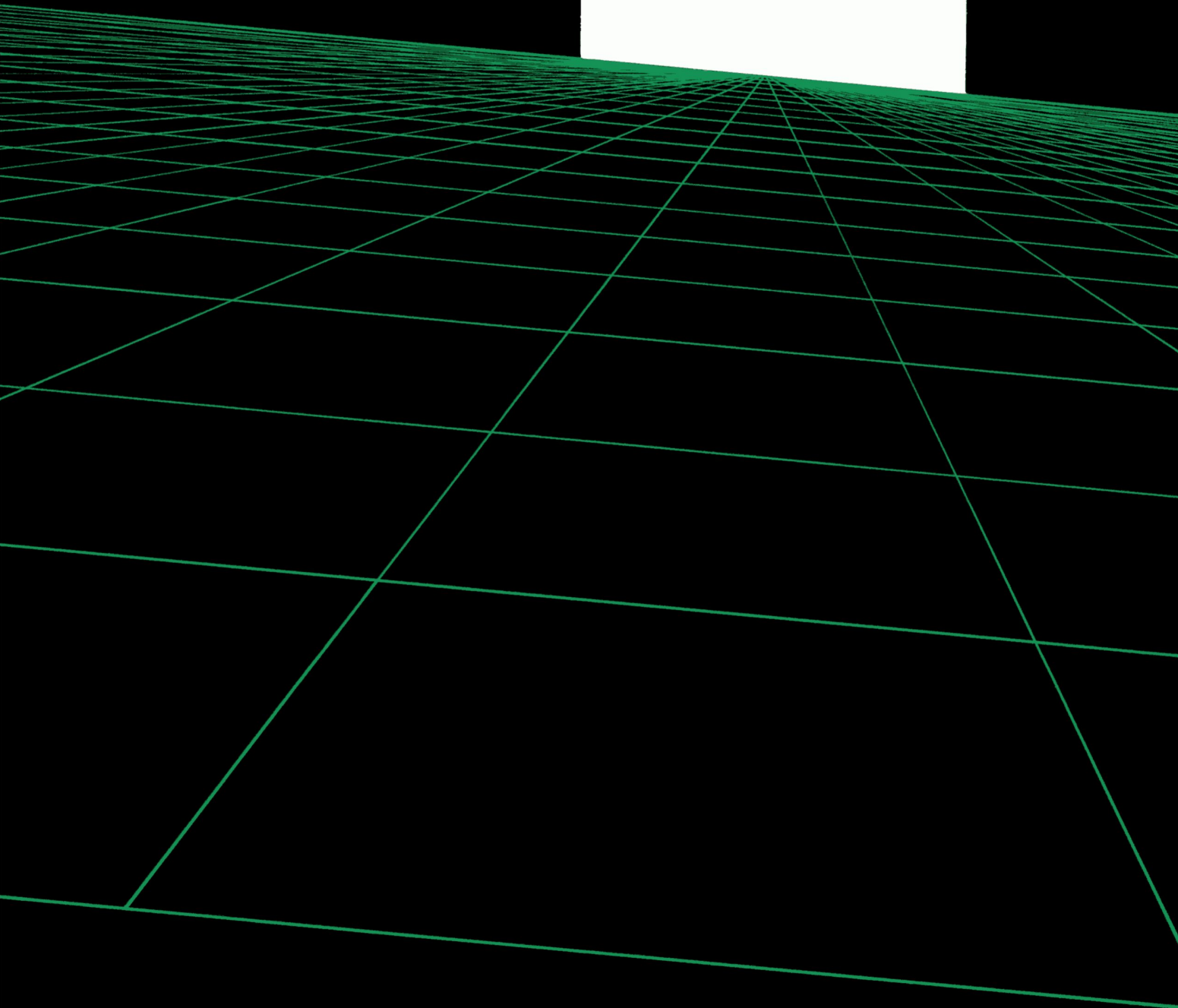




Relational Software Incorporated

ORACLE



ORACLE



Users' Guide

Relational Software Incorporated

RSI-100

R S I

O R A C L E

U S E R S - G U I D E

Oracle Users Guide - Version 2.3

Copyright (c) April 1981
By Relational Software Incorporated
All rights reserved. Printed in U.S.A.

R S I

ORACLE

SQL LANGUAGE

USER'S GUIDE

Oracle User's Guide - Version 2.3

Copyright (c) April 1981
By Relational Software Incorporated
All rights reserved. Printed in U.S.A.

S Q L

LANGUAGE USER'S GUIDE

TABLE OF CONTENTS

INTRODUCTION	1-1
DATA BASE CONCEPTS	1-3
QUERY FACILITIES	1-6
DATA MANIPULATION FACILITIES	1-38
DATA DEFINITION FACILITIES	1-45
DATA STRUCTURES	1-55
DATA INDEPENDENCE	1-59
TREE-STRUCTURED TABLES	1-61
SECURITY FACILITIES	1-71
DATA DICTIONARY STRUCTURE	1-82
CONCURRENCY CONTROL FACILITIES	1-94

SQL LANGUAGE - USER'S GUIDE

SQL LANGUAGE - EXAMPLES

INTRODUCTION

This section of the "User's Guide" introduces the SQL facilities for query, data manipulation, data definition and data control through a series of examples. The examples are based on the following database:

```

EMP      +-----+
        | EMPNO | ENAME | JOB  | SAL | COMM | DEPTNO |
        +-----+
DEPT     +-----+
        | DEPTNO | DNAME | LOC  | EMPcnt |
        +-----+
BONUS    +-----+
        | ENAME  | JOB   | SAL  | COMM |
        +-----+

```

The EMP table contains information on employees, giving the employee's number, name, job title, salary, commission, and department number. The department table gives the department's number, name, location, and a count of the employees in the department. The BONUS table contains an extract of the information in the EMP table. Through the course of the examples, the EMP table is expanded to contain a project number column and a supervisor column, and two new tables are added. The PROJ table contains columns for the project number, name, and budget. The PE table relates projects to employees where one project can have many employees, and one employee can be working on many projects. The expanded EMP table and the two added tables are as follows:

```

EMP      +-----+
        | EMPNO | ENAME | JOB  | SAL | COMM | DEPTNO | PROJNO | SUPR |
        +-----+
PROJ     +-----+
        | PROJNO | PNAME | BUDGET |
        +-----+
PE       +-----+
        | PROJNO | EMP   |
        +-----+

```

The examples in this section of the documentation were produced by executing the ORACLE User Friendly Interface (UFI) and capturing the output in hard-copy form. The actual output has been post-processed to add page numbers and to remove the SQL prompts at the beginning of each line. The command file which produces this output is provided with each ORACLE installation. It contains the comments interspersed with the SQL statements. This section of the documentation serves as a self-tutorial in the facilities of SQL as implemented in ORACLE.

ORACLE accepts SQL statements in free format. The arrangement of each SQL clause on a separate line and indentation in the following examples is used for clarity only.

1. DATA BASE CONTENTS

This section of the manual describes the structure and content of the example PERSONNEL data base.

1.1 Dictionary Contents

ORACLE's integrated data dictionary can be queried using standard SQL query facilities.

Example 1-1: List the names of the user tables in the data base.

```
SQL>SELECT  TABLE
SQL>FROM    TAB;
SQL>/
```

```
TABLE
-----
EMP
DEPT
BONUS
```

Example 1-2: List the names of the columns of the department and employee tables.

```
SQL>SELECT *
SQL>FROM COL
SQL>WHERE TABLE = 'DEPT';
SQL>/
```

TABLE	COLUMN
DEPT	DEPTNO
DEPT	DNAME
DEPT	LOC
DEPT	EMPCNT

```
SQL>SELECT *
SQL>FROM COL
SQL>WHERE TABLE = 'EMP';
SQL>/
```

TABLE	COLUMN
EMP	EMPNO
EMP	ENAME
EMP	JOB
EMP	SAL
EMP	COMM
EMP	DEPTNO

6 records selected.

1.2 Data Base Contents

Example 1-3: List all the columns and rows of the DEPT table.

```
SQL>SELECT * FROM DEPT;
SQL>/
```

DEPTNO	DNAME	LOC	EMPCNT
10	ADMINISTRATION	NEW YORK	
20	RESEARCH	SAN FRANCISCO	
30	SALES	CHICAGO	
40	OPERATIONS	BOSTON	

Example 1-4: List all the columns and rows of the EMP table.

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

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
7369	SMITH	CLERK	\$800.00		20
7499	ALLEN	SALESMAN	\$1,600.00	\$300.00	30
7521	WARD	SALESMAN	\$1,250.00	\$500.00	30
7566	JONES	MANAGER	\$2,975.00		20
7654	MARTIN	SALESMAN	\$1,250.00	\$1,400.00	30
7698	BLAKE	MANAGER	\$2,850.00		30
7782	CLARK	MANAGER	\$2,450.00		10
7788	SCOTT	ANALYST	\$3,000.00		20
7839	OATES	PRESIDENT	\$5,000.00		10
7844	TURNER	SALESMAN	\$1,500.00	\$0.00	30
7876	ADAMS	CLERK	\$1,100.00		20
7900	JAMES	CLERK	\$950.00		30
7902	FORD	ANALYST	\$3,000.00		20
7934	MILLER	CLERK	\$1,300.00		10

14 records selected.

2. QUERY FACILITIES

This section of the manual contains a description of the query facilities of ORACLE.

2.1 Query Block

The SELECT clause lists the columns to be returned. The FROM clause lists the tables involved in the query. The WHERE clause specifies the selection criteria.

Example 2-1: Find the name of department 10.

```
SQL>SELECT  DNAME
SQL>FROM    DEPT
SQL>WHERE   DEPTNO=10;
SQL>/
```

```
DNAME
-----
ADMINISTRATION
```

The SELECT clause can contain several columns. Character string constants are enclosed by single quotation marks.

Example 2-2: List the names, numbers, and departments of all clerks.

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

```
ENAME  EMPNO  DEPTNO
-----  -
SMITH   7369     20
ADAMS   7876     20
MILLER  7934     10
JAMES   7900     30
```

If all columns of the row are to be returned, `SELECT *` is specified.

Example 2-3: List all the columns in the employee table for employees in department 30.

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

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	\$1,600.00	\$300.00	30
7521	WARD	SALESMAN	\$1,250.00	\$500.00	30
7698	BLAKE	MANAGER	\$2,850.00		30
7654	MARTIN	SALESMAN	\$1,250.00	\$1,400.00	30
7844	TURNER	SALESMAN	\$1,500.00	\$0.00	30
7900	JAMES	CLERK	\$950.00		30

6 records selected.

The `WHERE` clause can compare two fields of a row with each other.

Example 2-4: List the name, salary, and commission of each employee whose commission is greater than his salary.

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

ENAME	SAL	COMM
MARTIN	\$1,250.00	\$1,400.00

The absence of a WHERE clause causes all rows to be returned.

Example 2-5: List all columns and all rows in the DEPT table.

```
SQL>SELECT *
SQL>FROM DEPT;
SQL>/
```

DEPTNO	DNAME	LOC	EMPCNT
10	ADMINISTRATION	NEW YORK	
20	RESEARCH	SAN FRANCISCO	
30	SALES	CHICAGO	
40	OPERATIONS	BOSTON	

2.2 Logical Expressions

Predicates within the WHERE clause may be connected by the boolean operators, AND and OR.

Example 2-6: List the name, job title, and salary of all employees in department 20 that make more than \$2,000.

```
SQL>SELECT ENAME,JOB,SAL
SQL>FROM EMP
SQL>WHERE DEPTNO = 20
SQL> AND SAL > 2000;
SQL>/
```

ENAME	JOB	SAL
JONES	MANAGER	\$2,975.00
SCOTT	ANALYST	\$3,000.00
FORD	ANALYST	\$3,000.00

The BETWEEN operator simplifies the syntax for specifying a range. The predicates SAL >= 1200 AND SAL <= 1400 are simplified to SAL BETWEEN 1200 AND 1400.

Example 2-7: List the name, job title, and salary of all employees who earn between \$1,200 AND \$1,400.

```
SQL>SELECT  ENAME,JOB,SAL
SQL>FROM    EMP
SQL>WHERE   SAL BETWEEN 1200 AND 1400;
SQL>/
```

ENAME	JOB	SAL
WARD	SALESMAN	\$1,250.00
MARTIN	SALESMAN	\$1,250.00
MILLER	CLERK	\$1,300.00

Example 2-8: List the name, job, salary, and commission of each employee whose job title begins with MAN or whose salary is greater than 3000 a month.

```
SQL>SELECT  ENAME,JOB,SAL,COMM
SQL>FROM    EMP
SQL>WHERE   JOB = 'MAN...'
SQL> OR    SAL > 3000;
SQL>/
```

ENAME	JOB	SAL	COMM
JONES	MANAGER	\$2,975.00	
BLAKE	MANAGER	\$2,850.00	
CLARK	MANAGER	\$2,450.00	
OATES	PRESIDENT	\$5,000.00	

Example 2-9: List the department number, name, job title, salary and commission of each employee in department 30 whose salary is greater than his commission.

```
SQL>SELECT DEPTNO,ENAME,JOB,SAL,COMM
SQL>FROM EMP
SQL>WHERE SAL > COMM
SQL> AND DEPTNO = 30;
SQL>/
```

DEPTNO	ENAME	JOB	SAL	COMM
30	ALLEN	SALESMAN	\$1,600.00	\$300.00
30	WARD	SALESMAN	\$1,250.00	\$500.00
30	TURNER	SALESMAN	\$1,500.00	\$0.00

Predicates within a WHERE clause form logical expressions with square brackets [] used to establish precedence.

Example 2-10: List the department number, name, job title, salary, and commission of all analysts, or all people in department 10 who earn more than \$2,500.

```
SQL>SELECT DEPTNO,ENAME,JOB,SAL,COMM
SQL>FROM EMP
SQL>WHERE JOB = 'ANALYST'
SQL> OR [SAL > 2500 AND DEPTNO = 10];
SQL>/
```

DEPTNO	ENAME	JOB	SAL	COMM
20	SCOTT	ANALYST	\$3,000.00	
20	FORD	ANALYST	\$3,000.00	
10	OATES	PRESIDENT	\$5,000.00	

Example 2-11: List the department number, name, job title, salary, and commission of employees in department 10 who are either analysts or earn more than \$2,500.

```
SQL>SELECT DEPTNO,ENAME,JOB,SAL,COMM
SQL>FROM EMP
SQL>WHERE [JOB = 'ANALYST'
SQL> OR SAL > 2500] AND DEPTNO = 10;
SQL>/
```

DEPTNO	ENAME	JOB	SAL	COMM
10	OATES	PRESIDENT	\$5,000.00	

2.3 Not Conditions

Predicates within a WHERE clause may be negated.

