
A

Practice Solutions

Practice 1 Solutions

1. Initiate an iSQL*Plus session using the user ID and password provided by the instructor.
2. iSQL*Plus commands access the database.

False

3. The following SELECT statement executes successfully:

True

```
SELECT last_name, job_id, salary AS Sal
FROM employees;
```

4. The following SELECT statement executes successfully:

True

```
SELECT *
FROM job_grades;
```

5. There are four coding errors in this statement. Can you identify them?

```
SELECT      employee_id, last_name
sal x 12    ANNUAL SALARY
FROM        employees;
```

- The EMPLOYEES table does not contain a column called **sal**. The column is called **SALARY**.
 - The multiplication operator is *****, not **x**, as shown in line 2.
 - The **ANNUAL SALARY** alias cannot include spaces. The alias should read **ANNUAL_SALARY** or be enclosed in double quotation marks.
 - A comma is missing after the column, **LAST_NAME**.
6. Show the structure of the DEPARTMENTS table. Select all data from the DEPARTMENTS table.

```
DESCRIBE departments
```

```
SELECT *
FROM departments;
```

7. Show the structure of the EMPLOYEES table. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first. Provide an alias STARTDATE for the HIRE_DATE column. Save your SQL statement to a file named lab1_7.sql.

```
DESCRIBE employees
```

```
SELECT employee_id, last_name, job_id, hire_date StartDate
FROM employees;
```

Practice 1 Solutions (continued)

8. Run your query in the file lab1_7.sql.

```
SELECT employee_id, last_name, job_id, hire_date
FROM employees;
```

9. Create a query to display unique job codes from the EMPLOYEES table.

```
SELECT DISTINCT job_id
FROM employees;
```

If you have time, complete the following exercises:

10. Copy the statement from lab1_7.sql into the iSQL*Plus Edit window. Name the column headings Emp #, Employee, Job, and Hire Date, respectively. Run your query again.

```
SELECT employee_id "Emp #", last_name "Employee",
       job_id "Job", hire_date "Hire Date"
FROM employees;
```

11. Display the last name concatenated with the job ID, separated by a comma and space, and name the column Employee and Title.

```
SELECT last_name || ', ' || job_id "Employee and Title"
FROM employees;
```

If you want an extra challenge, complete the following exercise:

12. Create a query to display all the data from the EMPLOYEES table. Separate each column by a comma. Name the column THE_OUTPUT.

```
SELECT employee_id || ',' || first_name || ',' || last_name
       || ',' || email || ',' || phone_number || ',' || job_id
       || ',' || manager_id || ',' || hire_date || ',' ||
       salary || ',' || commission_pct || ',' || department_id
       THE_OUTPUT
FROM employees;
```

Practice 2 Solutions

1. Create a query to display the last name and salary of employees earning more than \$12,000. Place your SQL statement in a text file named `lab2_1.sql`. Run your query.

```
SELECT last_name, salary
FROM employees
WHERE salary > 12000;
```

2. Create a query to display the employee last name and department number for employee number 176.

```
SELECT last_name, department_id
FROM employees
WHERE employee_id = 176;
```

3. Modify `lab2_1.sql` to display the last name and salary for all employees whose salary is not in the range of \$5,000 and \$12,000. Place your SQL statement in a text file named `lab2_3.sql`.

```
SELECT last_name, salary
FROM employees
WHERE salary NOT BETWEEN 5000 AND 12000;
```

4. Display the employee last name, job ID, and start date of employees hired between February 20, 1998, and May 1, 1998. Order the query in ascending order by start date.

```
SELECT last_name, job_id, hire_date
FROM employees
WHERE hire_date BETWEEN '20-Feb-1998' AND '01-May-1998'
ORDER BY hire_date;
```

Practice 2 Solutions (continued)

5. Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.

```
SELECT last_name, department_id
FROM employees
WHERE department_id IN (20, 50)
ORDER BY last_name;
```

6. Modify lab2_3.sql to list the last name and salary of employees who earn between \$5,000 and \$12,000, and are in department 20 or 50. Label the columns Employee and Monthly Salary, respectively. Resave lab2_3.sql as lab2_6.sql. Run the statement in lab2_6.sql.

```
SELECT last_name "Employee", salary "Monthly Salary"
FROM employees
WHERE salary BETWEEN 5000 AND 12000
AND department_id IN (20, 50);
```

7. Display the last name and hire date of every employee who was hired in 1994.

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date LIKE '%94';
```

8. Display the last name and job title of all employees who do not have a manager.

```
SELECT last_name, job_id
FROM employees
WHERE manager_id IS NULL;
```

9. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

```
SELECT last_name, salary, commission_pct
FROM employees
WHERE commission_pct IS NOT NULL
ORDER BY salary DESC, commission_pct DESC;
```

Practice 2 Solutions (continued)

If you have time, complete the following exercises.

10. Display the last names of all employees where the third letter of the name is an *a*.

```
SELECT last_name
FROM employees
WHERE last_name LIKE '__a%';
```

11. Display the last name of all employees who have an *a* and an *e* in their last name.

```
SELECT last_name
FROM employees
WHERE last_name LIKE '%a%'
AND last_name LIKE '%e%';
```

If you want an extra challenge, complete the following exercises:

12. Display the last name, job, and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to \$2,500, \$3,500, or \$7,000.

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id IN ('SA_REP', 'ST_CLERK')
AND salary NOT IN (2500, 3500, 7000);
```

13. Modify lab2_6.sql to display the last name, salary, and commission for all employees whose commission amount is 20%. Resave lab2_6.sql as lab2_13.sql. Rerun the statement in lab2_13.sql.

```
SELECT last_name "Employee", salary "Monthly Salary",
commission_pct
FROM employees
WHERE commission_pct = .20;
```

Practice 3 Solutions

1. Write a query to display the current date. Label the column Date.

```
SELECT    sysdate "Date"
FROM      dual;
```

2. For each employee, display the employee number, last_name, salary, and salary increased by 15% and expressed as a whole number. Label the column New Salary. Place your SQL statement in a text file named lab3_2.sql.

```
SELECT    employee_id, last_name, salary,
          ROUND(salary * 1.15, 0) "New Salary"
FROM      employees;
```

3. Run your query in the file lab3_2.sql.

```
SELECT    employee_id, last_name, salary,
          ROUND(salary * 1.15, 0) "New Salary"
FROM      employees;
```

4. Modify your query lab3_2.sql to add a column that subtracts the old salary from the new salary. Label the column Increase. Save the contents of the file as lab3_4.sql. Run the revised query.

```
SELECT    employee_id, last_name, salary,
          ROUND(salary * 1.15, 0) "New Salary",
          ROUND(salary * 1.15, 0) - salary "Increase"
FROM      employees;
```

