SELECT MAX(sal) max
, MIN(sal) min
, AVG(sal) avg
FROM emp;</td></tr>
<tr><td></td><td>9</td><td>SELECT job
, SUM(sal) total
FROM emp
GROUP by job;</td></tr>
<tr><td></td><td>10</td><td>SELECT COUNT(*)
FROM emp
WHERE job = 'CLERK';</td></tr>
<tr><td></td><td>11</td><td>SELECT deptno,COUNT(*) employees
FROM emp
GROUP BY deptno
HAVING COUNT(*) > 3;</td></tr>
</table>

Copyright © 1999 – 2004 ApplTop Solutions Limited

Answers To Exercises

Lab	Exercise	Answer

4	1	`SELECT e.ename`
		`,      d.dname`
		`FROM   emp  e`
		`,      dept d`
		`WHERE  d.deptno = e.deptno;`

	2	`SELECT e.ename`
		`,      e.deptno`
		`,      d.dname`
		`,      d.loc`
		`FROM   emp  e`
		`,      dept d`
		`WHERE  d.deptno = e.deptno;`

	3	`SELECT e.ename`
		`,      e.sal`
		`,      g.grade`
		`FROM   emp      e`
		`,      salgrade g`
		`WHERE  e.sal BETWEEN g.losal AND g.hisal;`

	4	`SELECT d.deptno`
		`FROM   dept d`
		`WHERE NOT EXISTS`
		`         (SELECT deptno`
		`          FROM   emp e`
		`          WHERE  e.deptno = d.deptno);`

	5	`SELECT e.ename`
		`,      m.ename`
		`FROM   emp e`
		`,      emp m`
		`WHERE  m.empno = e.mgr;`

	6	`SELECT e.ename`
		`,      NVL(m.ename,'NO MANAGER') manager`
		`FROM   emp e`
		`,      emp m`
		`WHERE  m.empno(+) = e.mgr;`

	7	`SELECT job,sal`
		`FROM   emp`
		`WHERE  (job,sal) IN`
		`             (SELECT   e.job,MAX(e.sal)`
		`              FROM     emp e`
		`              GROUP BY job)`

	8	`SELECT '*'||ename name`
		`,      hiredate`
		`FROM   emp`
		`WHERE  hiredate = (SELECT MAX(hiredate)`
		`                   FROM emp)`
		`UNION`
		`SELECT ename name`
		`,      hiredate`
		`FROM   emp`
		`WHERE  hiredate <> (SELECT MAX(hiredate)`
		`                    FROM emp);`

Copyright © 1999 – 2004 ApplTop Solutions Limited

Answers To Exercises

<table>
<tr><th>Lab</th><th>Exercise</th><th>Answer</th></tr>
<tr><td>5</td><td>1</td><td><pre>UPDATE emp
SET   sal = sal * 1.25;
COMMIT;
SELECT ename
,   sal
FROM   emp;</pre></td></tr>
<tr><td></td><td>2</td><td><pre>UPDATE emp
SET   sal = sal * 1.05
WHERE deptno IN (SELECT deptno
                 FROM dept
                 WHERE loc = 'NEW YORK');
ROLLBACK;
SELECT ename
,   sal
FROM   emp;</pre></td></tr>
<tr><td></td><td>3</td><td><pre>INSERT INTO dept VALUES (50,'IT','ENGLAND');</pre></td></tr>
<tr><td></td><td>4</td><td><pre>ROLLBACK</pre>or if you committed your work,..<pre>DELETE dept
WHERE deptno = 50;</pre></td></tr>
<tr><td></td><td>5</td><td><pre>CREATE SEQUENCE empinfo_seq
    START WITH 10
    INCREMENT BY 1;</pre></td></tr>
<tr><td></td><td>6</td><td><pre>CREATE TABLE employee_info
(info_id   NUMBER
,   empno   NUMBER
,   info_date DATE
,   info   VARCHAR2(80));</pre></td></tr>
<tr><td></td><td>7</td><td><pre>INSERT INTO employee_info VALUES
(empinfo_seq.NEXTVAL,7900,SYSDATE,'Some Info');</pre></td></tr>
<tr><td></td><td>8</td><td><pre>CREATE SYNONYM einfo
    FOR employee_info;</pre></td></tr>
<tr><td></td><td>9</td><td><pre>SELECT *
FROM   einfo;</pre></td></tr>
<tr><td></td><td>10</td><td><pre>DROP TABLE employee_info;</pre></td></tr>
</table>

Copyright © 1999 – 2004 ApplTop Solutions Limited

Answers To Exercises

Lab	Exercise	Answer

6	1	`SELECT *`
		`FROM   emp`
		`WHERE  job = '&1';`

	2	`SELECT &1` `FROM   emp;`

	3	`TTITLE 'Employees By Job'`
		`BREAK ON job SKIP 1`
		`SELECT job`
		`,      empno`
		`,      ename`
		`,      hiredate`
		`FROM   emp;`

Copyright © 1999 – 2004 ApplTop Solutions Limited

Demo Tables

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	New York
20	RESEARCH	Dallas
30	SALES	Chicago
40	OPERATIONS	Boston

EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2572.5		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5250		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1365		10

SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

Copyright © 1999 – 2004 ApplTop Solutions Limited

www.appltop.com
info@appltop.com

NOTES

Copyright © 1999 – 2004 ApplTop Solutions Limited

NOTES

Copyright © 1999 – 2004 ApplTop Solutions Limited