Example 2-12: List the name, salary, commission, job title, and department number of employees in department 30 who are not a salesman.

```
SQL>SELECT  ENAME,SAL,COMM,JOB,DEPTNO
SQL>FROM    EMP
SQL>WHERE    JOB ^= 'SALESMAN'
SQL>  AND    DEPTNO = 30;
SQL>/
```

ENAME	SAL	COMM	JOB	DEPTNO
BLAKE	\$2,850.00		MANAGER	30
JAMES	\$950.00		CLERK	30

Entire logical expressions may be negated.

Example 2-13: List the name, salary, commission, job title, and department number of employees in department 30 who are not a salesman or do not earn more than \$1,500.

```
SQL>SELECT  ENAME,SAL,COMM,JOB,DEPTNO
SQL>FROM    EMP
SQL>WHERE    NOT [JOB = 'SALESMAN' OR SAL > 1500]
SQL>  AND    DEPTNO = 30;
SQL>/
```

ENAME	SAL	COMM	JOB	DEPTNO
JAMES	\$950.00		CLERK	30

2.4 Set Inclusion Operator

A predicate in a WHERE clause may test a field for inclusion in a set of constant literal values. A set of constant literal values is enclosed within angle brackets < >.

Example 2-14: List the name and department number of employees in departments 10 and 30.

```
SQL>SELECT  ENAME,DEPTNO
SQL>FROM    EMP
SQL>WHERE    DEPTNO IN <10,30>;
SQL>/
```

ENAME	DEPTNO
CLARK	10
MILLER	10
OATES	10
ALLEN	30
WARD	30
BLAKE	30
MARTIN	30
TURNER	30
JAMES	30

9 records selected.

Example 2-15: List all fields from the department table for departments that are located in either Chicago or New York.

```
SQL>SELECT  *
SQL>FROM    DEPT
SQL>WHERE    LOC IN <'CHICAGO','NEW YORK'>;
SQL>/
```

DEPTNO	DNAME	LOC	EMPCNT
30	SALES	CHICAGO	
10	ADMINISTRATION	NEW YORK	

2.5 Nested Query

The result of one query may be used in the WHERE clause of another query. The inner query returns one or a set of values. The outer query uses this result as if it were given a set of constant literal values. Query blocks may be nested to any number of levels.

Example 2-16: List the name and job of employees who have the same job as Jones.

```
SQL>SELECT  ENAME,JOB
SQL>FROM      EMP
SQL>WHERE     JOB IN
SQL>          SELECT  JOB
SQL>          FROM      EMP
SQL>          WHERE     ENAME = 'JONES';
SQL>/
```

ENAME	JOB
JONES	MANAGER
BLAKE	MANAGER
CLARK	MANAGER

Example 2-17: List the name, job title, and salary of employees who have the same job and salary as Ford.

```
SQL>SELECT  ENAME,JOB,SAL
SQL>FROM      EMP
SQL>WHERE     <JOB,SAL> =
SQL>          SELECT  JOB,SAL
SQL>          FROM      EMP
SQL>          WHERE     ENAME = 'FORD';
SQL>/
```

ENAME	JOB	SAL
SCOTT	ANALYST	\$3,000.00
FORD	ANALYST	\$3,000.00

Inner query blocks may be connected by the boolean operators AND and OR to form compound nested queries.

Example 2-18: List the name, job, and department of employees who have the same job as Jones, or a salary greater than or equal to Ford.

```
SQL>SELECT  ENAME,JOB,DEPTNO,SAL
SQL>FROM    EMP
SQL>WHERE    JOB IN
SQL>          SELECT  JOB
SQL>          FROM    EMP
SQL>          WHERE   ENAME = 'JONES';
SQL>OR SAL >=
SQL>          SELECT  SAL
SQL>          FROM    EMP
SQL>          WHERE   ENAME = 'FORD';
SQL>/
```

ENAME	JOB	DEPTNO	SAL
JONES	MANAGER	20	\$2,975.00
BLAKE	MANAGER	30	\$2,850.00
CLARK	MANAGER	10	\$2,450.00
SCOTT	ANALYST	20	\$3,000.00
OATES	PRESIDENT	10	\$5,000.00
FORD	ANALYST	20	\$3,000.00

6 records selected.

2.6 ORDER BY

The ORDER BY clause specifies major and minor sort fields in ascending or descending order. Ascending order is default.

Example 2-19: List the name, job, department, and employee number of employees in a department whose number is greater than or equal to 20, in order of employee name.

```
SQL>SELECT  ENAME,JOB,DEPTNO,EMPNO
SQL>FROM    EMP
SQL>WHERE    DEPTNO >= 20
SQL>ORDER BY ENAME;
SQL>/
```

ENAME	JOB	DEPTNO	EMPNO
ADAMS	CLERK	20	7876
ALLEN	SALESMAN	30	7499
BLAKE	MANAGER	30	7698
FORD	ANALYST	20	7902
JAMES	CLERK	30	7900
JONES	MANAGER	20	7566
MARTIN	SALESMAN	30	7654
SCOTT	ANALYST	20	7788
SMITH	CLERK	20	7369
TURNER	SALESMAN	30	7844
WARD	SALESMAN	30	7521

11 records selected.

Example 2-20: List the department, salary, name, and job of all employees, in descending order by salary, ascending order by job within salary, and ascending order by name within job.

```
SQL>SELECT DEPTNO,SAL,JOB,ENAME
SQL>FROM EMP
SQL>ORDER BY SAL DESC,JOB,ENAME;
SQL>/
```

DEPTNO	SAL	JOB	ENAME
10	\$5,000.00	PRESIDENT	OATES
20	\$3,000.00	ANALYST	FORD
20	\$3,000.00	ANALYST	SCOTT
20	\$2,975.00	MANAGER	JONES
30	\$2,850.00	MANAGER	BLAKE
10	\$2,450.00	MANAGER	CLARK
30	\$1,600.00	SALESMAN	ALLEN
30	\$1,500.00	SALESMAN	TURNER
10	\$1,300.00	CLERK	MILLER
30	\$1,250.00	SALESMAN	MARTIN
30	\$1,250.00	SALESMAN	WARD
20	\$1,100.00	CLERK	ADAMS
30	\$950.00	CLERK	JAMES
20	\$800.00	CLERK	SMITH

14 records selected.

Expressions can be specified within the ORDER BY clause.

Example 2-21: List all salesman in ascending order of the ratio of their salary divided by their commission.

```
SQL>SELECT ENAME,SAL/COMM,SAL,COMM
SQL>FROM EMP
SQL>WHERE JOB = 'SALESMAN'
SQL>ORDER BY SAL/COMM
SQL>/
```

ENAME	SAL/COMM	SAL	COMM
MARTIN	.893	\$1,250.00	\$1,400.00
WARD	2.500	\$1,250.00	\$500.00
ALLEN	5.333	\$1,600.00	\$300.00
TURNER	~.000	\$1,500.00	\$0.00

2.7 UNIQUE

A query returns a set of rows that satisfy the WHERE clause. Duplicate rows are not eliminated unless SELECT UNIQUE is specified.

Example 2-22: List all the different jobs in the job table.

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

```
JOB
-----
CLERK
SALESMAN
MANAGER
ANALYST
PRESIDENT
```

If a WHERE clause has multiple predicates connected by an OR, there exists the possibility that a single row may satisfy both predicates and be returned twice in the query result. UNIQUE specified within the SELECT clause eliminates this duplication.

Example 2-23: List the name and job of employees who are in department 30, or employees who are managers. Sort by employee name.

```
SQL>SELECT  ENAME,JOB,DEPTNO
SQL>FROM    EMP
SQL>WHERE    DEPTNO = 30
SQL>  OR    JOB = 'MANAGER'
SQL>ORDER BY ENAME;
SQL>/
```

ENAME	JOB	DEPTNO
-----	-----	-----
ALLEN	SALESMAN	30
BLAKE	MANAGER	30
BLAKE	MANAGER	30
CLARK	MANAGER	10
JAMES	CLERK	30
JONES	MANAGER	20
MARTIN	SALESMAN	30
TURNER	SALESMAN	30
WARD	SALESMAN	30

9 records selected.

Example 2-24: List the name and job of employees who are in department 30 or, employees who are managers. Eliminate duplicate rows and sort the result by employee name.

```
SQL>SELECT  UNIQUE ENAME,JOB,DEPTNO
SQL>FROM      EMP
SQL>WHERE     DEPTNO = 30
SQL>  OR      JOB = 'MANAGER'
SQL>ORDER BY ENAME;
SQL>/
```

ENAME	JOB	DEPTNO
ALLEN	SALESMAN	30
BLAKE	MANAGER	30
CLARK	MANAGER	10
JAMES	CLERK	30
JONES	MANAGER	20
MARTIN	SALESMAN	30
TURNER	SALESMAN	30
WARD	SALESMAN	30

8 records selected.

Example 2-25: List the salary, job title, name, and department number for all employees in departments that have salesmen. Sort the results by salary in descending order.

```
SQL>SELECT  SAL,JOB,ENAME,DEPTNO
SQL>FROM      EMP
SQL>WHERE     DEPTNO IN
SQL>          SELECT  UNIQUE DEPTNO
SQL>          FROM      EMP
SQL>          WHERE     JOB = 'SALESMAN';
SQL>ORDER BY SAL DESC;
SQL>/
```

SAL	JOB	ENAME	DEPTNO
\$2,850.00	MANAGER	BLAKE	30
\$1,600.00	SALESMAN	ALLEN	30
\$1,500.00	SALESMAN	TURNER	30
\$1,250.00	SALESMAN	WARD	30
\$1,250.00	SALESMAN	MARTIN	30
\$950.00	CLERK	JAMES	30

6 records selected.

2.8 Arithmetic Expressions

The SELECT, WHERE, and HAVING clauses may all contain arithmetic expressions containing fields and constants.

Example 2-26: List the name, salary, commission, and sum of salary plus commission of employees in department 30.

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

ENAME	SAL	COMM	SAL+COMM
ALLEN	\$1,600.00	\$300.00	\$1,900.00
WARD	\$1,250.00	\$500.00	\$1,750.00
BLAKE	\$2,850.00		
MARTIN	\$1,250.00	\$1,400.00	\$2,650.00
TURNER	\$1,500.00	\$0.00	\$1,500.00
JAMES	\$950.00		

6 records selected.

Example 2-27: List the name, salary, and commission of employees whose commission is greater than or equal to 25% of their salary.

```
SQL>SELECT  ENAME,SAL,COMM
SQL>FROM      EMP
SQL>WHERE     COMM >= 0.25 * SAL;
SQL>/
```

ENAME	SAL	COMM
WARD	\$1,250.00	\$500.00
MARTIN	\$1,250.00	\$1,400.00

Parentheses are used to establish precedence within arithmetic expressions.

Example 2-28: List the name, salary, commission, and 1.25 times salary plus two-thirds of the commission of all salesmen.

```
SQL>SELECT ENAME,SAL,COMM,((SAL * 1.25) + (COMM * ( 2/3 )))
SQL>FROM EMP
SQL>WHERE JOB = 'SALES...';
SQL>/
```

ENAME	SAL	COMM	((SAL*1.25)+(COMM*(2/3)))
ALLEN	\$1,600.00	\$300.00	\$2,200.00
WARD	\$1,250.00	\$500.00	\$1,895.83
MARTIN	\$1,250.00	\$1,400.00	\$2,495.83
TURNER	\$1,500.00	\$0.00	\$1,875.00

2.9 Built-In Functions

ORACLE provides several built-in functions that may be used in either SELECT or HAVING clauses.

Example 2-29: Find the average salary of all employees who are clerks.

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

```
      AVG(SAL)
-----
    $1,037.50
```

Example 2-30: Find the maximum, average, and minimum salary of employees in department 10.

```
SQL>SELECT  MAX(SAL),AVG(SAL),MIN(SAL)
SQL>FROM    EMP
SQL>WHERE    DEPTNO = 10;
SQL>/
```

```
      MAX(SAL)      AVG(SAL)      MIN(SAL)
-----
    $5,000.00    $2,916.67    $1,300.00
```

Example 2-31: Find the sum of all salesmen's commissions.

```
SQL>SELECT  SUM(COMM)
SQL>FROM    EMP
SQL>WHERE    JOB = 'SALESMAN';
SQL>/
```

```
      SUM(COMM)
-----
    $2,200.00
```

Example 2-32: Find the number of employees in department 30.

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

```
COUNT(*)
-----
        6
```

Built-in functions can be used in arithmetic expressions.

Example 2-33: Compute the average annual salary plus commission for all salesmen.

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

```
AVG(SAL+COMM)*12
-----
$23,400.00
```

ORACLE allows functions to be applied to the results of other built-in functions.

Example 2-34: List the name, job, and salary of the employee who has the largest salary.

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

```
ENAME  JOB  SAL
-----
OATES  PRESIDENT  $5,000.00
```

2.10 GROUP-BY

A table may be partitioned into groups according to the values in a column or set of columns. A built-in function may then be applied to each group. When a built-in function is used, each item in the SELECT clause must be a unique property of the group.

Example 2-35: List the department number and average salary of each department.

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

DEPTNO	AVG(SAL)
10	\$2,916.67
20	\$2,175.00
30	\$1,566.67

Example 2-36: List the department number and average annual salary of each departments employees, excluding managers salaries.

```
SQL>SELECT  DEPTNO,AVG(SAL) * 12
SQL>FROM      EMP
SQL>WHERE     NOT JOB = 'MAN...'
SQL>GROUP BY DEPTNO;
SQL>/
```

DEPTNO	AVG(SAL)*12
10	\$37,800.00
20	\$23,700.00
30	\$15,720.00

A table can be partitioned into groups based on the values in more than one column.

Example 2-37: Divide all employees into groups by department, and by job within department. Count the employees in each group and compute each group's average salary.

