Lab 1 - Table Definition, Manipulation and Simple Queries

1. INTRODUCTION

Oracle is a relational database management system (DBMS) that stores, manages, and manipulates data. It consists of the database manager, the Database Administrator utility and other user utilities. The query language which will be used is called SQL (an acronym for Structured Query Language), and is implemented as part of a package called SQL*Plus distributed by ORACLE corporation.

2. USING SQL*PLUS

2.1 Logging into the gateway server

Launch the Terminal. Enter *SSH yourEID@gateway.cs.cityu.edu.hk* to connect to the gateway server with your EID. Answer Yes to the Host Identification question. Then enter your password for EID as your login password. You can then see the server shell command prompt.

2.2 Starting SQL*Plus

To start SQL*Plus, first login gateway server account and then type the following command:

sqlplus

From now on the username will be your EID and password for the sql system will be your student ID (8 digit number).

SQL*Plus will display the version of the Oracle Server and JServer already built in:

SQL*Plus: Release 9.0.1.0.0 - Production on Wed Jan 26 14:22:52 2011

(c) Copyright 2001 Oracle Corporation. All rights reserved.

Enter password:

Connected to:

Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production With the Partitioning, OLAP, Data Mining and Real Application Testing options

Then the SQL* Plus command prompt appears:

SQL>

This indicates that SQL*Plus is ready to accept your commands.

2.3 Exiting from SQL*Plus

When you have finished working with SQL*Plus and want to exit from the SQL*Plus environment, enter the following command at the SQL*Plus command prompt:

SQL> exit OR ***SQL> quit***

You will be disconnected from Oracle and return to the UNIX prompt.

Always remember to type *exit* or *quit* to terminate the SQL*Plus session normally. Otherwise your modification to the database may not take effect and your data may be lost.

2.4 Getting Help

You may get information on the commands and conventions of SQL*Plus, SQL, and PL/SQL using **help** command:

SQL > help

The following command displays a list of SQL*Plus, SQL and PL/SQL commands:

SQL > help index

Get help on the SQL*Plus clause LIST:

SQL > help list

3. LAB OBJECTIVES

In this lab section, you will learn to:

- Define tables
- Create your own tables
- Drop your own tables
- Manipulate tables
- Insert tuples into your tables
- Altering your own tables
- Query information from tables
- Select all the columns from a table
- Select specific columns from a table
- Creating a new table from existing data
- Updating data in tables
- Deleting tuples rows in tables

4. BASIC CONCEPT ON RELATIONAL DATABASE

A database is an organized collection of information. In a relational database, the information is organized in the form of tables. In this lab, we will cover two of the most important components of a database: tables and queries. Each student needs to create the following tables and insert records into the corresponding tables. Information stored in these tables is needed for all the laboratory exercise in this course.

The following set of tables contains the personal records for a fictitious company.

EMP contains information about the employees of the company
DEPT contains information about the departments of the company
SALGRADE group salary ranges into grades

DEPT

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

SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400

3	1401	2000
4	2001	3000
5	3001	9999

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
7655	JONES	MANAGER	7839	2-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	1-MAY-91	2850		30
7782	CLARK	MANAGER	7839	9-JUN-81	2450		10
7788	SCOTT	ANALYST	7655	21-MAR-87	3000		20
7839	KING	PRESIDENT		12-NOV-81	5000	0	10
7844	TURNER	SALESMAN	7698	18-SEP-81	1500		30
7876	ADAMS	CLERK	7788	24-APR-87	1100		20
7900	JAMES	CLERK	7698	3-DEC-81	950		30
7902	FORD	ANALYST	7655	3-DEC-81	3000		20
7934	MILLER	CLERK	7782	3-JAN-81	1300		10

5. SIMPLE DEFINITION OF TABLES

5.1 Create Tables

To create a user-defined table, a CREATE TABLE command is available which defines the table's name, its columns/attributes and other properties.

5.1.1 Common Data Types

CHAR (n)
n = maximum length of char string

NUMBER (n,d)
n = maximum number of digits
d = maximum number of digits right of decimal.

DATE
The relevant data plus the actual time of day

LONG
Up to 65,536 characters (64K) may be stored per field

RAW
Raw binary data

5.1.2 Forbidding NULL Values

To forbid NULL values in a column, enter the NOT NULL clause at the end of the column information. For example, the command:

```
SQL > CREATE TABLE DEPT
  2 (DEPTNO NUMBER (2) NOT NULL,
  3 DNAME VARCHAR (15),
  4 LOC VARCHAR (15));
```

creates the table definition of DEPT into your local database.
The SALGRADE and EMP tables are created as follows:

```
SQL > CREATE TABLE SALGRADE
  2 (GRADE NUMBER,
  3 LOSAL NUMBER,
  4 HISAL NUMBER);

SQL > CREATE TABLE EMP
  2 (EMPNO NUMBER (4) NOT NULL,
  3 ENAME VARCHAR (15),
  4 JOB VARCHAR (15),
  5 MGR NUMBER (4),
  6 HIREDATE DATE,
  7 SAL NUMBER (7,2),
  8 COMM NUMBER (7,2),
  9 DEPTNO NUMBER (2));
```

5.2 The DESCRIBE Command

You can use the DESCRIBE command to obtain a description of a specific table. The description will contain

Name -	name of column
Null? -	whether null values are allowed in this column
Type -	data type of column

```
SQL > DESCRIBE DEPT
```

5.3 Drop Tables

To destroy the table and its contents, use the DROP TABLE command:

```
SQL > DROP TABLE DEPT;
```

For the time being, ***don't*** use this command as you'll need the tables created for future uses.

6. SIMPLE MANIPULATIONS OF TABLES

6.1 Inserting Tuples into Tables

After the table is created, you can use the INSERT…VALUES command to insert tuples/rows into your table. For example, the command:

```
SQL > INSERT INTO DEPT
  2 VALUES (10, 'ACCOUNTING', 'NEW YORK');
```

inserts the first tuple into DEPT table.

User variables & (or called substitution variable) can be used to help reduce the tedium of data insertion. For example, to insert rows into the DEPT table, enter the following command:

```
SQL > INSERT INTO DEPT
  2 VALUES (&DEPTNO, '&DNAME', '&LOC');
```

and just repeatedly invoke this command by typing / or the **RUN** command, and enter new values accordingly. For those entries that are "empty", e.g. some COMM fields, use NULL as the input.

After each insert, update and delete operations that would modify the data in the database, you are suggested to do a "COMMIT" operation to ensure the changes take effect immediately:

SQL> COMMIT;

Commit complete.

6.2 Altering Tables

You may add a new column or change the length of an existing column by using the ALTER TABLE command with either the ADD or MODIFY parameters. Note that you may not reduce the width of a column if it contains data, nor may you change data type unless the column is empty.

- Change the length of column LOC in table DEPT from 15 to 20

SQL > ALTER TABLE DEPT
2 MODIFY (LOC VARCHAR(20));

7. SIMPLE QUERIES ON THE TABLES

To retrieve information out of the database, SELECT command is available for various kinds of queries against the database. The following starts with some simple usages of the SELECT command. More sophisticated ones will be illustrated and conducted in subsequent lab classes. We now begin the tutorial on this command with the SELECT clause.

7.1 The SELECT Clause

A SELECT command must have at least two clauses:

SELECT the columns to retrieve
FROM the tables which contain those columns

You may place the clauses on the same line or on separate lines. Always follow the last clause in the command with a semicolon (;). For example, the command:

SQL > SELECT DNAME FROM DEPT;

Displays the information in the department name column (DNAME) from the table DEPT. It could also be entered as:

SQL > SELECT DNAME
2 FROM DEPT;

or even:

SQL > SELECT DNAME
2 FROM DEPT
3 ;

For the time being, we will only SELECT columns from one table at a time. Combining tables will be considered in later labs.