5. Write a query that displays the employee's last names with the first letter capitalized and all other letters lowercase and the length of the name for all employees whose name starts with J, A, or M. Give each column an appropriate label. Sort the results by the employees' last names.

```
SELECT    INITCAP(last_name) "Name",
          LENGTH(last_name) "Length"
FROM      employees
WHERE     last_name LIKE 'J%'
OR        last_name LIKE 'M%'
OR        last_name LIKE 'A%'
ORDER BY last_name;
```

Practice 3 Solutions (continued)

6. For each employee, display the employee's last name, and calculate the number of months between today and the date the employee was hired. Label the column MONTHS_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

Note: Your results will differ.

```
SELECT last_name, ROUND(MONTHS_BETWEEN
                        (SYSDATE, hire_date)) MONTHS_WORKED
FROM employees
ORDER BY MONTHS_BETWEEN(SYSDATE, hire_date);
```

7. Write a query that produces the following for each employee:
<employee last name> earns <salary> monthly but wants <3 times salary>. Label the column Dream Salaries.

```
SELECT last_name || ' earns '
       || TO_CHAR(salary, 'fm$99,999.00')
       || ' monthly but wants '
       || TO_CHAR(salary * 3, 'fm$99,999.00')
       || '.' "Dream Salaries"
FROM employees;
```

If you have time, complete the following exercises:

8. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with \$. Label the column SALARY.

```
SELECT last_name,
       LPAD(salary, 15, '$') SALARY
FROM employees;
```

9. Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear in the format similar to "Monday, the Thirty-First of July, 2000."

```
SELECT last_name, hire_date,
       TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date, 6), 'MONDAY'),
              'fmDay, "the" Ddspth "of" Month, YYYY') REVIEW
FROM employees;
```

10. Display the last name, hire date, and day of the week on which the employee started. Label the column DAY. Order the results by the day of the week starting with Monday.

```
SELECT last_name, hire_date,
       TO_CHAR(hire_date, 'DAY') DAY
FROM employees
ORDER BY TO_CHAR(hire_date - 1, 'd');
```


Practice 3 Solutions (continued)

If you want an extra challenge, complete the following exercises:

11. Create a query that displays the employees' last names and commission amounts. If an employee does not earn commission, put "No Commission." Label the column COMM.

```
SELECT    last_name ,
          NVL(TO_CHAR(commission_pct), 'No Commission') COMM
FROM      employees;
```

12. Create a query that displays the employees' last names and indicates the amounts of their annual salaries with asterisks. Each asterisk signifies a thousand dollars. Sort the data in descending order of salary. Label the column EMPLOYEES_AND_THEIR_SALARIES.

```
SELECT    rpad(last_name, 8)||' '|| rpad(' ', salary/1000+1, '*')
          EMPLOYEES_AND_THEIR_SALARIES
FROM      employees
ORDER BY  salary DESC;
```

13. Using the DECODE function, write a query that displays the grade of all employees based on the value of the column JOB_ID, as per the following data:

<i>JOB</i>	<i>GRADE</i>
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA_REP	D
ST_CLERK	E
None of the above	0

```
SELECT job_id, decode (job_id,
                      'ST_CLERK', 'E',
                      'SA_REP', 'D',
                      'IT_PROG', 'C',
                      'ST_MAN', 'B',
                      'AD_PRES', 'A',
                      '0')GRADE
FROM employees;
```

Practice 3 Solutions (continued)

14. Rewrite the statement in the preceding question using the CASE syntax.

```
SELECT job_id, CASE job_id
      WHEN 'ST_CLERK' THEN 'E'
      WHEN 'SA_REP'   THEN 'D'
      WHEN 'IT_PROG'  THEN 'C'
      WHEN 'ST_MAN'   THEN 'B'
      WHEN 'AD_PRES'  THEN 'A'
      ELSE '0'        END GRADE
FROM employees;
```

Practice 4 Solutions

1. Write a query to display the last name, department number, and department name for all employees.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

2. Create a unique listing of all jobs that are in department 80. Include the location of the department in the output.

```
SELECT DISTINCT job_id, location_id
FROM employees, departments
WHERE employees.department_id = departments.department_id
AND employees.department_id = 80;
```

3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission.

```
SELECT e.last_name, d.department_name, d.location_id, l.city
FROM employees e, departments d, locations l
WHERE e.department_id = d.department_id
AND
d.location_id = l.location_id
AND e.commission_pct IS NOT NULL;
```

4. Display the employee last name and department name for all employees who have an *a* (lowercase) in their last names. Place your SQL statement in a text file named lab4_4.sql.

```
SELECT last_name, department_name
FROM employees, departments
WHERE employees.department_id = departments.department_id
AND last_name LIKE '%a%';
```

Practice 4 Solutions (continued)

5. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

```
SELECT e.last_name, e.job_id, e.department_id,
       d.department_name
FROM employees e JOIN departments d
ON (e.department_id = d.department_id)
JOIN locations l
ON (d.location_id = l.location_id)
WHERE LOWER(l.city) = 'toronto';
```

6. Display the employee last name and employee number along with their manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Place your SQL statement in a text file named lab4_6.sql.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id "Mgr#"
FROM employees w join employees m
ON (w.manager_id = m.employee_id);
```

Practice 4 Solutions (continued)

7. Modify lab4_6.sql to display all employees including King, who has no manager. Place your SQL statement in a text file named lab4_7.sql. Run the query in lab4_7.sql

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id "Mgr#"
FROM employees w
LEFT OUTER JOIN employees m
ON (w.manager_id = m.employee_id);
```

If you have time, complete the following exercises.

8. Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label.

```
SELECT e.department_id department, e.last_name employee,
       c.last_name colleague
FROM   employees e JOIN employees c
ON     (e.department_id = c.department_id)
WHERE  e.employee_id <> c.employee_id
ORDER BY e.department_id, e.last_name, c.last_name;
```

9. Show the structure of the JOB_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees.

```
DESC JOB_GRADES
SELECT e.last_name, e.job_id, d.department_name,
       e.salary, j.grade_level
FROM   employees e, departments d, job_grades j
WHERE  e.department_id = d.department_id
AND    e.salary BETWEEN j.lowest_sal AND j.highest_sal;
-- OR
SELECT e.last_name, e.job_id, d.department_name,
       e.salary, j.grade_level
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
JOIN   job_grades j
ON     (e.salary BETWEEN j.lowest_sal AND j.highest_sal);
```

Practice 4 Solutions (continued)

If you want an extra challenge, complete the following exercises:

10. Create a query to display the name and hire date of any employee hired after employee Davies.

```
SELECT e.last_name, e.hire_date
FROM   employees e, employees davies
WHERE  davies.last_name = 'Davies'
AND    davies.hire_date < e.hire_date
-- OR
SELECT e.last_name, e.hire_date
FROM   employees e JOIN employees davies
ON     (davies.last_name = 'Davies')
WHERE  davies.hire_date < e.hire_date;
```

11. Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee, Emp Hired, Manager, and Mgr Hired, respectively.

```
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
FROM   employees w, employees m
WHERE  w.manager_id = m.employee_id
AND    w.hire_date < m.hire_date;
-- OR
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
FROM   employees w JOIN employees m
ON     (w.manager_id = m.employee_id)
WHERE  w.hire_date < m.hire_date;
```

Practice 5 Solutions

Determine the validity of the following three statements. Circle either True or False.