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

DEPTNO	JOB	COUNT(*)	AVG(SAL)*12
10	CLERK	1	\$15,600.00
10	MANAGER	1	\$29,400.00
10	PRESIDENT	1	\$60,000.00
20	ANALYST	2	\$36,000.00
20	CLERK	2	\$11,400.00
20	MANAGER	1	\$35,700.00
30	CLERK	1	\$11,400.00
30	MANAGER	1	\$34,200.00
30	SALESMAN	4	\$16,800.00

9 records selected.

Built-in functions can be applied to the results of other group functions to form functions of functions.

Example 2-38: Total the salaries of all the departments and list the department with the maximum total.

```
SQL>SELECT DEPTNO,MAX(SUM(SAL))
SQL>FROM EMP
SQL>GROUP BY DEPTNO;
SQL>/
```

DEPTNO	MAX(SUM(SAL))
30	\$10,875.00

2.11 HAVING

After a table has been partitioned into groups, a predicate or set of predicates in a HAVING clause can be applied to the groups.

Example 2-39: List the average annual salary for all job groups having more than 2 employees in the group.

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

JOB	AVG(SAL)*12
CLERK	\$12,450.00
MANAGER	\$33,100.00
SALESMAN	\$16,800.00

A query block may contain both a WHERE and HAVING clause. First, the WHERE clause is applied to qualify rows; second, the groups are formed and the built-in functions are computed; third, the HAVING clause is applied to qualify groups.

Example 2-40: List all the departments that have more than two clerks.

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

```
DEPTNO
-----
20
```

HAVING clauses may contain query blocks.

Example 2-41: List the departments and their average salary that have a greater average salary than department 20.

```
SQL>SELECT  DEPTNO,AVG(SAL)
SQL>FROM      EMP
SQL>GROUP BY DEPTNO
SQL>HAVING   AVG(SAL) >
SQL>        SELECT  AVG(SAL)
SQL>        FROM      EMP
SQL>        WHERE    DEPTNO = 20;
SQL>/
```

```
DEPTNO    AVG(SAL)
-----
      10  $2,916.67
```

2.12 NULL Conditions

Predicates within a WHERE clause can explicitly test for null conditions within a column.

Example 2-42: List the name, salary, commission and job title of all employees who do not receive commissions.

```
SQL>SELECT  ENAME,SAL,COMM,JOB
SQL>FROM      EMP
SQL>WHERE     COMM = NULL;
SQL>/
```

ENAME	SAL	COMM	JOB
SMITH	\$800.00		CLERK
JONES	\$2,975.00		MANAGER
BLAKE	\$2,850.00		MANAGER
CLARK	\$2,450.00		MANAGER
SCOTT	\$3,000.00		ANALYST
OATES	\$5,000.00		PRESIDENT
ADAMS	\$1,100.00		CLERK
JAMES	\$950.00		CLERK
FORD	\$3,000.00		ANALYST
MILLER	\$1,300.00		CLERK

10 records selected.

Example 2-43: List the name, salary, commission, and job title of those employees who receive a commission.

```
SQL>SELECT  ENAME,SAL,COMM,JOB
SQL>FROM    EMP
SQL>WHERE    NOT COMM = NULL;
SQL>/
```

ENAME	SAL	COMM	JOB
ALLEN	\$1,600.00	\$300.00	SALESMAN
WARD	\$1,250.00	\$500.00	SALESMAN
MARTIN	\$1,250.00	\$1,400.00	SALESMAN
TURNER	\$1,500.00	\$0.00	SALESMAN

Example 2-44: If a predicate within a WHERE or HAVING clause can be expressed without using NOT or NULL, it is usually more efficient.

```
SQL>SELECT  ENAME,SAL,COMM,JOB
SQL>FROM    EMP
SQL>WHERE    COMM >= 0;
SQL>/
```

ENAME	SAL	COMM	JOB
ALLEN	\$1,600.00	\$300.00	SALESMAN
WARD	\$1,250.00	\$500.00	SALESMAN
MARTIN	\$1,250.00	\$1,400.00	SALESMAN
TURNER	\$1,500.00	\$0.00	SALESMAN

Null values in the data base are treated as unknowns in the evaluation of logical expressions. Only those rows that are known to satisfy the WHERE clause are returned as the result of a query.

Example 2-45: List all columns of the employee table for employees in department 30 or employees whose commission is less than or equal to \$1,000.

```
SQL>SELECT *
SQL>FROM EMP
SQL>WHERE DEPTNO = 30
SQL> AND COMM <= 1000;
SQL>/
```

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	\$1,600.00	\$300.00	30
7521	WARD	SALESMAN	\$1,250.00	\$500.00	30
7844	TURNER	SALESMAN	\$1,500.00	\$0.00	30

Example 2-46: List all the columns of the employee table for employees in department 30 or employees whose commission is less than or equal to \$1,000.

```
SQL>SELECT *
SQL>FROM EMP
SQL>WHERE DEPTNO = 30
SQL> OR COMM <= 1000;
SQL>/
```

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	\$1,600.00	\$300.00	30
7521	WARD	SALESMAN	\$1,250.00	\$500.00	30
7698	BLAKE	MANAGER	\$2,850.00		30
7654	MARTIN	SALESMAN	\$1,250.00	\$1,400.00	30
7844	TURNER	SALESMAN	\$1,500.00	\$0.00	30
7900	JAMES	CLERK	\$950.00		30
7499	ALLEN	SALESMAN	\$1,600.00	\$300.00	30
7521	WARD	SALESMAN	\$1,250.00	\$500.00	30
7844	TURNER	SALESMAN	\$1,500.00	\$0.00	30

9 records selected.

2.13 NULL-Function

When an expression or built-in function references a column of a table that contains one or more null values, the result of the expression or built-in function is null.

Example 2-47: In the following example the expression SAL + COMM returns a null value for all employees that have a null commission.

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

ENAME	JOB	SAL	COMM	SAL+COMM
ALLEN	SALESMAN	\$1,600.00	\$300.00	\$1,900.00
WARD	SALESMAN	\$1,250.00	\$500.00	\$1,750.00
BLAKE	MANAGER	\$2,850.00		
MARTIN	SALESMAN	\$1,250.00	\$1,400.00	\$2,650.00
TURNER	SALESMAN	\$1,500.00	\$0.00	\$1,500.00
JAMES	CLERK	\$950.00		

6 records selected.

The ORACLE Null-Value Function NVL can be used to assign a temporary value to nulls encountered within an expression.

Example 2-48: Assign null commissions a temporary value of zero within the expression SAL + COMM.

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

ENAME	JOB	SAL	COMM	SAL+NVL(COMM,0)
ALLEN	SALESMAN	\$1,600.00	\$300.00	\$1,900.00
WARD	SALESMAN	\$1,250.00	\$500.00	\$1,750.00
BLAKE	MANAGER	\$2,850.00		\$2,850.00
MARTIN	SALESMAN	\$1,250.00	\$1,400.00	\$2,650.00
TURNER	SALESMAN	\$1,500.00	\$0.00	\$1,500.00
JAMES	CLERK	\$950.00		\$950.00

6 records selected.

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

Example 2-49: Null values do not participate in the computation of built-in functions.

```
SQL>SELECT  SUM(SAL),COUNT(SAL),SUM(COMM),COUNT(COMM)
SQL>FROM    EMP
SQL>WHERE    DEPTNO = 30
SQL>/
```

SUM(SAL)	COUNT(SAL)	SUM(COMM)	COUNT(COMM)
-----	-----	-----	-----
\$9,400.00	6	\$2,200.00	4

In the above example the count of people who receive a salary, (4), is greater than the number of people that receive a commission, (6), because null commissions were not counted.

Example 2-50: 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).

```
SQL>SELECT  AVG(COMM),AVG(NVL(COMM,0))
SQL>FROM    EMP
SQL>WHERE    DEPTNO = 30
SQL>/
```

AVG(COMM)	AVG(NVL(COMM,0))
-----	-----
\$550.00	\$366.67

Example 2-51: 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 \$1000 commission).

```
SQL>SELECT  AVG(COMM),AVG(NVL(COMM,1000))
SQL>FROM    EMP
SQL>WHERE    DEPTNO = 30
SQL>/
```

AVG(COMM)	AVG(NVL(COMM,1000))
-----	-----
\$550.00	\$700.00

Example 2-52: For department 30, list the average salary of employees that receive a salary, the average commission of employees that receive a commission, the average salary plus commission of only those employees that receive a commission, and average salary plus commission of all employees including those who do not receive a commission.

```
SQL>SELECT  AVG(SAL),AVG(COMM),AVG(SAL+COMM),AVG(SAL+NVL(COMM,0))
SQL>FROM    EMP
SQL>WHERE    DEPTNO = 30
SQL>/
```

AVG(SAL)	AVG(COMM)	AVG(SAL+COMM)	AVG(SAL+NVL(COMM,0))
-----	-----	-----	-----
\$1,566.67	\$550.00	\$1,950.00	\$1,933.33

2.14 Join Query

A query may return values from more than one table. The FROM clause may list several tables. The WHERE clause specifies the relationship on which the tables are to be joined.

Example 2-53: List the names of all employees and the locations of their departments.

```
SQL>SELECT  ENAME,LOC
SQL>FROM      EMP,DEPT
SQL>WHERE     EMP.DEPTNO = DEPT.DEPTNO;
SQL>/
```

ENAME	LOC
CLARK	NEW YORK
MILLER	NEW YORK
OATES	NEW YORK
SMITH	SAN FRANCISCO
JONES	SAN FRANCISCO
ADAMS	SAN FRANCISCO
SCOTT	SAN FRANCISCO
FORD	SAN FRANCISCO
ALLEN	CHICAGO
WARD	CHICAGO
BLAKE	CHICAGO
MARTIN	CHICAGO
TURNER	CHICAGO
JAMES	CHICAGO

14 records selected.

Example 2-54: List names of employees and all the fields of the department table for employees in departments located in Chicago.

```
SQL>SELECT  ENAME,DEPT.*
SQL>FROM      EMP,DEPT
SQL>WHERE     EMP.DEPTNO = DEPT.DEPTNO
SQL>  AND     LOC = 'CHICAGO';
SQL>/
```

ENAME	DEPTNO	DNAME	LOC	EMPCNT
ALLEN	30	SALES	CHICAGO	
WARD	30	SALES	CHICAGO	
BLAKE	30	SALES	CHICAGO	
MARTIN	30	SALES	CHICAGO	
TURNER	30	SALES	CHICAGO	
JAMES	30	SALES	CHICAGO	

6 records selected.

Predicates in a WHERE clause may compare data values in columns from any number of tables in a join query.

Example 2-55: List the names and jobs of employees who are not salesmen and work for departments that are located in Chicago.

```
SQL>SELECT  ENAME,JOB,LOC
SQL>FROM    DEPT,EMP
SQL>WHERE    EMP.DEPTNO=DEPT.DEPTNO
SQL>  AND    LOC = 'CHICAGO'
SQL>  AND    JOB ^= 'SALESMAN';
SQL>/
```

ENAME	JOB	LOC
BLAKE	MANAGER	CHICAGO
JAMES	CLERK	CHICAGO

Example 2-56: List the name, location, salary, job of employees located in Chicago who have the same job as Allen. Sort the results by employee name.

```
SQL>SELECT  ENAME,LOC,SAL,JOB
SQL>FROM    EMP,DEPT
SQL>WHERE    LOC = 'CHICAGO'
SQL>  AND    EMP.DEPTNO = DEPT.DEPTNO
SQL>  AND    JOB =
SQL>          SELECT  JOB
SQL>          FROM    EMP
SQL>          WHERE    ENAME = 'ALLEN';
SQL>ORDER BY ENAME;
SQL>/
```

ENAME	LOC	SAL	JOB
ALLEN	CHICAGO	\$1,600.00	SALESMAN
MARTIN	CHICAGO	\$1,250.00	SALESMAN
TURNER	CHICAGO	\$1,500.00	SALESMAN
WARD	CHICAGO	\$1,250.00	SALESMAN

2.15 Self-Join

A table may be joined with itself by listing the same table more than once in the FROM clause and associating a temporary label with each table. This label is used in place of the table name in qualifying references to columns within the SELECT and FROM clauses in the query block.

Example 2-57: For each employee whose salary exceeds his department's managers's salary, list the employee's name and salary and the manager's name and salary. Within the context of this query, the EMP table is treated logically as if it were two separate tables named WORKER and MGR. However, the EMP table is "not" physically duplicated.

```
SQL>SELECT  WORKER.ENAME,WORKER.SAL,MGR.ENAME,MGR.SAL
SQL>FROM    EMP WORKER,EMP MGR
SQL>WHERE   WORKER.DEPTNO = MGR.DEPTNO
SQL> AND    [MGR.JOB = 'MANAGER' AND WORKER.SAL > MGR.SAL];
SQL>/
```

ENAME	SAL	ENAME	SAL
SCOTT	\$3,000.00	JONES	\$2,975.00
FORD	\$3,000.00	JONES	\$2,975.00
OATES	\$5,000.00	CLARK	\$2,450.00

2.16 Outer-Join

When the DEPT table is joined to the EMP table using the join predicate DEPT.DEPTNO = EMP.DEPTNO, a department without any employees would not satisfy the join and would not be returned as a result of the query.

Example 2-58: List all the departments in the DEPT table.

```
SQL>SELECT *
SQL>FROM DEPT
SQL>/
```

DEPTNO	DNAME	LOC	EMPCNT
10	ADMINISTRATION	NEW YORK	
20	RESEARCH	SAN FRANCISCO	
30	SALES	CHICAGO	
40	OPERATIONS	BOSTON	

Example 2-59: List all the employees in the EMP table.

```
SQL>SELECT DEPTNO,ENAME,JOB
SQL>FROM EMP
SQL>ORDER BY DEPTNO
SQL>/
```

DEPTNO	ENAME	JOB
10	CLARK	MANAGER
10	OATES	PRESIDENT
10	MILLER	CLERK
20	SMITH	CLERK
20	JONES	MANAGER
20	SCOTT	ANALYST
20	ADAMS	CLERK
20	FORD	ANALYST
30	ALLEN	SALESMAN
30	WARD	SALESMAN
30	MARTIN	SALESMAN
30	BLAKE	MANAGER
30	TURNER	SALESMAN
30	JAMES	CLERK

14 records selected.