7.1.1 Selecting All the Columns from a Table

(1) Select every column of the DEPT list the names of the columns:

SQL > SELECT DEPTNO, DNAME, LOC
2 FROM DEPT;

If all the columns of a table are to be selected you may use an asterisk (*) in place of the list of column names.

(2) Select all the columns from a table

SQL > SELECT * FROM DEPT;

7.1.2 Selecting Specific Columns from a Table

If only some of the columns are to be selected then list only the names of the columns that you want to see.

- Select just the DNAME and DEPTNO columns from the DEPT table.

```
SQL > SELECT DNAME, DEPTNO FROM DEPT;
```

The order in which the columns are listed in the SELECT clause controls the left-to-right sequence in which the columns are displayed. When an asterisk is used, the left-to-right sequence in which the columns are displayed is the order in which they were defined when the table was created.

- Select all columns from the EMP table.

```
SQL > SELECT * FROM EMP;
```

Look at the results of the queries in this section. Notice that the result of any query is itself a table called the *result table* – made up of columns and rows.

- List all the jobs in the EMP table without eliminating duplicates.

```
SQL > SELECT JOB FROM EMP;
```

You can eliminate duplicate rows in the result by specifying DISTINCT in the SELECT clause.

- List all the DISTINCT jobs in the EMP table.

```
SQL > SELECT DISTINCT JOB FROM EMP;
```

7.2 The WHERE Clause

In this subsection, you will learn about:

- Selecting specific rows from a table
- Selecting rows that satisfy multiple conditions
- Selecting rows that satisfy one of several conditions
- Selecting rows that DO NOT meet certain conditions
- Selecting rows using both ANDs and ORs – Boolean Precedence

7.2.1 Selecting Specific Rows from a Table

In the previous examples, all the rows of the table were selected. But suppose you want to retrieve only the employees in department 30. To do this, you will need to use a WHERE clause in your SELECT command. A WHERE clause, when used, always follows the FROM clause.

SELECT some columns
FROM some tables
WHERE certain conditions are met

Try the following example:

- Select only the employees in department 30.

```
SQL > SELECT *
  2 FROM EMP
  3 WHERE DEPTNO = 30;
```

A WHERE clause causes a "search" to be made, and only those rows that meet the search-conditions are retrieved. In the example above, only those rows WHERE the employee's department number is equal to 30 will be selected. That is, the WHERE clause was used to compare values stored in the DEPTNO column of the EMP table with the value 30 that you specified as a part of the query.

WHERE	DEPTNO	=	30
WHERE	Column-name	Comparison-operator	Constant-value

A WHERE clause can contain a numeric constant-value (30 in the example above) or a character constant-value shown as in the example below.

- List the names, numbers, and departments of all clerks

```
SQL > SELECT ENAME, EMPNO, DEPTNO
      2 FROM EMP
      3 WHERE JOB = 'CLERK';
```

If the constant value that you specify is character data then it has to be enclosed in single quotes ('CLERK'). The case also matters: try the previous query, but replace CLERK by clerk. Numeric data does not need to be enclosed in quotes.

A WHERE clause search condition can use any of the following comparison operators:

=	Equal to
!=	Not equal to
>	Greater than
> =	Greater than or equal to
<	Less than
<=	Less than or equal to

- Find the names and department numbers of all departments with a department number greater than 20.

```
SQL > SELECT DNAME,DEPTNO
      2 FROM DEPT
      3 WHERE DEPTNO > 20;
```

- Find the names and department numbers of all departments with a department number greater than or equal to 20.

```
SQL > SELECT DNAME, DEPTNO
      2 FROM DEPT
      3 WHERE DEPTNO >=20;
```

In addition to a comparison with a constant-value, a WHERE clause can compare a value stored in one column of a row with a value in another column of the same row.

WHERE	SAL	>	COMM
WHERE	Column-name	Comparison-operator	Column-name

- Find the employees whose commission is greater than his salary.

```
SQL > SELECT ENAME, SAL, COMM
      2 FROM EMP
      3 WHERE COMM > SAL;
```

7.2.2 Selecting Rows That Satisfy Multiple Conditions

At times, it is necessary to specify more than one search-condition in a WHERE clause to retrieve the desired rows from a table.

- List the name, job, title and salary of all employees in department 20 that make more than $2,000.

```
SQL > SELECT ENAME, JOB, SAL
```

```
2   FROM EMP
3   WHERE DEPTNO = 20
4   AND SAL > 2000;
```

Here, the two search conditions are connected by the word AND. The AND means that both search-conditions must be true for a given row to be retrieved. Any number of search-conditions may be ANDed to a WHERE clause.

- Find all the salesmen in department 30 who have a salary greater then or equal to $1,500.

```
SQL > SELECT ENAME, SAL, DEPTNO
2 FROM EMP
3 WHERE JOB = 'SALESMAN'
4 AND DEPTNO = 30
5 AND SAL >= 1500;
```

7.2.3 Selecting Rows That Satisfy One of Several Conditions

In addition to selecting rows that meet several conditions, you are also to select rows that meet one of several conditions.

- Find all the employees whose job is either manager or president.

```
SQL > SELECT ENAME, JOB, SAL
2 FROM EMP
3 WHERE JOB = 'MANAGER'
4 OR JOB = 'PRESIDENT';
```

Here the search-conditions are connected by the word OR. OR means that if either search-condition is true then the row is to be selected.

7.2.4 Selecting Rows that DO NOT Meet Certain Conditions

You can select rows that do not meet certain conditions as well as rows that do meet certain conditions.

- Find all managers who are NOT in department 30.

```
SQL > SELECT ENAME, SAL, COMM, JOB, DEPTNO
2 FROM EMP
3 WHERE JOB = 'MANAGER'
4 AND DEPTNO != 30;
```

7.2.5 Selecting Rows Using Both AND's and OR's

AND and OR may be combined in the same query to specify as many search conditions as are necessary to select the desired rows.

- Find everyone whose job title is manager, and all the clerks in department 10.

```
SQL > SELECT *
2 FROM EMP
3 WHERE JOB = 'MANAGER'
4 OR JOB = 'CLERK' AND DEPTNO = 10;
```

The WHERE clause in the above query can be written suing parentheses without changing the meaning of query.

- Find all the manager (in any department), and all the clerks in department 10.

```
SQL > SELECT *
2 FROM EMP
3 WHERE JOB = 'MANAGER'
```

4 OR (JOB = 'CLERK' AND DEPTNO = 10);

Parentheses are sued to group search-conditions together to control the sequence in which the search-conditions are applied. This is called logical precedence. Depending on position of the parentheses, the meaning of the WHERE clause may be changed

- Issue the same query again with the parentheses grouping the first two search conditions inserted of the last two.

```
SQL > SELECT *
      2 FROM EMP
      3 WHERE (JOB = 'MANAGER' OR JOB = 'CLERK')
     4 AND DEPTNO = 10;
```

Notice how in the last query both managers and clerks have to be in department 10, not just the clerks. When a WHERE clause contains more than two search conditions you should use parentheses to "classify the meaning" of the WHERE clause.

Any group of search-conditions can be ***negated*** by enclosing them in parentheses and preceding them with a NOT.

- Find everyone who is neither a manger nor a clerk, but is in department 10.

```
SQL > SELECT *
      2 FROM EMP
      3 WHERE NOT (JOB = 'MANAGER' OR JOB  = 'CLERK')
      4 AND DEPTNO = 10;
```

8. DATA MANIPULATION

8.1 Creating a New Table From Existing Data

You can create a new table containing data from existing tables.

- Create a new table MY_EMP from existing table EMP.

```
SQL > CREATE TABLE MY_EMP AS
       2  SELECT * FROM EMP;
```

8.2 Updating Data in Tables

The update statement changes all or part of an existing row in a single table.