1. Group functions work across many rows to produce one result.

True

2. Group functions include nulls in calculations.

False. Group functions ignore null values. If you want to include null values, use the NVL function.

3. The WHERE clause restricts rows prior to inclusion in a group calculation.

True

4. Display the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number. Place your SQL statement in a text file named lab5_6.sql.

```
SELECT    ROUND(MAX(salary),0) "Maximum",
          ROUND(MIN(salary),0) "Minimum",
          ROUND(SUM(salary),0) "Sum",
          ROUND(AVG(salary),0) "Average"
FROM      employees;
```

5. Modify the query in lab5_4.sql to display the minimum, maximum, sum, and average salary for each job type. Resave lab5_6.sql to lab5_4.sql. Run the statement in lab5_5.sql.

```
SELECT    job_id, ROUND(MAX(salary),0) "Maximum",
          ROUND(MIN(salary),0) "Minimum",
          ROUND(SUM(salary),0) "Sum",
          ROUND(AVG(salary),0) "Average"
FROM      employees
GROUP BY  job_id;
```

Practice 5 Solutions (continued)

6. Write a query to display the number of people with the same job.

```
SELECT  job_id, COUNT(*)
FROM    employees
GROUP BY job_id;
```

7. Determine the number of managers without listing them. Label the column Number of Managers. *Hint: Use the MANAGER_ID column to determine the number of managers.*

```
SELECT  COUNT(DISTINCT manager_id) "Number of Managers"
FROM    employees;
```

8. Write a query that displays the difference between the highest and lowest salaries. Label the column DIFFERENCE.

```
SELECT  MAX(salary) - MIN(salary) DIFFERENCE
FROM    employees;
```

If you have time, complete the following exercises.

9. Display the manager number and the salary of the lowest paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is \$6,000 or less. Sort the output in descending order of salary.

```
SELECT  manager_id, MIN(salary)
FROM    employees
WHERE   manager_id IS NOT NULL
GROUP BY manager_id
HAVING  MIN(salary) > 6000
ORDER BY MIN(salary) DESC;
```

10. Write a query to display each department's name, location, number of employees, and the average salary for all employees in that department. Label the columns Name, Location, Number of People, and Salary, respectively. Round the average salary to two decimal places.

```
SELECT  d.department_name "Name", d.location_id "Location",
        COUNT(*) "Number of People",
        ROUND(AVG(salary),2) "Salary"
FROM    employees e, departments d
WHERE   e.department_id = d.department_id
GROUP BY d.department_name, d.location_id;
```


Practice 5 Solutions (continued)

If you want an extra challenge, complete the following exercises:

11. Create a query that will display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

```
SELECT  COUNT(*) total,
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1995,1,0))"1995",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1996,1,0))"1996",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1997,1,0))"1997",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1998,1,0))"1998"
FROM    employees;
```

12. Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

```
SELECT  job_id "Job",
        SUM(DECODE(department_id , 20, salary)) "Dept 20",
        SUM(DECODE(department_id , 50, salary)) "Dept 50",
        SUM(DECODE(department_id , 80, salary)) "Dept 80",
        SUM(DECODE(department_id , 90, salary)) "Dept 90",
        SUM(salary) "Total"
FROM    employees
GROUP BY job_id;
```

Practice 6 Solutions

1. Write a query to display the last name and hire date of any employee in the same department as Zlotkey. Exclude Zlotkey.

```
SELECT last_name, hire_date
FROM employees
WHERE department_id = (SELECT department_id
                       FROM employees
                       WHERE last_name = 'Zlotkey')
AND last_name <> 'Zlotkey';
```

2. Create a query to display the employee numbers and last names of all employees who earn more than the average salary. Sort the results in ascending order of salary.

```
SELECT employee_id, last_name
FROM employees
WHERE salary > (SELECT AVG(salary)
                FROM employees)
ORDER BY salary;
```

3. Write a query that displays the employee numbers and last names of all employees who work in a department with any employee whose last name contains a *u*. Place your SQL statement in a text file named lab6_3.sql. Run your query.

```
SELECT employee_id, last_name
FROM employees
WHERE department_id IN (SELECT department_id
                       FROM employees
                       WHERE last_name like '%u%');
```

4. Display the last name, department number, and job ID of all employees whose department location ID is 1700.

```
SELECT last_name, department_id, job_id
FROM employees
WHERE department_id IN (SELECT department_id
                       FROM departments
                       WHERE location_id = 1700);
```

Practice 6 Solutions (continued)

5. Display the last name and salary of every employee who reports to King.

```
SELECT last_name, salary
FROM   employees
WHERE  manager_id = (SELECT employee_id
                    FROM   employees
                    WHERE  last_name = 'King');
```

6. Display the department number, last name, and job ID for every employee in the Executive department.

```
SELECT department_id, last_name, job_id
FROM   employees
WHERE  department_id IN (SELECT department_id
                       FROM   departments
                       WHERE  department_name = 'Executive');
```

If you have time, complete the following exercises:

7. Modify the query in lab6_3.sql to display the employee numbers, last names, and salaries of all employees who earn more than the average salary and who work in a department with any employee with a *u* in their name. Resave lab6_3.sql to lab6_7.sql. Run the statement in lab6_7.sql.

```
SELECT employee_id, last_name, salary
FROM   employees
WHERE  department_id IN (SELECT department_id
                       FROM   employees
                       WHERE  last_name like '%u%')
AND    salary > (SELECT AVG(salary)
                FROM   employees);
```

Practice 7 Solutions

Determine whether the following statements are true or false:

1. The following statement is correct:

```
DEFINE & p_val = 100
```

False

The correct use of DEFINE is DEFINE p_val=100. The & is used within the SQL code.

2. The DEFINE command is a SQL command.

False

The DEFINE command is an iSQL*Plus command.