Example 2-60: Join the DEPT table to the EMP table.

```
SQL>SELECT  DEPT.DEPTNO,DNAME,LOC,ENAME,JOB
SQL>FROM    DEPT,EMP
SQL>WHERE    DEPT.DEPTNO = EMP.DEPTNO
SQL>/
```

DEPTNO	DNAME	LOC	ENAME	JOB
10	ADMINISTRATION	NEW YORK	CLARK	MANAGER
10	ADMINISTRATION	NEW YORK	MILLER	CLERK
10	ADMINISTRATION	NEW YORK	OATES	PRESIDENT
20	RESEARCH	SAN FRANCISCO	SMITH	CLERK
20	RESEARCH	SAN FRANCISCO	JONES	MANAGER
20	RESEARCH	SAN FRANCISCO	ADAMS	CLERK
20	RESEARCH	SAN FRANCISCO	SCOTT	ANALYST
20	RESEARCH	SAN FRANCISCO	FORD	ANALYST
30	SALES	CHICAGO	ALLEN	SALESMAN
30	SALES	CHICAGO	WARD	SALESMAN
30	SALES	CHICAGO	BLAKE	MANAGER
30	SALES	CHICAGO	MARTIN	SALESMAN
30	SALES	CHICAGO	TURNER	SALESMAN
30	SALES	CHICAGO	JAMES	CLERK

14 records selected.

The result of this join does not include department 40 because department 40 does not have any employees. An "outer-join", will return those department rows that have no matching employees.

Example 2-61: List all departments that have employees, plus those departments that do not have employees.

```
SQL>SELECT DEPT.DEPTNO,DNAME,LOC,ENAME,JOB
SQL>FROM DEPT,EMP*
SQL>WHERE DEPT.DEPTNO = EMP.DEPTNO
SQL>/
```

DEPTNO	DNAME	LOC	ENAME	JOB
10	ADMINISTRATION	NEW YORK	CLARK	MANAGER
10	ADMINISTRATION	NEW YORK	MILLER	CLERK
10	ADMINISTRATION	NEW YORK	OATES	PRESIDENT
20	RESEARCH	SAN FRANCISCO	SMITH	CLERK
20	RESEARCH	SAN FRANCISCO	JONES	MANAGER
20	RESEARCH	SAN FRANCISCO	ADAMS	CLERK
20	RESEARCH	SAN FRANCISCO	SCOTT	ANALYST
20	RESEARCH	SAN FRANCISCO	FORD	ANALYST
30	SALES	CHICAGO	ALLEN	SALESMAN
30	SALES	CHICAGO	WARD	SALESMAN
30	SALES	CHICAGO	BLAKE	MANAGER
30	SALES	CHICAGO	MARTIN	SALESMAN
30	SALES	CHICAGO	TURNER	SALESMAN
30	SALES	CHICAGO	JAMES	CLERK
40	OPERATIONS	BOSTON		

15 records selected.

The asterisk (*) after the EMP table in the FROM clause indicates that an extra row containing a null value in every column is to be appended to the EMP table when processing this query block. This null row of the EMP table is joined to those DEPT rows that do not have any matching rows in the EMP table.

Example 2-62: List all departments that do not have any employees.

```
SQL>SELECT UNIQUE DEPT.DEPTNO,DNAME,LOC
SQL>FROM DEPT,EMP*
SQL>WHERE DEPT.DEPTNO = EMP.DEPTNO
SQL> AND EMPNO = NULL
SQL>/
```

DEPTNO	DNAME	LOC
40	OPERATIONS	BOSTON

The outer-join can be used to join more than two tables, however, at least one table in the join must not be outer-joined. The table that is not outer-joined must be listed first in the FROM clause.

3. DATA MANIPULATION FACILITIES

3.1 INSERT

The INSERT statement specifies the adding of a new row or set of rows into a table.

Example 3-1: Insert a new employee named Carter with an employee number of 7989, a job title of salesman, salary of 1500, and commission of 0, into department 30.

```
SQL>INSERT INTO EMP(EMPNO,ENAME,JOB,SAL,COMM,DEPTNO):  
SQL>      <7989,'CARTER','SALESMAN',1500,0,30>;  
SQL>/  
1 record created.
```

All fields do not have to be included in the INSERT statement.

Example 3-2: Insert a new employee named Wilson, employee number 7955, in department 20, having all other fields null.

```
SQL>INSERT INTO EMP(EMPNO,ENAME,DEPTNO):  
SQL>      <7955,'WILSON',20>;  
SQL>/  
1 record created.
```

If all fields are present in the right order, the list of column names may be omitted.

Example 3-3: Insert a new employee into named Jakes into the EMP table.

```
SQL>INSERT INTO EMP:
SQL>      <7956,'JAKES','CLERK',1000,NULL,20>;
SQL>/
1 record created.
```

```
SQL>SELECT *
SQL>FROM EMP
SQL>WHERE DEPTNO = 20;
SQL>/
```

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
7369	SMITH	CLERK	\$800.00		20
7566	JONES	MANAGER	\$2,975.00		20
7876	ADAMS	CLERK	\$1,100.00		20
7788	SCOTT	ANALYST	\$3,000.00		20
7902	FORD	ANALYST	\$3,000.00		20
7955	WILSON				20
7956	JAKES	CLERK	\$1,000.00		20

7 records selected.

An INSERT statement may store the result of a query into an existing table.

Example 3-4: Add to the BONUS table all those employees whose commission is greater than 25% of their salary, or those employees who have the job title of president or manager.

```
SQL>INSERT INTO BONUS:
SQL>      SELECT  ENAME,JOB,SAL,COMM
SQL>      FROM    EMP
SQL>      WHERE    COMM > 0.25 * SAL
SQL>      OR      JOB IN ('PRESIDENT','MANAGER');
SQL>/
6 records created.
```

Example 3-5: List the BONUS table.

```
SQL>SELECT *
SQL>FROM    BONUS;
SQL>/
```

ENAME	JOB	SAL	COMM
BLAKE	MANAGER	\$2,850.00	
CLARK	MANAGER	\$2,450.00	
JONES	MANAGER	\$2,975.00	
MARTIN	SALESMAN	\$1,250.00	\$1,400.00
OATES	PRESIDENT	\$5,000.00	
WARD	SALESMAN	\$1,250.00	\$500.00

6 records selected.

3.2 UPDATE

Update is a process of changing the values of fields within the data base. The rows to be updated are specified by means of a WHERE clause. The updates to be made are specified in a SET clause.

Example 3-6: Set employee number 7782's salary to \$2,750.

```
SQL>UPDATE EMP
SQL>SET SAL = 2750
SQL>WHERE EMPNO = 7782;
SQL>/
1 record updated.
```

```
SQL>SELECT * FROM EMP WHERE EMPNO=7782;
SQL>/
```

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
7782	CLARK	MANAGER	\$2,750.00		10

A SET clause may update multiple fields within a table.

Example 3-7: Update department 30's location to Paris and increase its employee count by two.

```
SQL>UPDATE DEPT
SQL>SET LOC = 'PARIS',EMPCNT=NVL(EMPCNT,0) + 2
SQL>WHERE DEPTNO = 30;
SQL>/
1 record updated.
```

```
SQL>SELECT *
SQL>FROM DEPT;
SQL>/
```

DEPTNO	DNAME	LOC	EMPCNT
10	ADMINISTRATION	NEW YORK	
20	RESEARCH	SAN FRANCISCO	
30	SALES	PARIS	2
40	OPERATIONS	BOSTON	

Example 3-8: Update the employee table by giving a 15% raise to all employees whose names appear in the BONUS table.

```
SQL>UPDATE EMP
SQL>SET SAL = SAL * 1.15
SQL>WHERE ENAME IN
SQL>SELECT ENAME
SQL>FROM BONUS;
SQL>/
6 records updated.
```

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

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
7369	SMITH	CLERK	\$800.00		20
7499	ALLEN	SALESMAN	\$1,600.00	\$300.00	30
7521	WARD	SALESMAN	\$1,437.50	\$500.00	30
7566	JONES	MANAGER	\$3,421.25		20
7654	MARTIN	SALESMAN	\$1,437.50	\$1,400.00	30
7698	BLAKE	MANAGER	\$3,277.50		30
7782	CLARK	MANAGER	\$3,162.50		10
7788	SCOTT	ANALYST	\$3,000.00		20
7839	OATES	PRESIDENT	\$5,750.00		10
7844	TURNER	SALESMAN	\$1,500.00	\$0.00	30
7876	ADAMS	CLERK	\$1,100.00		20
7900	JAMES	CLERK	\$950.00		30
7902	FORD	ANALYST	\$3,000.00		20
7934	MILLER	CLERK	\$1,300.00		10
7955	WILSON				20
7956	JAKES	CLERK	\$1,000.00		20
7989	CARTER	SALESMAN	\$1,500.00	\$0.00	30

17 records selected.

3.3 DELETE

DELETE removes rows from tables in the data base.

Example 3-9: Delete the employee named Oates from the BONUS table.

```
SQL>DELETE BONUS
SQL>WHERE ENAME = 'OATES';
SQL>/
1 record deleted.
```

```
SQL>SELECT * FROM BONUS;
SQL>/
```

ENAME	JOB	SAL	COMM
BLAKE	MANAGER	\$2,850.00	
CLARK	MANAGER	\$2,450.00	
JONES	MANAGER	\$2,975.00	
MARTIN	SALESMAN	\$1,250.00	\$1,400.00
WARD	SALESMAN	\$1,250.00	\$500.00

The WHERE clause in a DELETE statement may contain a query block.

Example 3-10: Delete from the BONUS table all the employees with the same job as Jones.

```
SQL>DELETE BONUS
SQL>WHERE JOB IN
SQL>    SELECT JOB
SQL>    FROM EMP
SQL>    WHERE ENAME = 'JONES';
SQL>/
3 records deleted.
```

```
SQL>SELECT * FROM BONUS;
SQL>/
```

ENAME	JOB	SAL	COMM
MARTIN	SALESMAN	\$1,250.00	\$1,400.00
WARD	SALESMAN	\$1,250.00	\$500.00

A DELETE statement without a WHERE clause specifies the removal of all rows in the table.

Example 3-11: Delete all rows from the BONUS table.

```
SQL>DELETE BONUS;
```

```
SQL>/
```

```
2 records deleted.
```

```
SQL>SELECT * FROM BONUS;
```

```
SQL>/
```

ENAME	JOB	SAL	COMM

```
no records selected
```


4. DATA DEFINITION FACILITIES

This section of the manual describes the CREATE TABLE, EXPAND TABLE DEFINE VIEW, and DROP facilities of the system.

4.1 CREATE TABLE

Example 4-1: Display projects table.

```
SQL>SELECT *
SQL>FROM    PROJ;
SQL>/
FROM      PROJ;
      ^
invalid table name                                [*** ERROR ***]
```

Example 4-2: Create a new table to contain project number, name, and budget information.

```
SQL>CREATE TABLE PROJ
SQL>      PROJNO(NUMBER NONULL IMAGE UNIQUE),
SQL>      PNAME(CHAR(10) IMAGE),
SQL>      BUDGET(NUMBER),
SQL>      EMPCNT(NUMBER);
SQL>/
Table created.

SQL>SELECT *
SQL>FROM    PROJ;
SQL>/

PROJNO PNAME          BUDGET EMPCNT
-----
no records selected
```

Example 4-3: Insert three projects into the project table.

```
SQL>INSERT INTO PROJ(PROJNO,PNAME,BUDGET):
SQL>      <101,'ALPHA',250000>;
SQL>/
1 record created.
```

```
SQL>INSERT INTO PROJ(PROJNO,PNAME,BUDGET):
SQL>      <102,'BETA',175000>;
SQL>/
1 record created.
```

```
SQL>INSERT INTO PROJ(PROJNO,PNAME,BUDGET):
SQL>      <103,'GAMMA',95000>;
SQL>/
1 record created.
```

```
SQL>SELECT *
SQL>FROM    PROJ;
SQL>/
```

PROJNO	PNAME	BUDGET	EMPCNT
101	ALPHA	\$250,000.00	
102	BETA	\$175,000.00	
103	GAMMA	\$95,000.00	

4.2 DROP TABLE

Tables and views may be dropped dynamically.

Example 4-4: Drop the BONUS table from the data base.

```
SQL>DROP TABLE BONUS;
SQL>/
Table dropped.
```

4.3 EXPAND TABLE

Example 4-5: An existing table may be expanded by adding a new column to it.

```
SQL>SELECT EMPNO,DEPTNO,PROJNO,ENAME
SQL>FROM EMP
SQL>WHERE DEPTNO = 10;
SQL>/
SELECT EMPNO,DEPTNO,PROJNO,ENAME
invalid column name [*** ERROR ***]
```

Example 4-6: Add a new project number column to the employee table.

```
SQL>EXPAND TABLE EMP
SQL>ADD COLUMN PROJNO(NUMBER IMAGE);
SQL>/
Table expanded.
```

```
SQL>SELECT EMPNO,ENAME,PROJNO,DEPTNO
SQL>FROM EMP
SQL>WHERE DEPTNO = 10;
SQL>/
```

EMPNO	ENAME	PROJNO	DEPTNO
7782	CLARK		10
7934	MILLER		10
7839	OATES		10

Example 4-7: Update the employee table by assigning employees to projects.

```
SQL>UPDATE EMP
SQL>SET PROJNO = 101
SQL>WHERE DEPTNO = 20
SQL> OR JOB = 'MANAGER';
SQL>/
10 records updated.
```

```
SQL>SELECT *
SQL>FROM EMP
SQL>WHERE PROJNO = 101;
SQL>/
```

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO	PROJNO
7369	SMITH	CLERK	\$800.00		20	101
7566	JONES	MANAGER	\$3,421.25		20	101
7698	BLAKE	MANAGER	\$3,277.50		30	101
7782	CLARK	MANAGER	\$3,162.50		10	101
7876	ADAMS	CLERK	\$1,100.00		20	101
7788	SCOTT	ANALYST	\$3,000.00		20	101
7902	FORD	ANALYST	\$3,000.00		20	101
7955	WILSON				20	101
7956	JAKES	CLERK	\$1,000.00		20	101

9 records selected.

```
SQL>UPDATE EMP
SQL>SET PROJNO = 102
SQL>WHERE EMPNO > 7700
SQL> AND NOT PROJNO = 101;
SQL>/
5 records updated.
```

Example 4-8: The PROJ table may now be joined to the EMP table.

```
SQL>SELECT  ENAME,PNAME
SQL>FROM    EMP,PROJ
SQL>WHERE    EMP.PROJNO = PROJ.PROJNO;
SQL>/
```

ENAME	PNAME
-----	-----
SMITH	ALPHA
JONES	ALPHA
BLAKE	ALPHA
CLARK	ALPHA
ADAMS	ALPHA
SCOTT	ALPHA
FORD	ALPHA
WILSON	ALPHA
JAKES	ALPHA
TURNER	BETA
MILLER	BETA
OATES	BETA
JAMES	BETA
CARTER	BETA
ALLEN	GAMMA
WAFD	GAMMA
MARTIN	GAMMA

17 records selected.