- Update the salary of the President in the MY_EMP table.

```
SQL > UPDATE MY_EMP
       2 SET SAL = SAL *1.1
       3 WHERE JOB = 'PRESIDENT';
```

8.3 Deleting Tuples in Tables

The delete statement is to remove rows from a single table.

- Delete the tuples in MY_EMP where the employee's JOB is CLERK.

```
SQL > DELETE FROM MY_EMP
      2 WHERE JOB = 'CLERK';
```

Lab 2 - Queries, Arithmetic Expressions and Functions

1. Objective

This set of laboratory exercises illustrates some more advanced features of SQL and SQL*Plus expressions. The number of examples to try is large, yet you are advised to complete all of them, both in and outside your timetabled lab classes.

As with the previous handout, the main source of most of the exercises in this handout is the SQL*Plus User's Manual published by the ORACLE Corporation.

In order to test all the suggested examples, you need to have the three tables existing on your Oracle database account, namely EMP, DEPT and SALGRADE. We now begin the tour of showing more advanced features of SQL*Plus commands first.

2. SQL*PLUS LINE EDITING COMMANDS

When you type a SQL command, Oracle stores it in an area called the *SQL command buffer*. The SQL command stays in the buffer until it is replaced by the next set of SQL commands that you enter. The command in the buffer is called the "current" SQL command. SQL*Plus provides some line editing commands to help you modify the buffer:

SQL*Plus Commands

COMMAND	ABBEVIATION	PURPOSE
APPEND	A text	Add text at the end of a line
CHANGE/old/new	C/old/new	Change old to new in the line
CHANGE/text	C text	Delete text from a line
CLEAR BUFFER	CL BUFF	Delete all lines
DEL	(none)	Delete a line
INPUT	I	Add one or more lines
LIST	L	List all lines in the SQL buffer
LIST n	L n OR n	List one line
LIST LAST	L LAST	List the last line
LIST m n	L m n	List a range of lines (m to n)

2.1 Change the Current SQL Command Without Retyping It

Oracle allows you to modify the current SQL command without having to retype the entire command.

- Enter a command with the column named DEPTNO mistakenly entered as DPTNO.

```
SQL > SELECT DPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE DEPTNO = 10;
```

Oracle responds to the typing mistake with an error message. Rather than retyping the entire command you can correct the mistake by modifying the command in the buffer. Firstly you must "position" to the line of the command that contains the error by using the LIST command (abbreviated L).

- Position to line 1 (L1) of the current query

```
SQL > L1
```

Once you have positioned to the line that contains the error you can fix the mistake using the CHANGE command.

- Change DPTNO to DEPTNO

SQL > CHANGE/DPTNO/DEPTNO

The CHANGE command changes the contents of a line in the SQL command buffer and displays the new contents of the line so that you can verify the change. (Note that the CHANGE command can be abbreviated as C).

2.2 Reissuing the Current SQL Command

You may now issue the RUN command to execute the correct SQL command in the SQL command buffer.

- Execute the current SQL command

SQL > RUN

Notice that as a result of the RUN command Oracle will display the contents of the SQL command buffer and then execute the command.

2.3 Listing the Current SQL Command

If you just want to look at but not execute the current SQL command you use the LIST command

- List the contents of the SQL command buffer.

SQL > LIST

Notice the asterisk after the number 3 (3* WHERE DEPTNO = 10). The asterisk indicates that you are "positioned" to line 3 in the SQL command buffer. If you issue a CHANGE command it will be applied to line number 3.

2.4 Adding One or More Lines

In addition to being able to change a line in the buffer, you can use the INPUT command (abbreviated as "I") to add a new line after the line you are positioning to.

- Add a new line to the current SQL query.

```
SQL > I
      4 ORDER BY SAL DESC
      5
SQL > R
```

Notice that when you type I you get line number prompt (4 in the example above). You can then enter one or more new lines into the SQL command buffer. To get back to the SQL*Plus prompt you must enter a blank line. After you get back to the SQL*Plus prompt you can issue the Run command (abbreviate as "R").

2.5 The Difference Between SQL Commands and SQL*Plus Commands

CHANGE, RUN and LIST are SQL*Plus commands – ***not*** SQL Commands. SQL commands like SELECT are used to access data in tables while SQL*Plus commands like CHANGE are used to modify or "edit" the contents of the SQL command buffer. There are additional SQL*Plus command or controlling the appearance of query output that we will discuss later.

3. MORE ON THE SELECT CLAUSE

3.1 Selecting Rows Within A Certain Range

The BETWEEN operator lets you select rows that contain values within a range

- Find all the employees who earn between $1,200 to $1,400.

SQL > SELECT ENAME, JOB, SAL
2 FROM EMP
3 WHERE SAL BETWEEN 1200 AND 1400;

- An equivalent WHERE clause have been specified as: where SAL >= 1200 and SAL <=1400. The BETWEEN operator can be modified with the work NOT.

SQL > SELECT ENAME, JOB, SAL
2 FROM EMP
3 WHERE SAL NOT BETWEEN 1200 AND 1400.

NOT BETWEEN means that only rows that are outside the range will be selected.

3.2 Selecting Rows That Match A Value In A List

The IN operator lets you select rows that contain a value that match one of the values in a list of values that you supply.

- Find the employees who are clerks, analysts or salesmen.

SQL > SELECT ENAME, JOB, DEPTNO
2 FROM EMP
3 WHERE JOB IN ('CLERK', 'ANALYST', 'SALESMAN');

The IN clause can be modified with NOT to select only the rows that have a value that is NOT in the list.

- Find the employees who are NOT clerks, analysts or salesmen.

SQL > SELECT ENAME, JOB, DEPTNO
2 FROM EMP
3. WHERE JOB NOT IN ('CLERK', 'ANALYST', 'SALESMAN');

3.3 Selecting Rows That Match A Character Pattern

In addition to being able to select rows that *completely* match a value that you supply (e.g., ENAME = 'ALLEN') you can select rows that *partially* match a combination of characters that you supply.

- Find all the employees whose names begin with the letter M.

SQL > SELECT ENAME, JOB, DEPTNO
2 FROM EMP
3 WHERE ENAME LIKE 'M%';

- Find all the employees whose names end with the letter N.

SQL > SELECT ENAME, JOB, DEPTNO
2 FROM EMP
3 WHERE ENAME LIKE '%N';

The LIKE operator uses two special characters:

%	The percent character represents any string of zero or more characters
_	The underscore character means a single character position

In the example above, the letter N is to be selected regardless of how many characters precede the N. In the example below, the row will be selected only if the letter N is preceded by exactly four letters.

- Find all the employees whose name are 5 characters long and end with the letter N.

```
SQL > SELECT ENAME, JOB, DEPTNO
  2  FROM EMP
  3  WHERE ENAME LIKE '_ _ _ _ N';
```

The % and the _ can be used in combination to select the desired "pattern" of characters.

- Find all the employees whose names are not 5 characters long.

```
SQL > SELECT ENAME, JOB, DEPTNO
      2 FROM EMP
      3 WHERE ENAME NOT LIKE '_ _ _ _ _';
```

4. ORDERING ROWS OF A QUERY RESULT

In all the examples so far, the rows of a query result have been displayed in an order determined by Oracle. You can control the order in which the selected rows are to be displayed by adding an ORDER BY clause to your SELECT command.

4.1 Ordering Rows In Ascending Order

An ORDER BY clause is always the last clause in a SELECT command.

- List the employees in department 30 in order by their salary.

```
SQL > SELECT SAL, JOB, ENAME
      2 FROM EMP
      3 WHERE DEPTNO = 30
      4 ORDER BY SAL;
```

The ORDER BY clause caused a "sort" of the rows into ascending (smallest salary first) order.