3. Write a script to display the employee last name, job, and hire date for all employees who started between a given range. Concatenate the name and job together, separated by a space and comma, and label the column Employees. In a separate SQL script file, use the DEFINE command to provide the two ranges. Use the format MM/DD/YYYY. Save the script files as lab7_3a.sql and lab7_3b.sql.

```
-- lab file lab7_3a.sql
SET ECHO OFF
SET VERIFY OFF
DEFINE low_date = 01/01/1998
DEFINE high_date = 01/01/1999

-- lab file lab7_3a.sql
SELECT last_name || ', ' || job_id EMPLOYEES, hire_date
FROM employees
WHERE hire_date BETWEEN TO_DATE('&low_date', 'MM/DD/YYYY')
AND TO_DATE('&high_date', 'MM/DD/YYYY')

/
UNDEFINE low_date
UNDEFINE high_date
SET VERIFY ON
SET ECHO ON
```

Practice 7 Solutions (continued)

4. Write a script to display the employee last name, job, and department name for a given location. The search condition should allow for case-insensitive searches of the department location. Save the script file as lab7_4.sql.

```
SET ECHO OFF
SET VERIFY OFF
COLUMN last_name HEADING "EMPLOYEE NAME"
COLUMN department_name HEADING "DEPARTMENT NAME"
SELECT e.last_name, e.job_id, d.department_name
FROM employees e, departments d, locations l
WHERE e.department_id = d.department_id
AND l.location_id = d.location_id
AND l.city = INITCAP('&p_location')
/
COLUMN last_name CLEAR
COLUMN department_name CLEAR
SET VERIFY ON
SET ECHO ON
```

Practice 7 Solutions (continued)

5. Modify the code in lab7_4.sql to create a report containing the department name, employee last name, hire date, salary, and each employee's annual salary for all employees in a given location. Label the columns DEPARTMENT NAME, EMPLOYEE NAME, START DATE, SALARY, and ANNUAL SALARY, placing the labels on multiple lines. Resave the script as lab7_5.sql and execute the commands in the script.

```
SET ECHO OFF
SET FEEDBACK OFF
SET VERIFY OFF
BREAK ON department_name
COLUMN department_name HEADING "DEPARTMENT|NAME"
COLUMN last_name HEADING "EMPLOYEE|NAME"
COLUMN hire_date HEADING "START|DATE"
COLUMN salary HEADING "SALARY" FORMAT $99,990.00
COLUMN asal HEADING "ANNUAL|SALARY" FORMAT $99,990.00
SELECT d.department_name, e.last_name, e.hire_date,
       e.salary, e.salary*12 asal
FROM   departments d, employees e, locations l
WHERE  e.department_id = d.department_id
AND    d.location_id   = l.location_id
AND    l.city          = '&p_location'
ORDER BY d.department_name

/
COLUMN department_name CLEAR
COLUMN last_name CLEAR
COLUMN hire_date CLEAR
COLUMN salary CLEAR
COLUMN asal CLEAR
CLEAR BREAK
SET VERIFY ON
SET FEEDBACK ON
SET ECHO ON
```

Practice 8 Solutions

Insert data into the MY_EMPLOYEE table.

1. Run the statement in the lab8_1.sql script to build the MY_EMPLOYEE table that will be used for the lab.

```
CREATE TABLE my_employee
(id NUMBER(4) CONSTRAINT my_employee_id_nn NOT NULL,
last_name VARCHAR2(25),
first_name VARCHAR2(25),
userid VARCHAR2(8),
salary NUMBER(9,2));
```

2. Describe the structure of the MY_EMPLOYEE table to identify the column names.

```
DESCRIBE my_employee
```

3. Add the first row of data to the MY_EMPLOYEE table from the following sample data. Do not list the columns in the INSERT clause.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

```
INSERT INTO my_employee
VALUES (1, 'Patel', 'Ralph', 'rpatel', 895);
```

4. Populate the MY_EMPLOYEE table with the second row of sample data from the preceding list. This time, list the columns explicitly in the INSERT clause.

```
INSERT INTO my_employee (id, last_name, first_name,
userid, salary)
VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);
```

5. Confirm your addition to the table.

```
SELECT *
FROM my_employee;
```

Practice 8 Solutions (continued)

6. Write an insert statement in a text file named `loademp.sql` to load rows into the `MY_EMPLOYEE` table. Concatenate the first letter of the first name and the first seven characters of the last name to produce the `userid`.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
        lower(substr('&p_first_name', 1, 1) ||
                 substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
```

7. Populate the table with the next two rows of sample data by running the insert statement in the script that you created.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
        lower(substr('&p_first_name', 1, 1) ||
                 substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
```

8. Confirm your additions to the table.

```
SELECT *
FROM my_employee;
```

9. Make the data additions permanent.

```
COMMIT;
```


Practice 8 Solutions (continued)

Update and delete data in the MY_EMPLOYEE table.

10. Change the last name of employee 3 to Drexler.

```
UPDATE my_employee
SET    last_name = 'Drexler'
WHERE  id = 3;
```

11. Change the salary to 1000 for all employees with a salary less than 900.

```
UPDATE my_employee
SET    salary = 1000
WHERE  salary < 900;
```

12. Verify your changes to the table.

```
SELECT last_name, salary
FROM    my_employee;
```

13. Delete Betty Dancs from the MY_EMPLOYEE table.

```
DELETE
FROM    my_employee
WHERE  last_name = 'Dancs';
```

14. Confirm your changes to the table.

```
SELECT *
FROM    my_employee;
```

15. Commit all pending changes.

```
COMMIT;
```

Control data transaction to the MY_EMPLOYEE table.

16. Populate the table with the last row of sample data by modifying the statements in the script that you created in step 6. Run the statements in the script.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
       lower(substr('&p_first_name', 1, 1) ||
       substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
```

Practice 8 Solutions (continued)

17. Confirm your addition to the table.

```
SELECT *  
FROM my_employee;
```

18. Mark an intermediate point in the processing of the transaction.

```
SAVEPOINT step_18;
```

19. Empty the entire table.

```
DELETE  
FROM my_employee;
```

20. Confirm that the table is empty.

```
SELECT *  
FROM my_employee;
```

21. Discard the most recent DELETE operation without discarding the earlier INSERT operation.

```
ROLLBACK TO step_18;
```

22. Confirm that the new row is still intact.

```
SELECT *  
FROM my_employee;
```

23. Make the data addition permanent.

```
COMMIT;
```

Practice 9 Solutions

1. Create the DEPT table based on the following table instance chart. Place the syntax in a script called lab9_1.sql, then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	NAME
Key Type		
Nulls/Unique		
FK Table		
FK Column		
Data type	Number	VARCHAR2
Length	7	25

```
CREATE TABLE dept
(id NUMBER(7),
 name VARCHAR2(25));
```

```
DESCRIBE dept
```

2. Populate the DEPT table with data from the DEPARTMENTS table. Include only columns that you need.

```
INSERT INTO dept
SELECT department_id, department_name
FROM departments;
```

3. Create the EMP table based on the following table instance chart. Place the syntax in a script called lab9_3.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK Table				
FK Column				
Data type	Number	VARCHAR2	VARCHAR2	Number
Length	7	25	25	7

Practice 9 Solutions (continued)

```
CREATE TABLE emp
(id          NUMBER(7),
 last_name   VARCHAR2(25),
 first_name  VARCHAR2(25),
 dept_id     NUMBER(7));
```

```
DESCRIBE emp
```

4. Modify the EMP table to allow for longer employee last names. Confirm your modification.