4.4 DEFINE VIEW

Alternative views of stored data may be defined. Any valid query may be used in the definition of a view.

Example 4-9: Define a view called EMP10 containing employee number, name, job and project number information for employees in department 10.

```
SQL>DEFINE VIEW EMP10 AS
SQL>      SELECT EMPNO,ENAME,JOB,PROJNO
SQL>      FROM EMP
SQL>      WHERE DEPTNO = 10;
SQL>/
View defined.
```

```
SQL>SELECT *
SQL>FROM EMP10;
SQL>/
```

EMPNO	ENAME	JOB	PROJNO
-----	-----	-----	-----
7782	CLARK	MANAGER	101
7934	MILLER	CLERK	102
7839	OATES	PRESIDENT	102

Example 4-10: A view can contain more than one table.

```
SQL>DEFINE VIEW PROJSTAFF(EMPLOYEE,PROJECT) AS
SQL>      SELECT ENAME,PNAME
SQL>      FROM EMP,PROJ
SQL>      WHERE EMP.PROJNO = PROJ.PROJNO;
SQL>/
View defined.
```

```
SQL>SELECT *
SQL>FROM PROJSTAFF;
SQL>/
```

EMPLOYEE	PROJECT
SMITH	ALPHA
JONES	ALPHA
BLAKE	ALPHA
CLARK	ALPHA
ADAMS	ALPHA
SCOTT	ALPHA
FORD	ALPHA
WILSON	ALPHA
JAKES	ALPHA
TURNER	BETA
MILLER	BETA
OATES	BETA
JAMES	BETA
CARTER	BETA
ALLEN	GAMMA
WARD	GAMMA
MARTIN	GAMMA

17 records selected.

When defining a view, ORACLE may draw the column names from the underlying tables or the user may rename the columns as in the example below.

Example 4-11: Define a view containing information from the project, employee, and department table.

```
SQL>DEFINE VIEW PROJECTS (PROJECT,EMPLOYEE,EMP#,LOCATION) AS
SQL>      SELECT PNAME,ENAME,EMPNO,LOC
SQL>      FROM    PROJ,EMP,DEPT
SQL>      WHERE   EMP.DEPTNO = DEPT.DEPTNO
SQL>      AND     EMP.PROJNO = PROJ.PROJNO;
SQL>/
View defined.
```

Example 4-12: Views may be selectively queried in the same way as a table.

```
SQL>SELECT PROJECT,EMPLOYEE,LOCATION
SQL>FROM    PROJECTS
SQL>WHERE   LOCATION = 'NEW YORK';
SQL>/
```

PROJECT	EMPLOYEE	LOC
ALPHA	CLARK	NEW YORK
BETA	MILLER	NEW YORK
BETA	OATES	NEW YORK

Example 4-13: Views may be joined to tables or other views.

```
SQL>SELECT ENAME,JOB,PNAME
SQL>FROM    PROJ,EMP10
SQL>WHERE   PROJ.PROJNO=EMP10.PROJNO
SQL> AND    JOB ^= 'CLERK'
SQL>/
```

ENAME	JOB	PNAME
CLARK	MANAGER	ALPHA
OATES	PRESIDENT	BETA

Views may be defined in terms of other views.

Example 4-14: Define a view containing the name of projects and employees located in Paris.

```
SQL>DEFINE VIEW PARIS (NAME,PROJ) AS
SQL>      SELECT EMPLOYEE,PROJECT
SQL>      FROM    PROJECTS
SQL>      WHERE   LOCATION = 'PARIS';
SQL>/
View defined.
```

```
SQL>SELECT *
SQL>FROM    PARIS;
SQL>/
```

NAME	PROJ
ALLEN	GAMMA
WARD	GAMMA
BLAKE	ALPHA
MARTIN	GAMMA
TURNER	BETA
JAMES	BETA
CARTER	BETA

7 records selected.

4.5 Virtual-Fields

A view may contain arithmetic expressions or built in functions. These expressions or functions appear to the user of the view as "virtual fields." When expressions or functions are used within a view, column names must be specified for the view.

Example 4-15: Define a view containing employee name, salary, annual salary, and department number.

```
SQL>DEFINE VIEW PAY (NAME,SAL,COMM,ASAL,DEPTNO) AS
SQL>      SELECT ENAME,SAL,COMM,SAL * 12,DEPTNO
SQL>      FROM    EMP;
SQL>/
View defined.
```

Example 4-16: List salary information for employees in department 30.

```
SQL>SELECT *
SQL>FROM PAY
SQL>WHERE DEPTNO = 30
SQL>/
```

NAME	SAL	COMM	ASAL	DEPTNO
ALLEN	\$1,600.00	\$300.00	\$19,200.00	30
WARD	\$1,437.50	\$500.00	\$17,250.00	30
BLAKE	\$3,277.50		\$39,330.00	30
MARTIN	\$1,437.50	\$1,400.00	\$17,250.00	30
TURNER	\$1,500.00	\$0.00	\$18,000.00	30
JAMES	\$950.00		\$11,400.00	30
CARTER	\$1,500.00	\$0.00	\$18,000.00	30

7 records selected.

Example 4-17: Define a view containing a departments minimum, average, maximum, and total compensation.

```
SQL>DEFINE VIEW DEPT_SAL (DEPTNO,LOSAL,MEDSAL,HISAL,TOTSAL) AS
SQL>SELECT DEPTNO,MIN(SAL),AVG(SAL),MAX(SAL),SUM(SAL)
SQL>FROM EMP
SQL>GROUP BY DEPTNO;
SQL>/
View defined.
```

Example 4-18: List minimum, average, and total salary for each department.

```
SQL>SELECT DEPTNO,LOSAL,HISAL,TOTSAL
SQL>FROM DEPT_SAL
SQL>/
```

DEPTNO	LOSAL	HISAL	TOTSAL
10	\$1,300.00	\$5,750.00	\$10,212.50
20	\$800.00	\$3,421.25	\$12,321.25
30	\$950.00	\$3,277.50	\$11,702.50

5. DATA STRUCTURES

Create a data structure that allows one employee to work on many projects, and one project to have many employees.

Example 5-1: Create a two column table relating employees to projects.

```
SQL>CREATE TABLE PE
SQL>      EMPNO(NUMBER NONULL IMAGE),
SQL>      PROJNO(NUMBER NONULL IMAGE);
SQL>/
Table created.
```

Example 5-2: Move the relationship between employees and projects from the EMP table to the PE table.

```
SQL>INSERT INTO PE(EMPNO,PROJNO):
SQL>      SELECT EMPNO,PROJNO
SQL>      FROM EMP;
SQL>/
17 records created.
```

```
SQL>SELECT *
SQL>FROM PE;
SQL>/
```

EMPNO	PROJNO
7369	101
7499	103
7521	103
7566	101
7654	103
7698	101
7782	101
7788	101
7839	102
7844	102
7876	101
7900	102
7902	101
7934	102
7955	101
7956	101
7989	102

17 records selected.

Example 5-3: Delete the data from the PROJNO column of the EMP table.

```
SQL>UPDATE EMP
SQL>SET PROJNO = NULL;
SQL>/
17 records updated.
```

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

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO	PROJNO
7369	SMITH	CLERK	\$800.00		20	
7499	ALLEN	SALESMAN	\$1,600.00	\$300.00	30	
7521	WARD	SALESMAN	\$1,437.50	\$500.00	30	
7566	JONES	MANAGER	\$3,421.25		20	
7654	MARTIN	SALESMAN	\$1,437.50	\$1,400.00	30	
7698	BLAKE	MANAGER	\$3,277.50		30	
7782	CLARK	MANAGER	\$3,162.50		10	
7788	SCOTT	ANALYST	\$3,000.00		20	
7839	OATES	PRESIDENT	\$5,750.00		10	
7844	TURNER	SALESMAN	\$1,500.00	\$0.00	30	
7876	ADAMS	CLERK	\$1,100.00		20	
7900	JAMES	CLERK	\$950.00		30	
7902	FORD	ANALYST	\$3,000.00		20	
7934	MILLER	CLERK	\$1,300.00		10	
7955	WILSON				20	
7956	JAKES	CLERK	\$1,000.00		20	
7989	CARTER	SALESMAN	\$1,500.00	\$0.00	30	

17 records selected.

Example 5-4: The new structure requires that the EMP table be joined to the PROJ table via the PE table.

```
SQL>SELECT  ENAME,PNAME
SQL>FROM    EMP,PROJ,PE
SQL>WHERE   EMP.EMPNO = PE.EMPNO
SQL>  AND   PROJ.PROJNO = PE.PROJNO;
SQL>/
```

ENAME	PNAME
-----	-----
SMITH	ALPHA
ALLEN	GAMMA
WARD	GAMMA
JONES	ALPHA
MARTIN	GAMMA
BLAKE	ALPHA
CLARK	ALPHA
SCOTT	ALPHA
OATES	BETA
TURNER	BETA
ADAMS	ALPHA
JAMES	BETA
FORD	ALPHA
MILLER	BETA
WILSON	ALPHA
JAKES	ALPHA
CARTER	BETA

17 records selected.

The new data structure with the PE table allows employees to be assigned to more than one project.

Example 5-5: Assign employee 7989 to projects 101 and 103, and employee 7956 to project 102.

```
SQL>INSERT INTO PE:
SQL>      <7989,101>;
SQL>/
1 record created.
```

```
SQL>INSERT INTO PE:
SQL>      <7989,103>;
SQL>/
1 record created.
```

```
SQL>INSERT INTO PE:
SQL>      <7956,103>;
SQL>/
1 record created.
```

Example 5-6: List the projects for employees 7956 and 7989.

```
SQL>SELECT ENAME,EMP.EMPNO,PNAME,PROJ.PROJNO
SQL>FROM EMP,PROJ,PE
SQL>WHERE [EMP.EMPNO = PE.EMPNO AND PROJ.PROJNO = PE.PROJNO]
SQL> AND EMP.EMPNO = <7956,7989>;
SQL>/
```

ENAME	EMPNO	PNAME	PROJNO
JAKES	7956	ALPHA	101
JAKES	7956	GAMMA	103
CARTER	7989	BETA	102
CARTER	7989	ALPHA	101
CARTER	7989	GAMMA	103

6. DATA INDEPENDENCE

The view facility within ORACLE in combination with the non-procedural nature of the SQL data language allows the user's programs to be independent to changes in data structure.

A new PROJSTAFF view can be defined on the new data structure. The new view will contain the same information as the old PROJSTAFF view only the definition of the view will be different. Users of the old PROJSTAFF view will be insulated from the change.

Example 6-1: Drop the old PROJSTAFF view.

```
SQL>DROP    VIEW PROJSTAFF;
SQL>/
View dropped.
```

Example 6-2: Define a new PROJSTAFF view that joins the EMP table to the PROJ table via the PE table.

```
SQL>DEFINE VIEW PROJSTAFF(EMPLOYEE,PROJECT) AS
SQL>SELECT  ENAME,PNAME
SQL>FROM    EMP,PROJ,PE
SQL>WHERE   EMP.EMPNO = PE.EMPNO
SQL> AND    PROJ.PROJNO = PE.PROJNO;
SQL>/
View defined.
```

Example 6-3: Queries that used the old PROJSTAFF view continue to run without modification once the new PROJSTAFF view has been defined even though the structure of the data base has been altered.

```
SQL>SELECT *
SQL>FROM PROJSTAFF
SQL>WHERE PROJECT = 'GAMMA';
SQL>/
```

EMPLOYEE	PNAME
-----	-----
ALLEN	GAMMA
WARD	GAMMA
MARTIN	GAMMA
CARTER	GAMMA
JAKES	GAMMA

7. OPERATIONS ON TREE-STRUCTURED TABLES

The EMP table does not contain all the information necessary to define the "reporting structure" between the employees in the EMP table. This is because the EMP table does not identify each employee's direct supervisor. In order to store this reporting structure information, the EMP table has to be expanded.