4.2 Ordering Rows In Descending Order

You may also order rows into descending (largest salary first) order.

- List all the employees in DESCending order by salary.

```
SQL > SELECT SAL, JOB, ENAME,
      2 FROM EMP
      3 WHERE DEPTNO = 30
      4 ORDER BY SAL DESC;
```

Descending order is indicated by placing a DESC after the column name that you are ordering by. But ordering is not limited to sorting on a single column.

4.3 Ordering Rows Using Multiple Columns.

- Order all employees by job, and within job put them in descending salary order.

```
SQL > SELECT JOB, SAL, ENAME
      2 FROM EMP
      3 ORDER BY JOB, SAL DESC;
```

CLERKs are in descending salary order, and all the MANAGERs are in descending salary order, etc.

4.4 Ordering Rows By Columns With NULL Values

When you order by columns with null values, Oracle 9i will consider the null entries as the largest number and put them into the end of the list if ascending. Or you can tell the DBMS the way you want to sort the NULL values. The following code segment indicates this:

SELECT ... ORDER BY ... ASC/DESC NULLS FIRST/LAST;

- Order the employees in department 30 in ascending order of their commission.

SQL > SELECT COMM, SAL, ENAME
2 FROM EMP
3 WHERE DEPTNO = 30
4 ORDER BY COMM;

- Order the employees in department 30 in descending order of the commission.

SQL > SELECT COMM, SAL, ENAME
2 FROM EMP
3 WHERE DEPTNO = 30
4 ORDER BY COMM DESC;

5. EXPRESSION AND FUNCTIONS

5.1 Arithmetic Expressions

A SQL command can contain arithmetic expressions. An arithmetic expression is made up of column names and constant numeric values connected by arithmetic operators. Any one of the following arithmetic operators may be used:

Symbol	Operation	Symbol	Operation
+	Addition	-	Subtraction
*	Multiplication	/	Division

An arithmetic expression can be used in a SELECT clause to retrieve calculated results.

- List the name, salary, commission, and sum of salary plus commission of all salesmen.

SQL > SELECT ENAME, SAL, COMM, SAL+COMM
2 FROM EMP
3 WHERE JOB = 'SALESMAN';

An arithmetic expression can be used in a WHERE clause to select rows based upon a calculated search-condition.

- List the name, salary and commission of employees whose commission is greater than 25% of their salary.

SQL > SELECT ENAME, SAL, COMM
2 FROM EMP
3 WHERE COMM > 0.25 * SAL;

Arithmetic-expression can also be used in an ORDER BY clause.

- List all salesmen in descending order of their commission divided by their salary.

SQL > SELECT ENAME, COMM/SAL, COMM, SAL
2 FROM EMP
3 WHERE JOB = 'SALESMAN'
4 ORDER BY COMM/SAL DESC;

More than one arithmetic operator can be used in an arithmetic expression. Parentheses are used to control the order of the operations to be performed.

- Calculate the total annual compensation of all salesmen based upon their monthly salary and their monthly commission.

```
SQL > SELECT ENAME, SAL, COMM, (SAL+COMM)*12
      2 FROM EMP
      3 WHERE JOB = 'SALESMAN';
```

5.2 The NULL Function

When an expression or function references a column that contains a null value, the result is null.

- Select SAL+COMM where some of the employees have a null commission.

```
SQL > SELECT ENAME, JOB, SAL, COMM, SAL+COMM
      2 FROM EMP
      3 WHERE DEPTNO = 30;
```

The expression SAL+COMM returns a null value for all employees that have a null value in their COMM field. The Oracle NULL-values, NVL, can be used to assign a temporary value to nulls encountered within an expression.

- Treat null commissions as zero commissions within the expression SAL+COMM.

```
SQL > SELECT ENAME, JOB, SAL, COMM, SAL+NVL(COMM, 0)
      2 FROM EMP
      3 WHERE DEPTNO = 30;
```

The expression SAL+NVL(COMM, 0) will return a value equal to SAL + 0 when the COMM field is null.

5.3 Column Labels

Column names are normally used as display headings for query results. To make query results more readable you can use column labels to "rename" either column names or arithmetic expressions in the SELECT clause. The column label will then be used as the display heading in place of either the column name or expression.

- Issue the previous query with the column label SALARY replacing SAL, COMMISSION replacing COMM and TOTAL_COMPENSATION replacing the expression SAL+NVL(COMM, 0).

```
SQL > SELECT ENAME, JOB, SAL SALARY, COMM COMMISSION,
      2 SAL+NVL(COMM, 0) TOTAL_COMPENSATION
      3 FROM EMP
      4 WHERE DEPTNO = 30;
```

As shown, column labels follow column names or expressions separated by a blank. Notice the use of the underscore (_) between TOTAL and COMPENSATION in the example above. This is because column labels, table names, column names, etc. may not contain embedded blanks or special characters (+,-,*,/) without being enclosing in double quotes (").

- Issue the previous query with the column label containing an embedded blank.

```
SQL > SELECT ENAME, JOB, SAL SALARY, COMM COMMISSION,
      2  SAL+NVL(COMM, 0) "TOTAL COMPENSATION"
      3  FROM EMP
```

```
4 WHERE DEPTNO = 30;
```

5.4 Arithmetic Functions

In addition to arithmetic operators (+,-,*,/), you can have *arithmetic functions* in a SQL command. The arithmetic functions include:

ROUND(number[, d])	Rounds number to d digits right of the decimal point
TRUNC (number[, d])	Truncates number to d digits right of the decimal point
ABS (number)	Absolute value of the number
SIGN (number)	+1 if number > 0; 0 if number = 0; -1 if number < 0
TO_CHAR (number)	Converts a number to a character
MOD (num1, num2)	Num1 modulo num2

When an arithmetic function is used, the column name must be enclosed in parentheses.

- Calculate the daily salary of the department 30 employees. Select the results in three separate columns: unrounded, rounded to the nearest dollar, and rounded to the nearest cent. For the purposes of the calculation assume there are 22 working days in a month.

```
SQL > SELECT ENAME, SAL, SAL/22, ROUND(SAL/22, 0), ROUND(SAL/22, 2)
  2 FROM EMP
  3 WHERE DEPTNO = 30;
```

In the example above, the first expression (SAL/22) was not rounded at all. The second expression (round(SAL/22,0)) was rounded to the nearest dollar. The third expression (round(SAL/22,2)) was rounded to the nearest cent (tow digits to the right of the decimal point).

- Compute the daily and hourly salary for employees in department 30. Round the results to the nearest cent. Assume there are 22 working days in a month and 8 working hours in a day.

```
SQL > SELECT ENAME, SAL MONTHLY, ROUND(SAL/22,2) DAILY,
  2 ROUND(SAL/(22*8),2) HOURLY
  3 FROM EMP
  4 WHERE DEPTNO = 30;
```

- Issue the same query as the previous one except this time truncate (TRUNC) to the nearest cent rather than round.

```
SQL > SELECT ENAME, SAL MONTHLY, TRUNC(SAL/22,2) DAILY,
  2 TRUNC(SAL/(22*8),2) HOURLY
  3 FROM EMP
  4 WHERE DEPTNO = 30;
```

Notice the difference between the results of the previous query that used truncation, and the one before it that used rounding.

5.5 Character String Functions

In addition to functions that operate on numbers (arithmetic functions), Oracle provides you with functions that operate on character data. These functions are called *character string functions*. (Note that some character string functions allow you to

operate on a combination of character fields and numbers.) The character string functions include:

string1 || string2	Concatenates string1 with string2
LENGTH(string)	Length of a string
SUBSTR(str,spos[,len]	Substring of a string
INSTR(sstr,str[,spos])	Position of substring in the string
UPPER(string)	Changes all lower case characters to upper case
LOWER(string)	Changes all upper case characters to lower case
TO_NUM(string)	Converts a character to a number
LPAD(string,len[,chr])	Left pads the string with fill character
RPAD(string,len[,chr])	Right pads the string with fill character

The *concatenation function* allows you to combine several columns and/or constant values into a single column.