```
ALTER TABLE emp
MODIFY (last_name   VARCHAR2(50));
```

```
DESCRIBE emp
```

5. Confirm that both the DEPT and EMP tables are stored in the data dictionary. (Hint: USER_TABLES)

```
SELECT table_name
FROM user_tables
WHERE table_name IN ('DEPT', 'EMP');
```

6. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY, and DEPARTMENT_ID columns. Name the columns in your new table ID, FIRST_NAME, LAST_NAME, SALARY, and DEPT_ID, respectively.

```
CREATE TABLE employees2 AS
SELECT employee_id id, first_name, last_name, salary,
       department_id dept_id
FROM employees;
```

7. Drop the EMP table.

```
DROP TABLE emp;
```

8. Rename the EMPLOYEES2 table to EMP.

```
RENAME employees2 TO emp;
```

Practice 9 Solutions (continued)

9. Add a comment to the DEPT and EMP table definitions describing the tables. Confirm your additions in the data dictionary.

```
COMMENT ON TABLE emp IS 'Employee Information';
COMMENT ON TABLE dept IS 'Department Information';
SELECT *
FROM   user_tab_comments
WHERE  table_name = 'DEPT'
OR     table_name = 'EMP';
```

10. Drop the FIRST_NAME column from the EMP table. Confirm your modification by checking the description of the table.

```
ALTER TABLE emp
DROP COLUMN FIRST_NAME;

DESCRIBE emp
```

11. In the EMP table, mark the DEPT_ID column in the EMP table as UNUSED. Confirm your modification by checking the description of the table.

```
ALTER TABLE   emp
SET   UNUSED (dept_id);

DESCRIBE emp
```

12. Drop all the UNUSED columns from the EMP table. Confirm your modification by checking the description of the table.

```
ALTER TABLE emp
DROP UNUSED COLUMNS;

DESCRIBE emp
```

Practice 10 Solutions

1. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column. The constraint should be named at creation. Name the constraint my_emp_id_pk

```
ALTER TABLE emp
ADD CONSTRAINT my_emp_id_pk PRIMARY KEY (id);
```

2. Create a PRIMARY KEY constraint to the DEPT table using the ID column. The constraint should be named at creation. Name the constraint my_dept_id_pk.

```
ALTER TABLE dept
ADD CONSTRAINT my_dept_id_pk PRIMARY KEY(id);
```

3. Add a column DEPT_ID to the EMP table. Add a foreign key reference on the EMP table that ensures that the employee is not assigned to a nonexistent department. Name the constraint my_emp_dept_id_fk.

```
ALTER TABLE emp
ADD (dept_id NUMBER(7));

ALTER TABLE emp
ADD CONSTRAINT my_emp_dept_id_fk
FOREIGN KEY (dept_id) REFERENCES dept(id);
```

4. Confirm that the constraints were added by querying the USER_CONSTRAINTS view. Note the types and names of the constraints. Save your statement text in a file called lab10_4.sql.

```
SELECT constraint_name, constraint_type
FROM user_constraints
WHERE table_name IN ('EMP', 'DEPT');
```

5. Display the object names and types from the USER_OBJECTS data dictionary view for the EMP and DEPT tables. Notice that the new tables and a new index were created.

```
SELECT object_name, object_type
FROM user_objects
WHERE object_name LIKE 'EMP%'
OR object_name LIKE 'DEPT%';
```

If you have time, complete the following exercise:

6. Modify the EMP table. Add a COMMISSION column of NUMBER data type, precision 2, scale 2. Add a constraint to the commission column that ensures that a commission value is greater than zero.

```
ALTER TABLE EMP
ADD commission NUMBER(2,2)
CONSTRAINT my_emp_comm_ck CHECK (commission >= 0);
```

Practice 11 Solutions

1. Create a view called EMPLOYEES_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.

```
CREATE OR REPLACE VIEW employees_vu AS
SELECT employee_id, last_name employee, department_id
FROM employees;
```

2. Display the contents of the EMPLOYEES_VU view.

```
SELECT *
FROM employees_vu;
```

3. Select the view name and text from the USER_VIEWS data dictionary view.

Note: Another view already exists. The EMP_DETAILS_VIEW was created as part of your schema.

Note: To see more contents of a LONG column, use the iSQL*Plus command SET LONG *n*, where *n* is the value of the number of characters of the LONG column that you want to see.

```
SET LONG 600
SELECT view_name, text
FROM user_views;
```

4. Using your EMPLOYEES_VU view, enter a query to display all employee names and department numbers.

```
SELECT employee, department_id
FROM employees_vu;
```

5. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. Label the view columns EMPNO, EMPLOYEE, and DEPTNO. Do not allow an employee to be reassigned to another department through the view.

```
CREATE VIEW dept50 AS
SELECT employee_id empno, last_name employee,
       department_id deptno
FROM employees
WHERE department_id = 50
WITH CHECK OPTION CONSTRAINT emp_dept_50;
```

Practice 11 Solutions (continued)

6. Display the structure and contents of the DEPT50 view.

```
DESCRIBE dept50
SELECT  *
FROM    dept50;
```

7. Attempt to reassign Matos to department 80.

```
UPDATE  dept50
SET     deptno = 80
WHERE   employee = 'Matos';
```

If you have time, complete the following exercise:

8. Create a view called SALARY_VU based on the employee last names, department names, salaries, and salary grades for all employees. Use the EMPLOYEES, DEPARTMENTS, and JOB_GRADES tables. Label the columns Employee, Department, Salary, and Grade, respectively.

```
CREATE OR REPLACE VIEW salary_vu
AS
SELECT e.last_name "Employee",
       d.department_name "Department",
       e.salary "Salary",
       j.grade_level "Grades"
FROM   employees e,
       departments d,
       job_grades j
WHERE  e.department_id = d.department_id
AND    e.salary BETWEEN j.lowest_sal and j.highest_sal;
```


Practice 12 Solutions

1. Create a sequence to be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1000. Have your sequence increment by ten numbers. Name the sequence DEPT_ID_SEQ.

```
CREATE SEQUENCE dept_id_seq
START WITH 200
INCREMENT BY 10
MAXVALUE 1000;
```

2. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number. Name the script lab12_2.sql. Run the statement in your script.

```
SELECT    sequence_name, max_value, increment_by, last_number
FROM      user_sequences;
```

3. Write a script to insert two rows into the DEPT table. Name your script lab12_3.sql. Be sure to use the sequence that you created for the ID column. Add two departments named Education and Administration. Confirm your additions. Run the commands in your script.

```
INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Education');

INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Administration');
```

4. Create a nonunique index on the foreign key column (DEPT_ID) in the EMP table.

```
CREATE INDEX emp_dept_id_idx ON emp (dept_id);
```

5. Display the indexes and uniqueness that exist in the data dictionary for the EMP table. Save the statement into a script named lab12_5.sql.