Example 7-1: Add a new column called SUPR to the EMP table.

```
SQL>EXPAND TABLE EMP
SQL>      ADD COLUMN SUPR(NUMBER IMAGE);
SQL>/
Table expanded.
```

Example 7-2: Assign a supervisor to each employee except OATES.

```
SQL>UPDATE EMP
SQL>SET SUPR=7839
SQL>WHERE EMPNO=7782
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7839
SQL>WHERE EMPNO=7566
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7839
SQL>WHERE EMPNO=7698
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7782
SQL>WHERE EMPNO=7934
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7566
SQL>WHERE EMPNO=7788
SQL>/
1 record updated.
```

Continue assigning supervisors to employees.

```
SQL>UPDATE EMP
SQL>SET SUPR=7566
SQL>WHERE EMPNO=7902
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7566
SQL>WHERE EMPNO=7955
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7788
SQL>WHERE EMPNO=7876
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7902
SQL>WHERE EMPNO=7369
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7955
SQL>WHERE EMPNO=7956
SQL>/
1 record updated.
```

Continue assigning supervisors to employees.

```
SQL>UPDATE EMP
SQL>SET SUPR=7698
SQL>WHERE EMPNO=7499
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7698
SQL>WHERE EMPNO=7521
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7698
SQL>WHERE EMPNO=7654
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7698
SQL>WHERE EMPNO=7844
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7698
SQL>WHERE EMPNO=7900
SQL>/
1 record updated.
```

```
SQL>UPDATE EMP
SQL>SET SUPR=7698
SQL>WHERE EMPNO=7989
SQL>/
1 record updated.
```

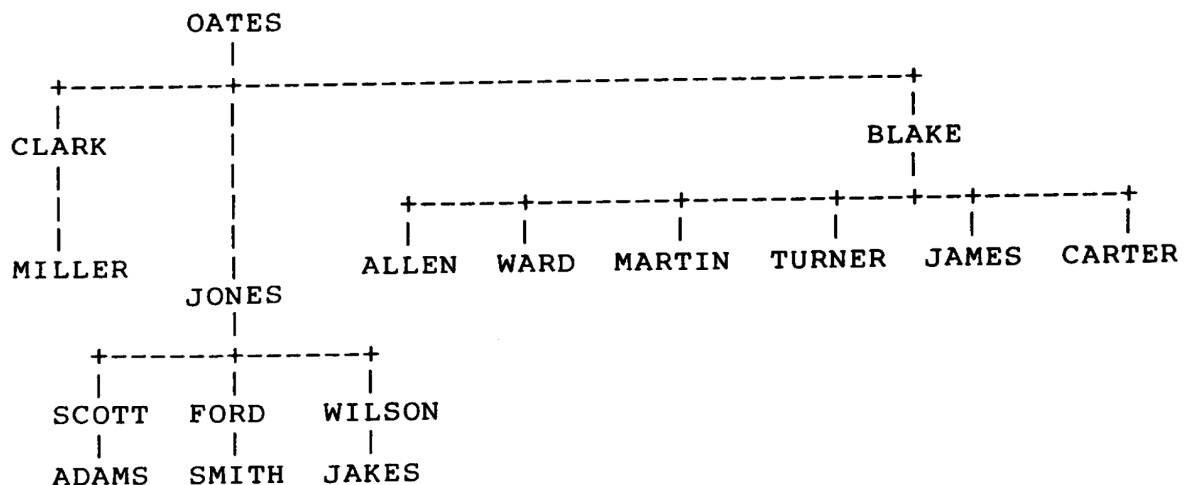
Example 7-3: List the reporting structure information from the EMP table including each employees name, number, department, and his supervisor's number.

```
SQL>SELECT  ENAME,EMPNO,SUPR,DEPTNO
SQL>FROM    EMP
SQL>/
```

ENAME	EMPNO	SUPR	DEPTNO
SMITH	7369	7902	20
ALLEN	7499	7698	30
WARD	7521	7698	30
JONES	7566	7839	20
MARTIN	7654	7698	30
BLAKE	7698	7839	30
CLARK	7782	7839	10
SCOTT	7788	7566	20
OATES	7839		10
TURNER	7844	7698	30
ADAMS	7876	7788	20
JAMES	7900	7698	30
FORD	7902	7566	20
MILLER	7934	7782	10
WILSON	7955	7566	20
JAKES	7956	7955	20
CARTER	7989	7698	30

17 records selected.

When considering the reporting structure information contained in the EMP table, it may be useful to think of the EMP table as tree-structured or hierarchical as in the diagram below.



7.1 CONNECT BY

ORACLE provides a unique set of operators that allows the user to query these "Tree-Structured" tables. Operations on tree-structured tables include three clauses: START WITH, CONNECT BY, and INCLUDING. CONNECT BY and START WITH are required clauses; INCLUDING is an optional clause.

The CONNECT BY clause indicates the two columns within the table that contain the information necessary to logically specify the structure of the tree. The START WITH clause indicates the leaf within the tree (row within the table) that the query is to start.

Example 7-4: Find all the people who work directly or indirectly for JONES.

```
SQL>SELECT UNIQUE ENAME,EMPNO,JOB,DEPTNO,SUPR
SQL>FROM EMP
SQL>START WITH ENAME = 'JONES'
SQL>CONNECT BY PRIOR EMPNO = SUPR
SQL>/
```

ENAME	EMPNO	JOB	DEPTNO	SUPR
JONES	7566	MANAGER	20	7839
SCOTT	7788	ANALYST	20	7566
FORD	7902	ANALYST	20	7566
WILSON	7955		20	7566
ADAMS	7876	CLERK	20	7788
SMITH	7369	CLERK	20	7902
JAKES	7956	CLERK	20	7955

7 records selected.

A query may "walk" the tree in either the UP or DOWN direction. The user specifies the direction the tree is to be walked by means of the PRIOR keyword within the CONNECT BY clause. In this example, if the PRIOR is placed before EMPNO, the tree is walked in the down direction. If the PRIOR is placed before SUPR, the tree is walked in the UP direction.

Example 7-5: List all the people in the reporting structure above SMITH.

```
SQL>SELECT UNIQUE ENAME,EMPNO,JOB,DEPTNO,SUPR
SQL>FROM EMP
SQL>START WITH ENAME = 'SMITH'
SQL>CONNECT BY EMPNO = PRIOR SUPR
SQL>/
```

ENAME	EMPNO	JOB	DEPTNO	SUPR
SMITH	7369	CLERK	20	7902
FORD	7902	ANALYST	20	7566
JONES	7566	MANAGER	20	7839
OATES	7839	PRESIDENT	10	

Example 7-6: SELECT UNIQUE must be specified when walking the tree in the UP direction.

7.2 START WITH

The START WITH clause can reference more than one starting point within the tree.

Example 7-7: List all the people who work for CLARK or BLAKE.

```
SQL>SELECT  UNIQUE ENAME,EMPNO,JOB,DEPTNO,SUPR
SQL>FROM      EMP
SQL>START    WITH ENAME = 'CLARK'
SQL> OR      ENAME = 'BLAKE'
SQL>CONNECT  BY PRIOR EMPNO = SUPR
SQL>ORDER   BY DEPTNO
SQL>/
```

ENAME	EMPNO	JOB	DEPTNO	SUPR
CLARK	7782	MANAGER	10	7839
MILLER	7934	CLERK	10	7782
BLAKE	7698	MANAGER	30	7839
ALLEN	7499	SALESMAN	30	7698
WARD	7521	SALESMAN	30	7698
MARTIN	7654	SALESMAN	30	7698
TURNER	7844	SALESMAN	30	7698
JAMES	7900	CLERK	30	7698
CARTER	7989	SALESMAN	30	7698

9 records selected.

Example 7-8: List all the people who work for people who have the same job as SCOTT.

```
SQL>SELECT  UNIQUE ENAME,EMPNO,JOB,DEPTNO,SUPR
SQL>FROM      EMP
SQL>START    WITH JOB IN
SQL>          SELECT  JOB
SQL>          FROM      EMP
SQL>          WHERE     ENAME = 'SCOTT';
SQL>CONNECT  BY PRIOR EMPNO = SUPR
SQL>/
```

ENAME	EMPNO	JOB	DEPTNO	SUPR
SCOTT	7788	ANALYST	20	7566
FORD	7902	ANALYST	20	7566
ADAMS	7876	CLERK	20	7788
SMITH	7369	CLERK	20	7902

7.3 INCLUDING

As a tree is being walked, a predicate or set of predicates can be applied to individual leafs of the tree or entire branches of the tree. The INCLUDING clause is used to qualify or disqualify leafs of the tree. The WHERE clause is used to "prune" entire branches of the tree.

Example 7-9: List all the people who work for JONES except SCOTT.

```
SQL>SELECT      UNIQUE ENAME,EMPNO,JOB,DEPTNO,SUPR
SQL>FROM        EMP
SQL>START       WITH ENAME = 'JONES'
SQL>CONNECT BY  PRIOR EMPNO = SUPR
SQL>INCLUDING   ENAME ^= 'SCOTT'
SQL>/
```

ENAME	EMPNO	JOB	DEPTNO	SUPR
JONES	7566	MANAGER	20	7839
FORD	7902	ANALYST	20	7566
WILSON	7955		20	7566
ADAMS	7876	CLERK	20	7788
SMITH	7369	CLERK	20	7902
JAKES	7956	CLERK	20	7955

6 records selected.

Example 7-10: List all the employees that work for JONES except SCOTT and the people who work for SCOTT.

```
SQL>SELECT UNIQUE ENAME,EMPNO,JOB,DEPTNO,SUPR
SQL>FROM EMP
SQL>WHERE ENAME ^= 'SCOTT'
SQL>START WITH ENAME = 'JONES'
SQL>CONNECT BY PRIOR EMPNO = SUPR
SQL>/
```

ENAME	EMPNO	JOB	DEPTNO	SUPR
JONES	7566	MANAGER	20	7839
FORD	7902	ANALYST	20	7566
WILSON	7955		20	7566
SMITH	7369	CLERK	20	7902
JAKES	7956	CLERK	20	7955

Note that ADAMS, who works for SCOTT was eliminated when SCOTT was pruned using the WHERE clause. ADAMS was not eliminated when SCOTT was excluded using the INCLUDING clause.

A query on a tree-structured table can contain both a WHERE clause and an INCLUDING clause. First, the tree is logically formed using the CONNECT BY clause; Second, the tree is walked in the direction specified by the PRIOR keyword in the CONNECT BY clause starting with the leaf specified in the START WITH clause; Third, the WHERE clause is applied to each leaf of the tree (row of the table) to prune branches from the tree; Fourth, the INCLUDING clause is applied to each leaf of the tree to qualify or disqualify individual rows.

Example 7-11: List all the employees that work for JONES except SCOTT and the people who work for SCOTT, and FORD.

```
SQL>SELECT UNIQUE ENAME,EMPNO,JOB,DEPTNO,SUPR
SQL>FROM EMP
SQL>WHERE ENAME ^= 'SCOTT'
SQL>START WITH ENAME = 'JONES'
SQL>CONNECT BY PRIOR EMPNO = SUPR
SQL>INCLUDING ENAME ^= 'FORD'
SQL>/
```

ENAME	EMPNO	JOB	DEPTNO	SUPR
JONES	7566	MANAGER	20	7839
WILSON	7955		20	7566
SMITH	7369	CLERK	20	7902
JAKES	7956	CLERK	20	7955

Queries on tree-structured tables can include joins.

Example 7-12: List all the employees and the location of their departments that work for JONES except SCOTT and the people who work for SCOTT, and FORD.

```
SQL>SELECT UNIQUE ENAME,LOC,EMPNO,JOB,DEPT.DEPTNO,SUPR
SQL>FROM EMP,DEPT
SQL>WHERE EMP.DEPTNO = DEPT.DEPTNO
SQL>AND ENAME ^= 'SCOTT'
SQL>START WITH ENAME = 'JONES'
SQL>CONNECT BY PRIOR EMPNO = SUPR
SQL>INCLUDING ENAME ^= 'FORD'
SQL>/
```

ENAME	LOC	EMPNO	JOB	DEPTNO	SUPR
JONES	SAN FRANCISCO	7566	MANAGER	20	7839
WILSON	SAN FRANCISCO	7955		20	7566
SMITH	SAN FRANCISCO	7369	CLERK	20	7902
JAKES	SAN FRANCISCO	7956	CLERK	20	7955

8. SECURITY FACILITIES

ORACLE supports both secure and nonsecure data bases. If a data base is defined as secure, that data base's dictionary contains information about the users of the data base in addition to a description of data stored within the data base. This allows ORACLE to control access to the data base on a user by access privilege basis.

An ORACLE data base is created by means of the DBF utility program. At the time a data base is created, the creating user specifies whether the data base is to be secure or nonsecure. If the data base is to be secure, the creating user specifies a USER-NAME and PASSWORD to DBF. The data base used in this manual is a secure data base created with the following DBF command:

```
DBF C PERSONNEL PERSONNEL.DBS 2048 SCOTT/TIGER
```

Initially, only the creating user, SCOTT with a PASSWORD of TIGER is authorized to operate on the PERSONNEL data base. All the SQL example operations prior to this section of the manual have been issued by the fully authorized creating user, SCOTT.

8.1 DEFINE USER

The creator of a secure data base can authorize additional users of the data base by means of the DEFINE USER command. A defined user of the data base is authorized to log on to the data base, create his own tables, define views on his tables, and define new users of the data base.

ORACLE security facilities prevent unauthorized users from logging on the secure PERSONNEL data base.

Have user SCOTT log off of the PERSONNEL data base and attempt to log on as user ADAMS with a password of WOOD.

Example 8-1: Log on to the PERSONNEL data base as user SCOTT.

```
SQL>#DBS PERSONNEL SCOTT/TIGER
Database 'PERSONNEL' opened.
```

The creator of the PERSONNEL data base, in this case SCOTT, can authorize additional users to log on to the data base by means of the DEFINE USER command.

Example 8-2: Define a new user of the PERSONNEL data base with a USER-NAME of ADAMS and a PASSWORD of WOOD.

```
SQL>DEFINE USER ADAMS/WOOD
SQL>/
User defined.
```

After a user has been defined, that user may log on to the data base.

Example 8-3: Log on as user ADAMS.

```
SQL>#DBS PERSONNEL ADAMS/WOOD
Database 'PERSONNEL' opened.
```

A new user can create his own tables, define views on his tables, and define new users of the data base.

Example 8-4: Have user ADAMS create a new table and insert a record into it.

```
SQL>CREATE TABLE PARTS
SQL>      PARTNO(NUMBER IMAGE UNIQUE),
SQL>      PART_NAME(CHAR(10));
SQL>/
Table created.
```

```
SQL>INSERT INTO PARTS:
SQL>      <1,'WIDGET'>;
SQL>/
1 record created.
```

8.2 GRANT PRIVILEGE

The ORACLE security facilities enable users to control access to their data by other users. It is the responsibility of the user who creates a table or view to control access to that table or view. A user may extend access to his table or view by means of the GRANT command.

Even though user ADAMS can create his own tables he can not access data via tables and views created by other users unless specifically authorized.

Example 8-5: Have user ADAMS attempt to list the DEPT table.

```
SQL>SELECT *
SQL>FROM DEPT
SQL>/
FROM DEPT
      ^
```

invalid table name

[*** ERROR ***]

At this point ADAMS is not authorized to access the DEPT table and is told that the table does not exist.

Example 8-6: Log back on as user SCOTT, the creator of the EMP and DEPT tables.

```
SQL>#DBS PERSONNEL SCOTT/TIGER
Database 'PERSONNEL' opened.
```

Example 8-7: Authorize user ADAMS to READ the DEPT table.

```
SQL>GRANT READ
SQL>ON DEPT
SQL>TO ADAMS
SQL>/
Privileges granted.
```

Example 8-8: Log on as user ADAMS.

```
SQL>#DBS PERSONNEL ADAMS/WOOD
Database 'PERSONNEL' opened.
```

Example 8-9: Have ADAMS list the DEPT table.

```
SQL>SELECT *
SQL>FROM DEPT
SQL>/
```

DEPTNO	DNAME	LOC	EMPCNT
10	ADMINISTRATION	NEW YORK	
20	RESEARCH	SAN FRANCISCO	
30	SALES	PARIS	2
40	OPERATIONS	BOSTON	

Example 8-10: Have ADAMS list the PARTS table he created.

```
SQL>SELECT *
SQL>FROM PARTS
SQL>/
```

PARTNO	PART_N
1	WIDGET

Example 8-11: Log on as user SCOTT.

```
SQL>#DBS PERSONNEL SCOTT/TIGER
Database 'PERSONNEL' opened.
```

Example 8-12: Have user SCOTT attempt to list the PARTS table.

```
SQL>SELECT *
SQL>FROM PARTS
SQL>/
FROM PARTS
^
```

invalid table name

[*** ERROR ***]

Even though user SCOTT defined user ADAMS to the PERSONNEL data base, SCOTT is not allowed to see data stored in tables created by ADAMS unless specifically authorized to do so by means of a GRANT privilege command.