- Concatenate the DNAME column to the LOC column separated by a blank space, a dash and another blank space (‘ – ‘).

SQL > SELECT DNAME || ‘ – ‘ || LOC DEPARTMENTS
2 FROM DEPT;

5.6 Substring

- Abbreviate the department name to the first 5 characters of the department name.

SQL > SELECT SUBSTR(DNAME,1,5)
2 FROM DEPT;

- Select an employee name in both UPPER and LOWER case.

SQL > SELECT ENAME, UPPER(ENAME),LOWER(ENAME)
2 FROM EMP
3 WHERE UPPER(ENAME) = UPPER(‘Ward’);

Note the use of the UPPER function on the WHERE clause to insure that the case would match.

- Find the number of characters in department names.

SQL > SELECT DNAME, LENGTH(DNAME)
2 FROM DEPT;

Lab 3 - Aggregate Functions, Group By and Having Clauses

1. AGGREGATE FUNCTIONS

In all the examples so far, we have selected values stored in each row of a table (like SAL), or values calculated for each row (like SAL*12). That is, we have selected information about *individual* rows stored in the database. You can also select "summary" information about *groups* of rows in the database.

Oracle provides five aggregate functions (or group functions) that can be applied to data retrieved in a query:

AVG	Computes the average value
SUM	Computes the total value
MIN	Finds the minimum value
MAX	Finds the maximum value
COUNT	Counts the number of values

1.1 Selecting Summary Information From One Group

As shown below, the column that the function is applied to must be enclosed in parentheses.

- Find the average salary for clerks (group.1)

```
SQL > SELECT AVG(SAL)
    2   FROM EMP
    3   WHERE JOB = 'CLERK';
```

Oracle uses all the rows that satisfy the search-condition (WHERE JOB= 'CLERK') to compute the summary information. Rather than displaying a value (SAL) for each individual row (CLERK) selected, Oracle calculates a single value – AVG(SAL) – as a result.

You can have more than one group function in a SELECT clause.

- Find the total salary and total commission for salesmen (group 2)

```
SQL > SELECT SUM(SAL), SUM(COMM)
    2 FROM EMP
    3 WHERE JOB = 'SALESMAN';
```

You can use group functions in arithmetic expressions.

- Compute the average annual salary plus commission for all salesmen (group 3).

```
SQL > SELECT AVG(SAL + COMM)*12
    2   FROM EMP
    3   WHERE JOB = 'SALESMAN';
```

- Find the highest and lowest paid employee salaries and the difference between them (group 4).

```
SQL > SELECT MAX(SAL), MIN(SAL), MAX(SAL)-MIN(SAL)
    2 FROM EMP;
```

You can use group functions with arithmetic or string functions.

- Find the number of characters in the longest department name (group 5).

```
SQL > SELECT MAX(LENGTH(DNAME))
    2 FROM DEPT;
```

In standard SQL, you may NOT have both aggregates and non-aggregates in the same SELECT list. For example, SELECT ENAME, AVG(SAL) is an error. This is because ENAME is an attribute of a *each* row selected and AVG(SAL) is an attribute of *all* the rows selected. So if you want to find the name and salary of the employee (or employees) who receive the maximum salary, you cannot use the above query. Instead, you should use a sub-query as in the following example.

- Find the name and salary of the employee (or employees) who receive the maximum salary (group 6).

```
SQL > SELECT ENAME, JOB, SAL
      2 FROM EMP
      3 WHERE SAL =
      4 (SELECT MAX(SAL) FROM EMP);
```

The COUNT function can be used to count the number of values, number of distinct values, or number of rows selected by the query.

- Count the number of employees who receive a commission (group 7)

```
SQL > SELECT COUNT (COMM)
      2 FROM EMP;
```

COUNT may be used with the keyword DISTINCT to return the number of different values stored in the column. That is, duplicate values are eliminated before values are counted.

- Count the number of different jobs held by employees in department 30 (group.8)

```
SQL > SELECT COUNT(DISTINCT JOB)
      2 FROM EMP
      3 WHERE DEPTNO = 30;
```

- Count the number of employees in department 30 (group. 9)

```
SQL > SELECT COUNT (*)
      2 FROM EMP
      3 WHERE DEPTNO = 30;
```

2 SELECTING SUMMARY INFORMATION FROM GROUPS – GROUP BY

Let's say you want to know the average salary for the employees in department 10, department 20 and department 30. You could solve this problem with the following three queries:

- Find the average salary of the employees in each department.

```
SQL > SELECT AVG (SAL)
      2 FROM EMP
      3 WHERE DEPTNO = 10;
SQL > SELECT AVG (SAL)
      2 FROM EMP
      3 WHERE DEPTNO = 20;
SQL > SELECT AVG (SAL)
      2 FROM EMP
      3 WHERE DEPTNO = 30;
```

The same information could be returned by a single query with a GROUP BY clause. The GROUP BY clause divides a table into groups of rows with matching values in the same column (or columns).

- List the department number and average salary of each department (group.10)

```
SQL > SELECT DEPTNO, AVG(SAL)
  2 FROM EMP
  3 GROUP BY DEPTNO;
```

In the above query, all the employees are divided into groups based on their department number (GROUP BY DEPTNO). Thus, there are three different groups: the employees in department 10, the employees in department 20 and the employees in department 30. The group functions AVG(SAL) is then applied to all the rows in each group.

When you have a GROUP BY clause you can select the group column in addition to group functions. This is because the grouping column is an attribute of the group (all the rows of the group have the same value for the grouping column).

The GROUP BY clause always follows the WHERE clause, or the FROM clause when there is no WHERE clause in the command.

- Find each department's average annual salary for all its employees except the managers and the president (group. 11).

```
SQL > SELECT DEPTNO, AVG(SAL)*12
    2 FROM EMP
    3 WHERE JOB NOT IN ('MANAGER', 'PRESIDENT')
    4 GROUP BY DEPTNO;
```

You can partition rows of a table into groups based on the values in more than one column.

- Divide all employees into groups by department, and by jobs within department. Count the employees in each group and compute each group's average annual salary. (group. 12)

```
SQL > SELECT DEPTNO, JOB, COUNT(*), AVG(SAL)*12
    2 FROM EMP
    3 GROUP BY DEPTNO, JOB;
```

In the example above, a group is formed for every job within every department (GROUP BY DEPTNO, JOB). Look at the last row of the query result. This group of department 30 salesmen has a count of 4 and an average salary of $16,800.

You can have join-conditions and group functions in the same query.

- Issue the same query as above except list the department name rather than the department number (group. 13)

```
SQL > SELECT DNAME, JOB, COUNT(*), AVG(SAL)*12
    2 FROM EMP, DEPT
    3 WHERE DEPT.DEPTNO = EMP.DEPTNO
    4 GROUP BY DNAME, JOB;
```

3 SPECIFYING A SEARCH-CONDITION FOR GROUPS – HAVING

You can specify search-conditions for groups just as you can specify conditions for individual rows. As you know, search-conditions for individual rows are specified in the WHERE clause, for groups, you should use HAVING clause to do specification.

- List the average annual salary for all job groups having more than 2 employees in the group. (group.14)

SQL > SELECT JOB, COUNT(*), AVG(SAL)*12
2 FROM EMP
3 GROUP BY JOB
4 HAVING COUNT(*) > 2;

A HAVING clause, when used, always follows the GROUP BY clause. The HAVING clause compares some property of the group (COUNT(*) in the example above) with a constant value (2 in the example above). If a group satisfies the search-condition in the HAVING clause, it is included in the query result.
A query can contain both a WHERE and a HAVING clause. Firstly, the WHERE clause is applied to qualify rows; secondly, the groups are formed and the group-functions are applied; thirdly, the HAVING clause is applied to qualify groups.