```
SELECT    index_name, table_name, uniqueness
FROM      user_indexes
WHERE     table_name = 'EMP';
```

Practice 13 Solutions

1. What privilege should a user be given to log on to the Oracle Server? Is this a system or an object privilege?

The CREATE SESSION system privilege

2. What privilege should a user be given to create tables?

The CREATE TABLE privilege

3. If you create a table, who can pass along privileges to other users on your table?

You can, or anyone you have given those privileges to by using the WITH GRANT OPTION.

4. You are the DBA. You are creating many users who require the same system privileges. What should you use to make your job easier?

Create a role containing the system privileges and grant the role to the users

5. What command do you use to change your password?

The ALTER USER statement

6. Grant another user access to your DEPARTMENTS table. Have the user grant you query access to his or her DEPARTMENTS table.

Team 2 executes the GRANT statement.

```
GRANT select
ON    departments
TO    <user1>;
```

Team 1 executes the GRANT statement.

```
GRANT select
ON    departments
TO    <user2>;
```

WHERE *user1* is the name of team 1 and *user2* is the name of team 2.

7. Query all the rows in your DEPARTMENTS table.

```
SELECT *
FROM   departments;
```

Practice 13 Solutions (continued)

8. Add a new row to your DEPARTMENTS table. Team 1 should add Education as department number 500. Team 2 should add Human Resources department number 510. Query the other team's table.

Team 1 executes this INSERT statement.

```
INSERT INTO departments(department_id, department_name)
VALUES (500, 'Education');
COMMIT;
```

Team 2 executes this INSERT statement.

```
INSERT INTO departments(department_id, department_name)
VALUES (510, 'Administration');
COMMIT;
```

9. Create a synonym for the other team's DEPARTMENTS table.

Team 1 creates a synonym named team2.

```
CREATE SYNONYM team2
FOR <user2>.DEPARTMENTS;
```

Team 2 creates a synonym named team1.

```
CREATE SYNONYM team1
FOR <user1>. DEPARTMENTS;
```

10. Query all the rows in the other team's DEPARTMENTS table by using your synonym.

Team 1 executes this SELECT statement.

```
SELECT *
FROM team2;
```

Team 2 executes this SELECT statement.

```
SELECT *
FROM team1;
```

Practice 13 Solutions (continued)

11. Query the USER_TABLES data dictionary to see information about the tables that you own.

```
SELECT table_name
FROM user_tables;
```

12. Query the ALL_TABLES data dictionary view to see information about all the tables that you can access. Exclude tables that you own.

```
SELECT table_name, owner
FROM all_tables
WHERE owner <> <your account>;
```

13. Revoke the SELECT privilege from the other team.

Team 1 revokes the privilege.

```
REVOKE select
ON departments
FROM user2;
```

Team 2 revokes the privilege.

```
REVOKE select
ON departments
FROM user1;
```

14. Remove the row you inserted into the DEPARTMENTS table in step 8 and save the changes.

Team 1 executes this INSERT statement.

```
DELETE FROM departments
WHERE department_id = 500;
COMMIT;
```

Team 2 executes this INSERT statement.

```
DELETE FROM departments
WHERE department_id = 510;
COMMIT;
```

Practice 14 Solutions

1. Create the tables based on the following table instance charts. Choose the appropriate data types and be sure to add integrity constraints.

a. Table name: MEMBER

Column Name	MEMBER_ID	LAST_NAME	FIRST_NAME	ADDRESS	CITY	PHONE	JOIN_DATE
Key Type	PK						
Null/Unique	NN,U	NN					NN
Default Value							System Date
Data Type	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
Length	10	25	25	100	30	15	

```

CREATE TABLE member
(member_id      NUMBER(10)
CONSTRAINT member_member_id_pk PRIMARY KEY,
last_name      VARCHAR2(25)
CONSTRAINT member_last_name_nn NOT NULL,
first_name     VARCHAR2(25),
address        VARCHAR2(100),
city           VARCHAR2(30),
phone          VARCHAR2(15),
join_date      DATE DEFAULT SYSDATE
CONSTRAINT member_join_date_nn NOT NULL);

```

Practice 14 Solutions (continued)

b. Table name: TITLE

Column_ Name	TITLE_ID	TITLE	DESCRIPTION	RATING	CATEGORY	RELEASE_ DATE
Key Type	PK					
Null/ Unique	NN,U	NN	NN			
Check				G, PG, R, NC17, NR	DRAMA, COMEDY, ACTION, CHILD, SCIFI, DOCUMENTARY	
Data Type	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
Length	10	60	400	4	20	

```
CREATE TABLE title
(title_id NUMBER(10)
    CONSTRAINT title_title_id_pk PRIMARY KEY,
title VARCHAR2(60)
    CONSTRAINT title_title_nn NOT NULL,
description VARCHAR2(400)
    CONSTRAINT title_description_nn NOT NULL,
rating VARCHAR2(4)
    CONSTRAINT title_rating_ck CHECK
        (rating IN ('G', 'PG', 'R', 'NC17', 'NR')),
category VARCHAR2(20),
    CONSTRAINT title_category_ck CHECK
        (category IN ('DRAMA', 'COMEDY', 'ACTION',
            'CHILD', 'SCIFI', 'DOCUMENTARY'))),
release_date DATE);
```

Practice 14 Solutions (continued)

c. Table name: TITLE_COPY

Column Name	COPY_ID	TITLE_ID	STATUS
Key Type	PK	PK,FK	
Null/Unique	NN,U	NN,U	NN
Check			AVAILABLE, DESTROYED, RENTED, RESERVED
FK Ref Table		TITLE	
FK Ref Col		TITLE_ID	
Data Type	NUMBER	NUMBER	VARCHAR2
Length	10	10	15

```
CREATE TABLE title_copy
(copy_id      NUMBER(10),
 title_id    NUMBER(10)

 CONSTRAINT title_copy_title_id_fk REFERENCES title(title_id),
 status      VARCHAR2(15)
 CONSTRAINT title_copy_status_nn NOT NULL
 CONSTRAINT title_copy_status_ck CHECK (status IN
 ('AVAILABLE', 'DESTROYED', 'RENTED', 'RESERVED')),
 CONSTRAINT title_copy_copy_id_title_id_pk
 PRIMARY KEY (copy_id, title_id));
```

Practice 14 Solutions (continued)

d. Table name: RENTAL

Column Name	BOOK_DATE	MEMBER_ID	COPY_ID	ACT_RET_DATE	EXP_RET_DATE	TITLE_ID
Key Type	PK	PK,FK1	PK,FK2			PK,FK2
Default Value	System Date				System Date + 2 days	
FK Ref Table		MEMBER	TITLE_COPY			TITLE_COPY
FK Ref Col		MEMBER_ID	COPY_ID			TITLE_ID
Data Type	DATE	NUMBER	NUMBER	DATE	DATE	NUMBER
Length		10	10			10