8.3 PRIVILEGES

The following privileges may be granted: READ, INSERT, DELETE, UPDATE (by column), and EXPAND. In addition, the grantor may allow the grantee to GRANT the listed privileges to other users.

Example 8-13: Log back on as user SCOTT. #COMMENT #DBS PERSONNEL SCOTT/TIGER #COMMENT *** Give the following privileges on the EMP table to user ADAMS: the right to READ, INSERT, and UPDATE only the JOB, and DEPTNO columns.

```
SQL>GRANT    READ,INSERT,UPDATE(JOB,DEPTNO)
SQL>ON      EMP
SQL>TO      ADAMS
SQL>/
Privileges granted.
```

Example 8-14: Log on as ADAMS.

```
SQL>#DBS PERSONNEL ADAMS/WOOD
Database 'PERSONNEL' opened.
```

Example 8-15: Have ADAMS update the EMP table.

```
SQL>UPDATE EMP
SQL>SET      JOB='ANALYST'
SQL>WHERE    ENAME='WILSON'
SQL>/
1 record updated.
```

Example 8-16: Have ADAMS attempt to UPDATE both the JOB and SAL column of the EMP table.

```
SQL>UPDATE EMP
SQL>SET JOB='ANALYST',SAL=100000
SQL>WHERE ENAME='WILSON'
SQL>/
SET JOB='ANALYST',SAL=100000
```

security violation

[*** ERROR ***]

ORACLE security facilities detect a security violation and indicate the column of the EMP table that ADAMS was not authorized to UPDATE.

The phrase ALL RIGHTS may be substituted for the privilege list in the GRANT statement.

Example 8-17: Have ADAMS grant SCOTT all privileges on the PARTS table.

```
SQL>GRANT    ALL RIGHTS
SQL>ON       PARTS
SQL>TO       SCOTT
SQL>/
Privileges granted.
```

The phrase ALL BUT can be specified preceding a privilege list.

Example 8-18: Log on as user SCOTT.

```
SQL>#DBS PERSONNEL SCOTT/TIGER
Database 'PERSONNEL' opened.
```

Example 8-19: Give ADAMS all the privileges on the DEPT table except EXPAND, along with the right to GRANT these privileges to other users.

```
SQL>GRANT    ALL BUT EXPAND
SQL>ON       DEPT
SQL>TO       ADAMS
SQL>WITH GRANT OPTION
SQL>/
Privileges granted.
```

Privileges can be granted to all users by specifying PUBLIC in place of the user list.

Example 8-20: Give all users READ privileges on the PROJSTAFF view.

```
SQL>GRANT    READ
SQL>ON       PROJSTAFF
SQL>TO       PUBLIC
SQL>/
Privileges granted.
```

Note that the only privilege that may be granted on a view is the READ privilege.

8.4 USER Keyword

ORACLE requires users to enter their name and password in order to log on to a secure data base. ORACLE maintains the name of the current user as a keyword constant called USER. The keyword USER may be specified in a SQL statement anywhere a constant is allowed. USER will always return the name of the currently logged on user.

The USER keyword can be specified in the WHERE clause of a SQL statement to control access to the data base. This is especially useful when defining views on the data base.
#WORKSIZE 8

Example 8-21: Define a view of the EMP table allowing any employee to see his own department number, name, salary, commission, and job but not any information about any other employee.

```
SQL>DEFINE VIEW MYSELF AS
SQL>      SELECT *
SQL>      FROM EMP
SQL>      WHERE ENAME = USER;
SQL>/
View defined.
```

Example 8-22: Give all users access to the MYSELF view.

```
SQL>GRANT READ
SQL>ON MYSELF
SQL>TO PUBLIC
SQL>/
Privileges granted.
```

The USER keyword will always return the USER-NAME of the user currently logged on to the data base.

Example 8-23: Have user SCOTT query the MYSELF view.

```
SQL>SELECT *
SQL>FROM MYSELF
SQL>/
```

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO	PROJNO	SUPR
7788	SCOTT	ANALYST	\$3,000.00		20		7566

Example 8-24: Log on as user ADAMS and query the MYSELF view.

```
SQL>#DBS PERSONNEL ADAMS/WOOD
Database 'PERSONNEL' opened.
SQL>SELECT *
SQL>FROM MYSELF
SQL>/
```

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO	PROJNO	SUPR
7876	ADAMS	CLERK	\$1,100.00		20		7788

Example 8-25: Define a view giving managers access to employees in their department only if the manager earns the same amount or more than the employee.

```
SQL>DEFINE VIEW MY EMPS (NAME,JOB,SAL,COMM,DEPTNO) AS
SQL>SELECT WORKER.ENAME,WORKER.JOB,WORKER.SAL,WORKER.COMM,
SQL>SELECT WORKER.DEPTNO
SQL>FROM EMP WORKER,EMP MGR
SQL>WHERE WORKER.DEPTNO = MGR.DEPTNO
SQL> AND [MGR.JOB = 'MANAGER' AND MGR.SAL >= WORKER.SAL]
SQL> AND MGR.ENAME=USER;
SQL>/
```

```
SELECT WORKER.DEPTNO
      ^
```

missing 'from' keyword

[*** ERROR ***]

Example 8-26: Define JONES and CLARK as users of the PERSONNEL data base.

```
SQL>DEFINE USER JONES/WOOD
SQL>/
User defined.
```

```
SQL>DEFINE USER CLARK/CLOTH
SQL>/
User defined.
```

Example 8-27: Grant all users access to the MY_EMPS view. Note that the view will only return data if the user's JOB = MANAGER.

```
SQL>GRANT READ
SQL>ON MY EMPS
SQL>TO PUBLIC
SQL>/
TO PUBLIC
      ^
```

security violation

[*** ERROR ***]

Example 8-28: Have user ADAMS, who is not a manager attempt to query the MY_EMPS view.

```
SQL>SELECT *
SQL>FROM MY_EMPS
SQL>/
FROM MY_EMPS
      ^
invalid table name                                     [*** ERROR ***]
```

Example 8-29: Have users JONES and CLARK, who are managers, query the MY_EMPS view.

```
SQL>#DBS PERSONNEL JONES/WOOD
Database 'PERSONNEL' opened.
SQL>SELECT *
SQL>FROM MY_EMPS
SQL>/
FROM MY_EMPS
      ^
invalid table name                                     [*** ERROR ***]
```

Example 8-30: The view only allows the manager to see people in their own department.

```
SQL>#DBS PERSONNEL CLARK/CLOTH
Database 'PERSONNEL' opened.
SQL>SELECT *
SQL>FROM MY_EMPS
SQL>/
FROM MY_EMPS
      ^
invalid table name                                     [*** ERROR ***]
```

Note that CLARK was not able to see employee OATES because the view prevents managers from seeing anyone in their department who earns more than they do.

8.5 REVOKE Privilege

Once a privilege has been granted it may be withdrawn by means of the REVOKE command. Privileges are revoked from the named grantee and from all users to whom he has granted them.

Example 8-31: Log on the PERSONNEL data base as user ADAMS.

```
SQL>#DBS PERSONNEL ADAMS/WOOD
Database 'PERSONNEL' opened.
```

Example 8-32: Have ADAMS INSERT a new department into the DEPT table.

```
SQL>INSERT INTO DEPT:
SQL>      <50,'SERVICE','DETROIT',NULL>
SQL>/
1 record created.
```

Example 8-33: Log on the PERSONNEL data base as user SCOTT.

```
SQL>#DBS PERSONNEL SCOTT/TIGER
Database 'PERSONNEL' opened.
```

Example 8-34: Revoke from ADAMS the right to INSERT into the DEPT table.

```
SQL>REVOKE INSERT
SQL>ON      DEPT
SQL>FROM    ADAMS
SQL>/
Privileges revoked.
```

Example 8-35: Log on the PERSONNEL data base as user ADAMS.

```
SQL>#DBS PERSONNEL ADAMS/WOOD
Database 'PERSONNEL' opened.
```

Example 8-36: Have ADAMS attempt to INSERT a new department into the DEPT table.

```
SQL>INSERT INTO DEPT:
SQL>      <60,'DEVELOPMENT','PORTLAND',NULL>
SQL>/
INSERT INTO DEPT:
invalid table name
```

[*** ERROR ***]

9. DATA DICTIONARY STRUCTURE

This section of the manual contains a description of ORACLE's integrated data dictionary.

The ORACLE data dictionary is made up of several system defined tables and views. These dictionary tables are dynamically updated by ORACLE to contain a current description of all user tables, views, and access privileges. In addition, the ORACLE dictionary is self describing. Therefore, a user may query the dictionary to determine the names of the tables that make up the dictionary.

DTAB contains the names and a description of the dictionary tables. DCOL contains the names of the columns of the dictionary tables.

Example 9-1: List the names and a description of the tables in the dictionary.

```
SQL>SELECT *
SQL>FROM DTAB
SQL>/
```

TABLE	COMMENT
COL	COLUMN NAMES OF USERS TABLES AND VIEWS
COLDEF	DEFINITION OF COLUMNS IN USERS TABLES
DCOL	COLUMN NAMES OF DICTIONARY TABLES
DTAB	COMMENTS ON DICTIONARY TABLES
DTABLES	DESCRIPTION OF DICTIONARY TABLES
EXPDEF	COLUMN DEFINITIONS USED BY EXPORT
GRANTS	ACCESS PRIVILEGES GRANTED BY USER
PRIVS	ACCESS PRIVILEGES HELD BY USER
TAB	NAMES OF USERS TABLES AND VIEWS
USERS	NAME OF USERS YOU DEFINED
VIEWS	DEFINITIONS OF VIEWS
VXREF	CROSS-REFERENCE OF VIEWS OF TABLES

12 records selected.

Example 9-2: List the names of the columns of the dictionary tables.

```
SQL>SELECT *
SQL>FROM DCOL
SQL>/
```

TABLE	COLUMN
-----	-----
TAB	TABLE
TAB	TYPE
TAB	CREATOR
TAB	GRANTEE
COL	TABLE
COL	COLUMN
COLDEF	TABLE
COLDEF	COLUMN
COLDEF	DATATYPE
COLDEF	LENGTH
COLDEF	IMAGE
COLDEF	NONULL
EXPDEF	TABLE
EXPDEF	COLID
EXPDEF	COLUMN
EXPDEF	DATATYPE
EXPDEF	LENGTH
EXPDEF	IMAGE
EXPDEF	NONULL
VIEWS	VIEW
VIEWS	TEXT
VXREF	VIEW
VXREF	TABLE
USERS	USER
USERS	OWNER
DTAB	TABLE
DTAB	COMMENT
DTABLES	TABLE
DTABLES	TYPE
DTABLES	CREATOR
DTABLES	GRANTEE
DCOL	TABLE
DCOL	COLUMN
GRANTS	TABLE
GRANTS	COLUMN
GRANTS	GRANTEE
GRANTS	ACCESS
PRIVS	TABLE
PRIVS	COLUMN
PRIVS	GRANTOR
PRIVS	ACCESS

41 records selected.

The dictionary table TAB allows a user to list the names of all the tables and views that that user has access privileges on. TAB also indicates if the table is a view, the name of the user who created the table (or view), and the grantee of the privileges.

Example 9-3: List user ADAM's tables.

```
SQL>SELECT *
SQL>FROM TAB
SQL>/
```

TABLE	TYPE	CREATOR	GRANTEE
PARTS	TABLE	ADAMS	ADAMS
DEPT	TABLE	SCOTT	ADAMS
EMP	TABLE	SCOTT	ADAMS
PROJSTAFF	VIEW	SCOTT	PUBLIC
MYSELF	VIEW	SCOTT	PUBLIC

Example 9-4: Log on as user SCOTT.

```
SQL>#DBS PERSONNEL SCOTT/TIGER
Database 'PERSONNEL' opened.
```


Example 9-5: List user SCOTT's table.

```
SQL>SELECT *
SQL>FROM TAB
SQL>/
```

TABLE	TYPE	CREATOR	GRANTEE
EMP	TABLE	SCOTT	SCOTT
DEPT	TABLE	SCOTT	SCOTT
PROJ	TABLE	SCOTT	SCOTT
EMP10	VIEW	SCOTT	SCOTT
PROJECTS	VIEW	SCOTT	SCOTT
PARIS	VIEW	SCOTT	SCOTT
PAY	VIEW	SCOTT	SCOTT
DEPT_SAL	VIEW	SCOTT	SCOTT
PE	TABLE	SCOTT	SCOTT
PROJSTAFF	VIEW	SCOTT	SCOTT
PARTS	TABLE	ADAMS	SCOTT
MYSELF	VIEW	SCOTT	SCOTT
PROJSTAFF	VIEW	SCOTT	PUBLIC
MYSELF	VIEW	SCOTT	PUBLIC

14 records selected.

Example 9-6: The dictionary table COL contains the names of the columns of user defined tables.

```
SQL>#DBS PERSONNEL ADAMS/WOOD
Database 'PERSONNEL' opened.
```

Example 9-7: List the names of the columns of user ADAMS' tables.

```
SQL>SELECT *
SQL>FROM COL
SQL>/
```

TABLE	COLUMN
-----	-----
PARTS	PARTNO
PARTS	PART NAME
DEPT	DEPTNO
DEPT	DNAME
DEPT	LOC
DEPT	EMPCNT
EMP	EMPNO
EMP	ENAME
EMP	JOB
EMP	SAL
EMP	COMM
EMP	DEPTNO
EMP	PROJNO
EMP	SUPR
PROJSTAFF	EMPLOYEE
PROJSTAFF	PROJECT
MYSELF	EMPNO
MYSELF	ENAME
MYSELF	JOB
MYSELF	SAL
MYSELF	COMM
MYSELF	DEPTNO
MYSELF	PROJNO
MYSELF	SUPR

24 records selected.