- List all the departments that have at least two clerks. (group. 15)

SQL > SELECT DEPTNO
2 FROM EMP
3 WHERE JOB = 'CLERK'
4 GROUP BY DEPTNO
5 HAVING COUNT(*) >=2;

A HAVING clause can also compare one attribute of the group with another attribute of the group.

- Find all departments with an average commission greater than 25% of average salary. (group. 16)

SQL > SELECT DEPTNO, AVG(SAL), AVG(COMM), AVG(SAL)*0.25
2 FROM EMP
3 GROUP BY DEPTNO
4 HAVING AVG(COMM) > AVG(SAL)*0.25;

You can use a sub-query in a HAVING clause to compare an attribute of the group with a computed attribute of another group.

- List the job groups that have an average salary greater than the average salary of managers. (group.17)

SQL > SELECT JOB, AVG(SAL)
2 FROM EMP
3 GROUP BY JOB
4 HAVING AVG(SAL) >
5 (SELECT AVG(SAL) FROM EMP
6 WHERE JOB = 'MANAGER');

4 EFFECTS OF NULL VALUES ON GROUP FUNCTIONS

Null values do not participate in the computation of group functions.

- Count the number of people in department 30 who receive a salary and the number of people who receive a commission. (group 18)

SQL > SELECT COUNT (SAL), COUNT(COMM)
2 FROM EMP
3 WHERE DEPTNO = 30;

The count of people who receive a salary, 6, is greater than the count of people who receive a commission, 3. This is because null commissions were not counted.

- Null values do not participate in the computation of any group functions.(group 19)

SQL > SELECT SUM(SAL), COUNT(SAL), AVG(SAL), SUM(COMM),
2 COUNT (COMM), AVG(COMM)
3 FROM EMP
4 WHERE DEPTNO = 30;

- List the average commission of employees who receive a commission, and the average commission of all employees (treating employees who do not receive a commission as receiving a zero commission). (group. 20)

SQL > SELECT AVG(COMM),AVG(NVL(COMM, 0))
2 FROM EMP
3 WHERE DEPTNO = 30;

5 FORMATING QUERY RESULTS INTO A REPORT

In this section, you will learn some more SQL*Plus commands to improve the appearance (format) of a query result. The commands to be covered in this section are:

COLUMN	Format a column's heading and data
TTITLE	Put a title on the top of the page
BTITLE	Put a title on the bottom of the page
BREAK	Break the report up into groups of rows
COMPUTE	Perform various computation in each group

5.1 Column Formatting Commands

Through the SQL*Plus COLUMN command you can change the column headings and reformat the column data in your query results.

- List the department number, name and salary of all employees in department 20.

SQL > SELECT DEPTNO, ENAME, SAL
2 FROM EMP
3 WHERE DEPTNO = 20;

5.1.1 Changing Column Headings

We can make the output from the previous query more readable by changing the column headings. You do this by using the COLUMN command with a HEADING clause.

Enter the command:

SQL > COLUMN DEPTNO HEADING DEPARTMENT
SQL > RUN

A column heading may be more than one word long, but if it contains a blank character then it must be enclosed in single quotes (for example, 'EMPLOYEE NAME'). A column heading may be displayed on more than one line. This is done by placing a *vertical bar* (|) character between the words that you want to separate lines.

SQL > COLUMN ENAME HEADING 'EMPLOYEE | NAME'
SQL > RUN

5.1.2 Reformatting Column Data

The FORMAT clause in COLUMN command allows us to change the default format of numbers and text.

- Put a heading of MONTHLY SALARY on the SAL column and format the data with a dollar sign, a comma in the thousands position, and a decimal point between the dollars and cents position.

SQL > COLUMN SAL HEADING 'MONTHLY|SALARY' FORMAT $99,999.99
SQL > RUN

The following is a list of the different format masks that can be used to control the appearance of column data:

FORMAT	VALUE	DISPLAYED
999.99	56.478	56.48
999V99	56.478	5648
9,999	8410	8,410
99999	607	607
09999	607	00607
9999	-5609	-5609
9999MI	-5629	5609-
9999PR	-5609	<5609>
B999	564	564
99.99	124.98	###.##
$99.99	45.23	$45.23
A20	Customer	Customer
A5	Customer	Custo

5.1.3 Clearing Column Attributes

To reset the display attributes for all columns(or a specific column), use the CLEAR command (or CLEAR clause of the COLUMN command).

SQL > COLUMN column_name CLEAR

or

SQL > CLEAR COLUMNS

5.2 Getting Your Report To Print On Separate Pages

- Set the pagesize to 48 lines and cause a form feed to a newpage prior to printing every heading.

SQL > SET PAGESIZE 48
SQL > SET NEWPAGE 0
SQL > RUN

- Set the pagesize back to 24 and newpage back 1.

SQL > SET PAGESIZE 24
SQL > SET NEWPAGE 1
SQL > RUN

5. 3 Page Formatting Commands

In addition to using the COLUMN command to control the appearance of individual columns on the page, there are other SQL*Plus commands that allow you to control the appearance of the page itself.

5.3.1 TTITLE

- Put a title of ACME WIDGET – PERSONAL REPORT on separate lines at the top of each page of the report.

SQL > TTITLE 'ACME WIDGET || PERSONEL REPORT'
SQL > SELECT DEPTNO, ENAME, SAL
2 FROM EMP
3 ORDER BY DEPTNO;

Notice that the bar character can be used to separate TTITLE into separated lines just as in column headings. The tow bar characters (||) caused the title to contain a blank line between "ACME WIDGET" and "PERSONAL REPORT".

5.3.2 BTITLE

- Put a title of COMPANY CONFIDENTIAL at the bottom of the page.

SQL > BTITLE 'COMPANY CONFIDENTIAL'
SQL > RUN

5.3.3 BREAK

The rows of a report may be "broken-up' into groups by using the BREAK command.

BREAK ON column_name [SKIP n]
[PAGE]

- BREAK ON DEPTNO to separate the rows of the query result into groups. Skip 2 lines between each group.

SQL > BREAK ON DEPTNO SKIP 2
SQL > RUN

The BREAK command in the above example grouped together all the rows with the same value of DEPTNO, and only displayed it once. When a new value of DEPTNO was found, 2 lines were skipped. It is important to remember that the column that you BREAK ON should be the same as the column that you ORDER BY (otherwise the rows of your report will be broken into meaningless groups).

5.3.4 COMPUTE

After the rows of the query result have been divided into groups you can do computations for columns. Note that the COMPUTE command has no effect without a corresponding BREAK command.

- Compute the total salary of employees for each department.

SQL > COMPUTE SUM OF SAL ON DEPTNO
SQL > RUN

Note that subtotals were computed every time the value of DEPTNO changed.

- Compute the average salary of all employees at the end of the report.

SQL > BREAK ON DEPTNO SKIP 2 ON REPORT

SQL > COMPUTE AVG OF SAL ON REPORT
SQL > RUN

5.4 SQL*Plus Command Buffer

Oracle saves SQL*Plus commands like BREAK and COMPUTE in several SQL*Plus command buffers (one buffer for each different SQL*Plus command). You can display the contents of a SQL*Plus command buffer by typing the command name by itself.

- Display the current TTITLE

SQL > TTITLE

Oracle 'remembers' each of these commands and reuses them unless you turn them OFF or use a CLEAR command.

- Clear the current BREAK and COMPUTE commands.