```
CREATE TABLE rental
(book_date      DATE DEFAULT SYSDATE,
 member_id     NUMBER(10)
              CONSTRAINT rental_member_id_fk
              REFERENCES member(member_id),
 copy_id       NUMBER(10),
 act_ret_date  DATE,
 exp_ret_date  DATE DEFAULT SYSDATE + 2,
 title_id      NUMBER(10),
              CONSTRAINT rental_book_date_copy_title_pk
              PRIMARY KEY (book_date, member_id,
                          copy_id,title_id),
              CONSTRAINT rental_copy_id_title_id_fk
              FOREIGN KEY (copy_id, title_id)
              REFERENCES title_copy(copy_id, title_id));
```


Practice 14 Solutions (continued)

e. Table name: RESERVATION

Column Name	RES_ DATE	MEMBER_ ID	TITLE_ ID
Key Type	PK	PK,FK1	PK,FK2
Null/ Unique	NN,U	NN,U	NN
FK Ref Table		MEMBER	TITLE
FK Ref Column		MEMBER_ID	TITLE_ID
Data Type	DATE	NUMBER	NUMBER
Length		10	10

```
CREATE TABLE reservation
(res_date    DATE,
 member_id   NUMBER(10)
             CONSTRAINT reservation_member_id
             REFERENCES member(member_id),
 title_id    NUMBER(10)
             CONSTRAINT reservation_title_id
             REFERENCES title(title_id),
 CONSTRAINT reservation_resdate_mem_tit_pk PRIMARY KEY
 (res_date, member_id, title_id));
```

Practice 14 Solutions (continued)

2. Verify that the tables and constraints were created properly by checking the data dictionary.

```
SELECT table_name
FROM user_tables
WHERE table_name IN ('MEMBER', 'TITLE', 'TITLE_COPY',
                    'RENTAL', 'RESERVATION');
```

```
SELECT constraint_name, constraint_type, table_name
FROM user_constraints
WHERE table_name IN ('MEMBER', 'TITLE', 'TITLE_COPY',
                    'RENTAL', 'RESERVATION');
```

3. Create sequences to uniquely identify each row in the MEMBER table and the TITLE table.
 - a. Member number for the MEMBER table: start with 101; do not allow caching of the values. Name the sequence MEMBER_ID_SEQ.

```
CREATE SEQUENCE member_id_seq
START WITH 101
NOCACHE;
```

- b. Title number for the TITLE table: start with 92; no caching. Name the sequence TITLE_ID_SEQ.

```
CREATE SEQUENCE title_id_seq
START WITH 92
NOCACHE;
```

- c. Verify the existence of the sequences in the data dictionary.

```
SELECT sequence_name, increment_by, last_number
FROM user_sequences
WHERE sequence_name IN ('MEMBER_ID_SEQ', 'TITLE_ID_SEQ');
```

Practice 14 Solutions (continued)

4. Add data to the tables. Create a script for each set of data to add.
 - a. Add movie titles to the TITLE table. Write a script to enter the movie information. Save the statements in a script named lab14_4a.sql. Use the sequences to uniquely identify each title. Enter the release dates in the DD-MON-YYYY format. Remember that single quotation marks in a character field must be specially handled. Verify your additions.

```
SET ECHO OFF
INSERT INTO title(title_id, title, description, rating,
                 category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Willie and Christmas Too',
       'All of Willie''s friends make a Christmas list for
       Santa, but Willie has yet to add his own wish list.',
       'G', 'CHILD', TO_DATE('05-OCT-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id , title, description, rating,
                 category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Alien Again', 'Yet another
       installment of science fiction history. Can the
       heroine save the planet from the alien life form?',
       'R', 'SCIFI', TO_DATE( '19-MAY-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
                 category, release_date)
VALUES (title_id_seq.NEXTVAL, 'The Glob', 'A meteor crashes
       near a small American town and unleashes carnivorous
       goo in this classic.', 'NR', 'SCIFI',
       TO_DATE( '12-AUG-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
                 category, release_date)
VALUES (title_id_seq.NEXTVAL, 'My Day Off', 'With a little
       luck and a lot ingenuity, a teenager skips school for
       a day in New York.', 'PG', 'COMEDY',
       TO_DATE( '12-JUL-1995','DD-MON-YYYY'))
/
...
COMMIT
/
SET ECHO ON

SELECT title
FROM title;
```

Practice 14 Solutions (continued)

Title	Description	Rating	Category	Release_date
Willie and Christmas Too	All of Willie's friends make a Christmas list for Santa, but Willie has yet to add his own wish list.	G	CHILD	05-OCT-1995
Alien Again	Yet another installation of science fiction history. Can the heroine save the planet from the alien life form?	R	SCIFI	19-MAY-1995
The Glob	A meteor crashes near a small American town and unleashes carnivorous goo in this classic.	NR	SCIFI	12-AUG-1995
My Day Off	With a little luck and a lot of ingenuity, a teenager skips school for a day in New York.	PG	COMEDY	12-JUL-1995
Miracles on Ice	A six-year-old has doubts about Santa Claus, but she discovers that miracles really do exist.	PG	DRAMA	12-SEP-1995
Soda Gang	After discovering a cache of drugs, a young couple find themselves pitted against a vicious gang.	NR	ACTION	01-JUN-1995

Practice 14 Solutions (continued)

- b. Add data to the MEMBER table. Place the insert statements in a script named lab14_4b.sql. Execute commands in the script. Be sure to use the sequence to add the member numbers.

First_Name	Last_Name	Address	City	Phone	Join_Date
Carmen	Velasquez	283 King Street	Seattle	206-899-6666	08-MAR-1990
LaDoris	Ngao	5 Modrany	Bratislava	586-355-8882	08-MAR-1990
Midori	Nagayama	68 Via Centrale	Sao Paolo	254-852-5764	17-JUN-1991
Mark	Quick-to-See	6921 King Way	Lagos	63-559-7777	07-APR-1990
Audry	Ropeburn	86 Chu Street	Hong Kong	41-559-87	18-JAN-1991
Molly	Urguhart	3035 Laurier	Quebec	418-542-9988	18-JAN-1991

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO member(member_id, first_name, last_name, address,
                   city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, '&first_name', '&last_name',
       '&address', '&city', '&phone', TO_DATE('&join_date',
       'DD-MM-YYYY'));
COMMIT;
SET VERIFY ON
SET ECHO ON
```

Practice 14 Solutions (continued)

c. Add the following movie copies in the TITLE_COPY table:

Note: Have the TITLE_ID numbers available for this exercise.