Example 9-8: Log on as user SCOTT.

```
SQL>#DBS PERSONNEL SCOTT/TIGER
Database 'PERSONNEL' opened.
```

The dictionary table COLDEF contains the definition of columns in user tables.

Example 9-9: List the definition of the columns in the EMP table.

```
SQL>SELECT COLUMN,DATATYPE,LENGTH,IMAGE,NONULL
SQL>FROM COLDEF
SQL>WHERE TABLE = 'EMP'
SQL>/
```

COLUMN	DATATYPE	LENGTH	IMAGE	NON
EMPNO	NUMBER	22	UNIQUE	YES
ENAME	CHAR	10	NON-UNIQUE	NO
JOB	CHAR	9	NON-UNIQUE	NO
SAL	NUMBER	22		NO
COMM	NUMBER	22		NO
DEPTNO	NUMBER	22	NON-UNIQUE	NO
PROJNO	NUMBER	22	NON-UNIQUE	NO
SUPR	NUMBER	22	NON-UNIQUE	NO

8 records selected.

VIEWS is the dictionary table that contains the SQL text of the DEFINE VIEW statement. Comments that were entered as a part of the SQL view definition are also stored in VIEWS.

Example 9-10: List the definition of the PROJECTS view.

```
SQL>SELECT *
SQL>FROM VIEWS
SQL>WHERE VIEW = 'PROJECTS';
SQL>/
```

VIEW	TEXT
PROJECTS	DEFINE VIEW PROJECTS (PROJECT,EMPLOYEE,EMP#,LOC
PROJECTS	ATION) AS
PROJECTS	SELECT PNAME,ENAME,EMPNO,LOC
PROJECTS	FROM PROJ,EMP,DEPT
PROJECTS	WHERE EMP.DEPTNO = DEPT.DEPTNO
PROJECTS	AND EMP.PROJNO = PROJ.PROJNO;

6 records selected.

The VXREF dictionary table defines the relationship of user views to underlying tables and views. The first column contains the name of the view. The second column contains the name of the underlying table or view.

Example 9-11: List a cross reference of views and their base tables.

```
SQL>SELECT *
SQL>FROM   VXREF
SQL>/
```

VIEW	TABLE
-----	-----
EMP10	EMP
PROJECTS	EMP
PAY	EMP
DEPT SAL	EMP
PROJSTAFF	EMP
MYSELF	EMP
PROJECTS	DEPT
PROJECTS	PROJ
PROJSTAFF	PROJ
PARIS	PROJECTS
PROJSTAFF	PE

11 records selected.

The ORACLE dictionary allows users to obtain information about only those tables that the user has access privileges on. The user may also determine: the names of users he has directly or indirectly defined, the access privileges he holds, and the access privileges he has directly or indirectly granted.

The USERS table contains the names of all those users that were defined by the user who is currently logged on to the system. The first column in the USERS table contains the name of the defined user. The second column of the USERS table contains the name of the creating user.

Example 9-12: List the names of the users that originated with SCOTT.

```
SQL>SELECT  *
SQL>FROM    USERS
SQL>/
```

```
USER  OWNER
-----
ADAMS SCOTT
JONES ADAMS
CLARK ADAMS
```

User ADAMS was defined directly by user SCOTT, but users JONES and CLARK were defined by ADAMS and thereby indirectly by SCOTT.

Example 9-13: Log on as user ADAMS.

```
SQL>#DBS PERSONNEL ADAMS/WOOD
Database 'PERSONNEL' opened.
```

Example 9-14: List the users that originated with ADAMS.

```
SQL>SELECT  *
SQL>FROM    USERS
SQL>/
```

```
USER  OWNER
-----
JONES ADAMS
CLARK ADAMS
```

The PRIVS table contains access privileges held by users of the data base. Each user can only see the privileges he holds. The PRIVS table contains: the name of the table the privileges are on, the name of the column the privileges apply to, the name user who granted the privileges, and the privilege mask described below. A PRIVS table entry will only contain a column name if update privileges have been granted on a column column rather than a table basis.

Example 9-15: List all the privileges held by ADAMS.

```
SQL>SELECT *
SQL>FROM PRIVS
SQL>/
```

TABLE	COLUMN	GRANTOR	ACCESS
PARTS		ADAMS	RGIGDGUGEGMGLGCG
DEPT		SCOTT	RGXXDGUGXXMGLGCG
EMP		SCOTT	RXXXXXXXXXXXXXXXXX
EMP	JOB	SCOTT	XXXXXXXXXXXXXXXXXX
EMP	DEPTNO	SCOTT	XXXXXXXXXXXXXXXXXX
TAB		ORACLE	RXXXXXXXXXXXXXXXXX
COL		ORACLE	RXXXXXXXXXXXXXXXXX
COLDEF		ORACLE	RXXXXXXXXXXXXXXXXX
EXPDEF		ORACLE	RXXXXXXXXXXXXXXXXX
VIEWS		ORACLE	RXXXXXXXXXXXXXXXXX
VXREF		ORACLE	RXXXXXXXXXXXXXXXXX
USERS		ORACLE	RXXXXXXXXXXXXXXXXX
DTAB		ORACLE	RXXXXXXXXXXXXXXXXX
DTABLES		ORACLE	RXXXXXXXXXXXXXXXXX
DCOL		ORACLE	RXXXXXXXXXXXXXXXXX
GRANTS		ORACLE	RXXXXXXXXXXXXXXXXX
PRIVS		ORACLE	RXXXXXXXXXXXXXXXXX
PROJSTAFF		SCOTT	RXXXXXXXXXXXXXXXXX
MYSELF		SCOTT	RXXXXXXXXXXXXXXXXX

19 records selected.

Note that ADAMS has all privileges on MY_EMPS and PARTS, the table and view he created. He has all but INSERT and EXPAND on the DEPT table. He has READ and INSERT without the GRANT option, and UPDATE on the JOB and DEPTNO columns of the EMP table. The diagram below describes the entries in the ACCESS column of both the PRIVS and GRANTS dictionary tables.

PRIVILEGE MASK															
R	G	I	G	D	G	U	G	E	G	M	G	L	G	C	G
															not used
															+-----
															not used
															+-----
															not used
															+-----
															EXPAND
															+-----
															UPDATE
															+-----
															DELETE
															+-----
															INSERT
															+-----
															READ
															+-----

The G following each privilege flag indicates that the GRANT OPTION is present for that privilege. If an X is present for any privilege or grant flag, that privilege was not granted.

The GRANTS table indicates the privileges granted either directly or indirectly by the current user. The GRANTS table contains: the name of the table the privileges are on, the name of the column the privileges apply to, the name of the user who is the grantee of the privileges, and the privilege mask. A GRANTS table entry will only contain a column name if update privileges have been granted on a column rather than a table basis.

Example 9-16: List all the privileges granted by ADAMS.

```
SQL>SELECT *
SQL>FROM GRANTS
SQL>/
```

TABLE	COLUMN	GRANTEE	ACCESS
PARTS		ADAMS	RGIGDGUGEGMGLGCG
PARTS		SCOTT	RXIXDXUXEXMXLXCX

9.1 Dictionary Extensions

The user is free to expand the ORACLE data dictionary by creating additional tables, and defining views of the user tables joined to the system defined dictionary tables and views.

Example 9-17: Create a dictionary extension to contain comments on user defined tables.

```
SQL>CREATE TABLE TAB_COM
SQL>      TABLE(CHAR(20) IMAGE UNIQUE),
SQL>      COMMENT(CHAR(35))
SQL>/
Table created.
```

Example 9-18: Enter a comment on the EMP table into the TAB_COM table.

```
SQL>INSERT INTO TAB_COM:
SQL>      <'EMP','Information about company employees'>
SQL>/
1 record created.
```

Example 9-19: Define a view joining TAB from the dictionary to the TAB_COM table.

```
SQL>DEFINE VIEW TABC AS
SQL>      SELECT TAB.TABLE,TYPE,CREATOR,COMMENT
SQL>      FROM TAB,TAB_COM
SQL>      WHERE TAB.TABLE=TAB_COM.TABLE;
SQL>/
View defined.
```

```
SQL>#WORKSIZE 12
```

Example 9-20: List the characteristics of the EMP table.

```
SQL>SELECT *
SQL>FROM TABC
SQL>WHERE TABLE = 'EMP'
SQL>/
```

```
-----
TYPE  CREATOR COMMENT
-----
```

```
EMP
TABLE SCOTT  Information about company employees
```

10. Concurrency Control Facilities

ORACLE allows multiple user to concurrently UPDATE the same table in a data base. Yet, ORACLE does not require a user to issue locking statements or statements of intent to update. The setting and clearing of locks are the responsibility of ORACLE.

No explicit lock requests are required to insure that concurrent UPDATE operations do not read the same row of a table and attempt to write back that row. ORACLE automatically places locks on individual records to synchronize UPDATE operations preventing the updates from overwriting each other.

A user may not wish to operate on data that has been modified by an operation that is still in progress. To accomplish this serialization the user can place his SQL request between BEGIN TRANSACTION and END TRANSACTION statements.

In addition, placing several SQL statements inside a transaction causes ORACLE to execute these SQL statements as an "atomic act" without permitting interference (UPDATES to the same tables) by other users during the transaction.

ORACLE's automatic update synchronization locks a record (row of a table) at a time. USER requested transactions lock a table at a time.

Example 10-1: Log on as user SCOTT

```
SQL>#DBS PERSONNEL SCOTT/TIGER
Database 'PERSONNEL' opened.
```

Example 10-2: Begin a transaction to calculate the average salaries of each job group within the EMP table.

```
SQL>BEGIN TRANSACTION
SQL>ON TABLE EMP READ
SQL>/
Transaction begun.
```

```
SQL>SELECT JOB,AVG(SAL)
SQL>FROM EMP
SQL>GROUP BY JOB
SQL>/
```

JOB	AVG(SAL)
ANALYST	\$3,000.00
CLERK	\$1,030.00
MANAGER	\$3,287.08
PRESIDENT	\$5,750.00
SALESMAN	\$1,495.00

```
SQL>END TRANSACTION
SQL>/
Transaction ended.
```

In the above READ TRANSACTION, all UPDATE activity on the EMP table was suspended for the life of the transaction. All READ activity on the EMP table was allowed to continue concurrently.

Example 10-3: Execute an UPDATE TRANSACTION that gives all employees with the job of ANALYST a 10% raise.

```
SQL>BEGIN TRANSACTION
SQL>ON TABLE EMP UPDATE
SQL>/
Transaction begun.
```

```
SQL>UPDATE EMP
SQL>SET SAL = SAL * 1.10
SQL>WHERE JOB = 'ANALYST'
SQL>/
3 records updated.
```

```
SQL>END TRANSACTION
SQL>/
Transaction ended.
```

In the above UPDATE TRANSACTION, all UPDATE activity on the EMP table was suspended for the life of the transaction. In addition, all READ transactions on the EMP table also waited for the UPDATE transaction to complete. READ activity on the EMP table that was not a part of a TRANSACTION was allowed to continue concurrently.

Placing several SQL statements inside a transaction causes ORACLE to execute these SQL statements as a "atomic act" without permitting interference (UPDATES to the same tables) by other users during the transaction.

Example 10-4: SELECT the count of employees in department 20 from the EMP table, and UPDATE the DEPT table with the result.

```
SQL>BEGIN TRANSACTION
SQL>ON TABLE EMP READ, DEPT UPDATE
SQL>/
Transaction begun.
```

```
SQL>SELECT  COUNT(*)
SQL>FROM    EMP
SQL>WHERE   DEPTNO = 20
SQL>/
```

```
COUNT(*)
-----
       7
```

```
SQL>UPDATE  DEPT
SQL>SET     EMPCNT = 7
SQL>WHERE   DEPTNO = 20
SQL>/
1 record updated.
```

```
SQL>END TRANSACTION
SQL>/
Transaction ended.
```

If the above transaction had been done from within a program it could have used from one to three cursors: one for the BEGIN TRANSACTION, one for the query on the EMP table, and one for the UPDATE of the DEPT table.

TRANSACTIONS may be nested. If they are nested, they must be numbered.

Example 10-5: SELECT the count of employees in department 20 from the EMP table and UPDATE the DEPT table with the result as a nested transaction rather than a single transaction.

```
SQL>BEGIN TRANSACTION 1
SQL>ON TABLE EMP READ
SQL>/
Transaction begun.
```

```
SQL>SELECT COUNT(*)
SQL>FROM EMP
SQL>WHERE DEPTNO = 20
SQL>/
```

```
COUNT(*)
-----
          7
```

```
SQL>BEGIN TRANSACTION 2
SQL>ON TABLE DEPT UPDATE
SQL>/
Transaction begun.
```

```
SQL>UPDATE DEPT
SQL>SET EMPCNT = 7
SQL>WHERE DEPTNO = 20
SQL>/
1 record updated.
```

```
SQL>END TRANSACTION 2
SQL>/
Transaction ended.
```

```
SQL>END TRANSACTION 1
SQL>/
Transaction ended.
```

In doing the above operation as a nested transaction rather than a single transaction, the DEPT table was not locked during the query on the EMP table. This allowed update activity to continue on the DEPT table while the EMP table was being queried. This could be significant depending on the length of time it takes the query to execute. However, the nesting of transactions creates the possibility that TRANSACTION 2 will have to wait for access to the DEPT table which may be locked by another ongoing transaction. The nesting of transactions creates the possibility of deadlock. Doing the entire operation as one transaction has no potential for deadlock because once a transaction has begun, it has acquired all the data resources required for completion.

It is important to remember that the above query/update operation could have been done three different ways: 1) without placing them within a transaction; 2) as a single transaction; 3) as a nested transaction. It is up to the user to decide what level of serialization of operations he requires for a particular application.

The following chart indicates what operations on a table wait, and what operations continue when a READ, READ TRANSACTION, UPDATE OR UPDATE TRANSACTION is active on that table.

CONCURRENCY CONTROL TABLE				
	READ	READ TRANS	UPDATE	UPDATE TRANS
READ	ALLOWED	ALLOWED	ALLOWED	ALLOWED
READ TRANS	ALLOWED	ALLOWED	SUSPENDED	SUSPENDED
UPDATE	ALLOWED	SUSPENDED	ALLOWED	SUSPENDED
UPDATE TRANS	ALLOWED	SUSPENDED	SUSPENDED	SUSPENDED