SQL > CLEAR BREAK
SQL > CLEAR COMPUTE
SQL > RUN

The BREAK and COMPUTE are now gone but the TTITLE and BTITLE are still on.

- Turn off TTITLE and BTITLE

SQL > TTITLE OFF
SQL > BTITLE OFF
SQL > RUN

Lab 4 - Joining Two or More Tables and SQL Sub-queries

1 JOINING TWO OR MORE TALBES

All the previous query examples that you have worked on are related to only one table. The join feature of SQL lets you select data from two or more tables and combine the selected data into a single query result.

1.1 Relationships Between Tables

The relationships between rows in one table and rows in another table are represented by the values stored in fields of those rows. For example, the EMPLOYEE table and the DEPARTMENT table each have a column DEPTNO that contains department numbers. It is the department numbers stored in both these columns that allows you to relate rows in the department table to rows in the employee table.

1.2 Selecting Data From Two Or More Tables

Let's say you want to know the location of the employee named ALLEN. Note that the EMP table does not contain a location (LOC) column but the DEPT table does. By looking at the employee table, you can see that ALLEN works for department 30. By looking at the DEPT table you can see that the department 30 is located in CHICAGO. Thus, you were able to relate one row in the EMP table to another row in the DEPT table by using data in a column present in both tables: the DEPTNO column.
Similarly, Oracle can "look" at data values stored in tables, and use those values to relate or *join* rows in one table to a row in other tables. You list the tables to be joined in the FROM clause and the relationship between the tables in the WHERE clause.

- Find Allen's name from the EMP table and location of Allen's department from the DEPT table.(join.1)

```
SQL > SELECT ENAME, LOC
      2 FROM EMP, DEPT
      3 WHERE ENAME = 'ALLEN'
      4 AND EMP.DEPTNO = DEPT.DEPTNO;
```

The WHERE clause says to retrieve only the rows for employee ALLEN. The WHERE clause also specifies that if the department number of a row in the employee table is equal to the department number of a row in the department table (EMP.DEPTNO = DEPT.DEPTNO), then join the rows together. That is, join the ALLEN row from the EMP table to the department 30 row from the DEPT table. The SELECT clause directs that only the ENAME field from the EMP table and the LOC field from the DEPT table are to be retrieved from the joined row.

1.3 Prefixing Column Names With Table Names

Notice that the DEPTNO column name is prefixed with the table name EMP or DEPT (EMP.DEPTNO = DEPT.DEPTNO). This is because both the EMP table and the DEPT table have a column named DEPTNO. If a column name is unique among the tables listed in the FROM clause (ENAME for example) the column name need not be prefixed.

1.4 Equi-join

- Join the DEPT table to the EMP table.(join.2)

```
SQL > SELECT DEPT.DEPTNO,DNAME,JOB,ENAME
      2 FROM DEPT, EMP
```

```
3 WHERE DEPT.DEPTNO = EMP.DEPTNO
4 ORDER BY DEPT.DEPTNO;
```

There is only one search-condition in the WHERE clause in the above example and it specifies the relationship between the EMP table and the DEPT table. This special type of search-condition is called a *join-condition*. Specifically, this join is called an *equi-join* because the comparison operator in the join-condition is *equals*. In addition to the equi-join, there are several types of join-conditions, but most joins that you will do will be either equi-joins or *outer-joins*.

1.5 Using Table Labels To Abbreviate Table Names

Join queries can become rather tedious to type when column names have to be prefixed with table names.

- List the department name and all the fields from the employee table for employees that work in Chicago. (join.3)

```
SQL > SELECT DNAME, EMPNO, ENAME, JOB, MGR, HIREDATE,
2 SAL, COMM, EMP.DEPTNO
3 FROM EMP, DEPT
4 WHERE EMP.DEPTNO = DEPT.DEPTNO
5 AND LOC = 'CHICAGO'
6 ORDER BY EMP.DEPTNO;
```

SQL allows you to define a temporary label in the FROM clause by placing the label after the table name separated by a blank. You may then use these labels in place of the full table names within the query.

- Issue the same query as the last example but use temporary labels to abbreviate the table names. (join.4)

```
SQL > SELECT DNAME, E.*
2 FROM EMP E, DEPT D
3 WHERE E.DEPTNO = D.DEPTNO
4 AND LOC = 'CHICAGO'
5 ORDER BY E.DEPTNO;
```

In the example above the letter E refers to the EMP table and the letter D refers to the DEPT table. Also notice the use of E.* in the SELECT clause to retrieve all of the columns of the EMP table.

1.6 Joining A Table To Itself

You can use a table label for more than just abbreviating a table name in a query. It also lets you "join" a table to itself as thought it were two separate tables.

- For each employee whose salary exceeds his manager's salary, list the employees' names and salary and the manager's name and salary. (join.5)

```
SQL > SELECT WORKER.ENAME, WORKER.SAL, MANAGER.ENAME,
2 MANAGER.SAL
3 FROM EMP WORKER, EMP MANAGER
4 WHERE WORKER.MGR = MANAGER.EMPNO
5 AND WORKER.SAL > MANAGER.SAL;
```

In the above query, the EMP table is treated as if it were two separated tables named WORKER and MANAGER. Firstly all the WORKERs are joined to their

MANAGERs using the WORKER's manager's employee number (WORKER.MGR) and the MANAGER's employee number (MANAGER.EMPNO). For example, SCOTT's manager is JONES because SCOTT has a MGR column with a value of 7566 and JONES' EMPNO is 7566. The WHERE clause then eliminates all WORKER MANAGER pairs except those where the WORKER earn more than the manager (WORKER.SAL > MANAGER.SAL).
Note that the EMP table is *not physically* duplicated into two separate tables during the query processing.

1.7 Other Join Operator
All the examples so far show tables being joined using the equal comparison operator (including outer-join). As we said earlier, equal is by far the most common join-condition, but tables can be joined to another using any comparison operator including:

=	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
BETWEEN	
LIKE	

- Find all the employees that earn more then JONES.(join.6)

```
SQL > SELECT X.ENAME, X.SAL, X.JOB, Y.ENAME, Y.SAL, Y.JOB
  2 FROM EMP X, EMP Y
  3 WHERE X.SAL > Y.SAL
  4 AND Y.ENAME = 'JONES';
```

In the example above, the EMP table is joined to itself using the comparison operator greater than (>).

To demonstrate a BETWEEN join we will use a new table that contain salary grades.

- List the SALGRADE table. (join.7)

```
SQL > SELECT * FROM SALGRADE;
```

- Now find the salary grade of each employee by joining the EMP table to the SALGRADE table.(join.8)

```
SQL > SELECT GRADE, JOB, ENAME, SAL
  2 FROM EMP, SALGRADE
  3 WHERE SAL BETWEEN LOSAL AND HISAL
  4 ORDER BY GRADE, JOB;
```

In the BETWEEN join, the SAL field of each row of the employee table is "tested" to make sure it is both greater than or equal to LOSAL and less than or equal to HISAL of the SALOGRADE table.

1.8 Selecting All Possible Combinations of Rows
If the WHERE clause contains no join-condition at all, then all possible combinations of rows from tables listed in the FROM clause are displayed. This result, called a cartesian product, is normally not desired so that a join-condition is usually specified.

- Join the ALLEN row from the EMP table with all the rows of the DEPT table. (join. 9)

SQL > SELECT ENAME, LOC
2 FROM EMP, DEPT
3 WHERE ENAME = 'ALLEN';

2 SQL SUBQUERIES

One of the reasons why SQL is so powerful is that you can build complex queries out of several simple queries. This is possible because the WHERE clause of one query may contain another query (called a *sub-query*). Oracle uses the sub-query to "dynamically build" a search-condition from values stored in the database.

2.1 Sub-queries That Return Only One Value

Suppose you want to find all the employees that have the same job as JONES. To answer this question, you can issue the following two queries:

- Find JONES' job.(subq.1)

SQL > SELECT JOB
2 FROM EMP
3 WHERE ENAME = 'JONES';

- Now that the first query has informed you that JONES is a MANAGER, you can issue a second query to find all the manager. (subq.2)

SQL > SELECT ENAME, JOB
2 FROM EMP
3 WHERE JOB = 'MANAGER';

You can arrive at the same result that required the previous two queries by using a single query with an embedded sub-query.

- List the name and job of employees who have the same job as JONES. (subq.3)

SQL > SELECT ENAME, JOB FROM EMP
2 WHERE JOB =
3 (SELECT JOB FROM EMP WHERE ENAME = 'JONES');

The second SELECT command returns the value MANAGER. Oracle uses this value to build the search-condition WHERE JOB = 'MANAGER'. This dynamically constructed search-condition is then used by the first SELECT command (called the main query) to select the desired rows. Note that sub-queries must be enclosed in parentheses but need not be indented.

2.2 Sub-queries That Return A Set Of Values

If a sub-query may return a set of values (more than one) you must attach the word ANY or ALL to the comparison operator (=,!=,>,>=,<,<=) preceding the sub-query to clarify the meaning of your query.

- Find the employees that earn more than ANY employee in department 30. (subq.4)

SQL > SELECT DISTINCT SAL, JOB, ENAME, DEPTNO FROM EMP
2 WHERE SAL > ANY
3 (SELECT SAL FROM EMP
4 WHERE DEPTNO = 30)
5 ORDER BY SAL DESC;

If an employee's salary is more than ANY of the set of salaries returned by the sub-query, then that employee is included in the query result. The lowest salary in department 30 is 950(JAMES). Thus the main query returns only those employees that earn more than 950.

The next query returns only those employees that earn more than ALL employees in department 30.

- Find the employees that earn more then ALL employees in department 30. (subq.5)

```
SQL > SELECT SAL, JOB, ENAME, DEPTNO FROM EMP
  2 WHERE SAL > ALL
  3 (SELECT SAL FROM EMP
  4 WHERE DEPTNO = 30)
  5 ORDER BY SAL DESC;
```

Blake, the manager of department 30, earns 2,850. Thus the query above returns only those employees that earn more than 2,850.

The operator IN has the same meaning and may be substituted for =ANY and the operator NOT IN is the same as !=ALL.

- Find all the employees in department 10 that have a job that is the same as anyone in department 30. (subq.6)

```
SQL > SELECT ENAME, JOB FROM EMP
  2 WHERE DEPTNO = 10
  3 AND JOB IN
  4 (SELECT JOB FROM EMP
  5 WHERE DEPTNO = 30);
```

Another way to state the above query is, "Find all employees with a job that is IN the set of jobs returned by the sub-query." The sub-query returns a set of all the jobs held by employees in department 30. The main query "tests" each employees in department 10 to see if his job is in the set of jobs returned by the sub-query.

- Find all the employees in department 10 that have a job that is NOT the same as anyone in department 10. (subq.7)

```
SQL > SELECT ENAME, JOB FROM EMP
  2 WHERE DEPTNO = 10
  3 AND JOB NOT IN
  4 (SELECT JOB FROM EMP
  5 WHERE DEPTNO = 30);
```

Another way to state the above query is, "Find all employees with a job that is NOT IN the set of jobs returned by the sub-query".

2.3 Sub-queries That Return More Than One Column

Oracle allows you to have more then one column in the SELECT list of the sub-query.

- List the name, job title, and salary of employees who have the same job and salary as Ford. (subq.8)

```
SQL > SELECT ENAME, JOB, SAL
  2 FROM EMP
```

```
3 WHERE (JOB,SAL)=
4 (SELECT JOB, SAL FROM EMP
5 WHERE ENAME = 'FORD');
```

Parentheses must be used to enclose the SELECT list of a sub-query when it contains more then one column.

2.4 Compound Queries with Multiple Sub-queries

SQL lets you combine basic commands to construct more and more complex commands. It is in this way that SQL obtains its power without sacrificing simplicity. The WHERE clause of a query may contain a combination of standard search-conditions, join-conditions and multiple sub-queries as shown in the following examples. Oracle does not limit you to just one sub-query.

- List the name, job, and department of employees who have the same job as Jones, or a salary greater than or equal to Ford. (subq.9)

```
SQL > SELECT ENAME, JOB, DEPTNO, SAL FROM EMP
      2 WHERE JOB IN
      3 (SELECT JOB FROM EMP
      4 WHERE ENAME = 'JONES')
      5 OR SAL > =
      6 (SELECT SAL FROM EMP
      7 WHERE ENAME = 'FORD')
      8 ORDER BY JOB, SAL;
```

You may have up to 16 sub-queries "linked" to the next higher-level query.

- Find all the employees in department 10 that have a job that is the same as anyone in the SALES department. (subq.10)

```
SQL > SELECT ENAME, JOB FROM EMP
      2 WHERE DEPTNO = 10
      3 AND JOB IN
      4 (SELECT JOB FROM EMP
      5 WHERE DEPTNO IN
      6 (SELECT DEPTNO FROM DEPT
      7 WHERE DNAME = 'SALES'));
```

2.5 Synchronizing A Repeating Sub-query With A Main Query

Depending on how you structure your sub-query, it can operate in different ways. In the previous examples, the sub-query was executed once and the resulting value was substituted into the WHERE clause of the main query. The following example shows a sub-query that is *executed repeatedly*, once for each row considered for selection (called the *candidate row*) by the main query.

Let's say you want to find the department number, name and salary of the employees that earn more than the average salary in their department. Firstly you need a main query to select the desired data from the EMP table.

```
SELECT    DEPTNO,ENAME,SAL
FROM      EMP
WHERE     SAL > (average salary of candidate employee's department)
```

Next you need a sub-query that will calculate the average salary of an employee's department as that employee is being considered for selection by the main query.

```
SELECT   DEPTNO, ENAME, SAL
FROM     EMP X
WHERE    SAL >
         (SELCT AVG(SAL)
         FROM EMP
         WHERE DEPTNO = (department of candidate employee))
```

The sub-query does not know which department average to compute until it knows the DEPTNO of the candidate employee being processed by the main query. The main query tells the sub-query which department average to compute. The sub-query computes the average salary for the candidate employee's department. The main-query then compares the average salary with the candidate employee's salary.

- Find all the employees that earn more than the average salary of employees in their department. (subq.11)

```
SQL > SELECT DEPTNO, ENAME, SAL FROM EMP X
  2 WHERE SAL >
  3 (SELECT AVG(SAL) FROM EMP
  4 WHERE X.DEPTNO = DEPTNO)
  5 ORDER BY DEPTNO;
```

The table label X in the main query and WHERE clause in the sub-query tell Oracle to "synchronize" the sub-query with the main memory. Oracle calculates the average salary in the sub-query using the DEPTNO of the employee being considered for selection by the main query. X.DEPTNO in the WHERE clause of the sub-query refers to the candidate row's department number and specifies that it must be equal to the DEPTNO used to select rows for computing average salary in the sub-query.

3. CREATE VIEWS

A view is a window into the data in one or more tables. The view itself contains no data and takes up no space.

Syntax:

```
CREATE VIEW view_name
[(column_name[,column_name]…)]
AS SELECT_STATEMENT;
```

- Create a view with DEPTNO, DNAME, ENAME, JOB from the DEPT and EMP tables.

```
SQL > CREATE VIEW DEPT_EMP
  2 AS SELECT DEPT.DEPTNO, DNAME, ENAME, JOB
  3 FROM DEPT, EMP
  4 WHERE DEPT.DEPTNO = EMP.DEPTNO;

SQL > SELECT * FROM DEPT_EMP;
```