Title	Copy_Id	Status
Willie and Christmas Too	1	AVAILABLE
Alien Again	1	AVAILABLE
	2	RENTED
The Glob	1	AVAILABLE
My Day Off	1	AVAILABLE
	2	AVAILABLE
	3	RENTED
Miracles on Ice	1	AVAILABLE
Soda Gang	1	AVAILABLE

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 92, 'AVAILABLE');
```

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 93, 'AVAILABLE');
```

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (2, 93, 'RENTED');
```

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 94, 'AVAILABLE');
```

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 95, 'AVAILABLE');
```

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (2, 95, 'AVAILABLE');
```

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (3, 95, 'RENTED');
```

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 96, 'AVAILABLE');
```

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 97, 'AVAILABLE');
```

Practice 14 Solutions (continued)

d. Add the following rentals to the RENTAL table:

Note: Title number may be different depending on sequence number.

Title_ Id	Copy_ Id	Member_ Id	Book_date	Exp_Ret_Date	Act_Ret_Date
92	1	101	3 days ago	1 day ago	2 days ago
93	2	101	1 day ago	1 day from now	
95	3	102	2 days ago	Today	
97	1	106	4 days ago	2 days ago	2 days ago

```
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date, act_ret_date)
VALUES (92, 1, 101, sysdate-3, sysdate-1, sysdate-2);
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date, act_ret_date)
VALUES (93, 2, 101, sysdate-1, sysdate-1, NULL);
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date, act_ret_date)
VALUES (95, 3, 102, sysdate-2, sysdate, NULL);
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date, act_ret_date)
VALUES (97, 1, 106, sysdate-4, sysdate-2, sysdate-2);
COMMIT;
```

Practice 14 Solutions (continued)

5. Create a view named TITLE_AVAIL to show the movie titles and the availability of each copy and its expected return date if rented. Query all rows from the view. Order the results by title.

```
CREATE VIEW title_avail AS
SELECT  t.title, c.copy_id, c.status, r.exp_ret_date
FROM    title t, title_copy c, rental r
WHERE   t.title_id = c.title_id
AND     c.copy_id = r.copy_id(+)
AND     c.title_id = r.title_id(+);

SELECT  *
FROM    title_avail
ORDER BY title, copy_id;
```

6. Make changes to data in the tables.
- a. Add a new title. The movie is “Interstellar Wars,” which is rated PG and classified as a science fiction movie. The release date is 07-JUL-77. The description is “Futuristic interstellar action movie. Can the rebels save the humans from the evil empire?” Be sure to add a title copy record for two copies.

```
INSERT INTO title(title_id, title, description, rating,
                 category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Interstellar Wars',
       'Futuristic interstellar action movie. Can the
       rebels save the humans from the evil Empire?',
       'PG', 'SCIFI', '07-JUL-77');

INSERT INTO title_copy (copy_id, title_id, status)
VALUES (1, 98, 'AVAILABLE');

INSERT INTO title_copy (copy_id, title_id, status)
VALUES (2, 98, 'AVAILABLE');
```

- b. Enter two reservations. One reservation is for Carmen Velasquez, who wants to rent “Interstellar Wars.” The other is for Mark Quick-to-See, who wants to rent “Soda Gang.”

```
INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, 101, 98);
INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, 104, 97);
```


Practice 14 Solutions (continued)

- c. Customer Carmen Velasquez rents the movie “Interstellar Wars,” copy 1. Remove her reservation for the movie. Record the information about the rental. Allow the default value for the expected return date to be used. Verify that the rental was recorded by using the view you created.

```
INSERT INTO rental(title_id, copy_id, member_id)
VALUES (98,1,101);
UPDATE title_copy
SET    status= 'RENTED'
WHERE  title_id = 98
AND    copy_id = 1;
DELETE
FROM   reservation
WHERE  member_id = 101;
SELECT *
FROM   title_avail
ORDER BY title, copy_id;
```

7. Make a modification to one of the tables.
 - a. Add a PRICE column to the TITLE table to record the purchase price of the video. The column should have a total length of eight digits and two decimal places. Verify your modifications.

```
ALTER TABLE title
ADD (price NUMBER(8,2));
DESCRIBE title
```

Practice 14 Solutions (continued)

- b. Create a script named lab14_7b.sql that contains update statements that update each video with a price according to the following list. Run the commands in the script.

Note: Have the TITLE_ID numbers available for this exercise.

Title	Price
Willie and Christmas Too	25
Alien Again	35
The Glob	35
My Day Off	35
Miracles on Ice	30
Soda Gang	35
Interstellar Wars	29

```
SET ECHO OFF
SET VERIFY OFF
DEFINE price=
DEFINE title_id=
UPDATE title
SET price = &price
WHERE title_id = &title_id;
SET VERIFY OFF
SET ECHO OFF
```

- c. Ensure that in the future all titles contain a price value. Verify the constraint.

```
ALTER TABLE title
MODIFY (price CONSTRAINT title_price_nn NOT NULL);
SELECT constraint_name, constraint_type,
       search_condition
FROM   user_constraints
WHERE  table_name = 'TITLE';
```

Practice 14 Solutions (continued)

8. Create a report titled Customer History Report. This report contains each customer's history of renting videos. Be sure to include the customer name, movie rented, dates of the rental, and duration of rentals. Total the number of rentals for all customers for the reporting period. Save the commands that generate the report in a script file named lab14_8.sql.

```
SET ECHO OFF
SET VERIFY OFF
TTITLE 'Customer History Report'
BREAK ON member SKIP 1 ON REPORT
SELECT      m.first_name||' '||m.last_name MEMBER, t.title,
            r.book_date, r.act_ret_date - r.book_date
DURATION
FROM        member m, title t, rental r
WHERE       r.member_id = m.member_id
AND         r.title_id = t.title_id
ORDER BY member;

CLEAR BREAK
TTITLE OFF
SET VERIFY ON
SET ECHO ON
```

Practice 15 Solutions

1. List the department IDs for departments that do not contain the job ID ST_CLERK, using SET operators.

```
SELECT department_id
FROM departments
MINUS
SELECT department_id
FROM employees
WHERE job_id = 'ST_CLERK';
```

2. Display the country ID and the name of the countries that have no departments located in them, using SET operators.

```
SELECT country_id,country_name
FROM countries
MINUS
SELECT l.country_id,c.country_name
FROM locations l, countries c
WHERE l.country_id = c.country_id;
```

3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID, using SET operators.

```
COLUMN dummy NOPRINT
SELECT job_id, department_id, 'x' dummy
FROM employees
WHERE department_id = 10
UNION
SELECT job_id, department_id, 'y'
FROM employees
WHERE department_id = 50
UNION
SELECT job_id, department_id, 'z'
FROM employees
WHERE department_id = 20
ORDER BY 3;
COLUMN dummy PRINT
```

Practice 15 Solutions (continued)

4. List the employee IDs and job IDs of those employees who currently have the job title that they held before beginning their tenure with the company.

```
SELECT    employee_id,job_id
FROM      employees
INTERSECT
SELECT    employee_id,job_id
FROM      job_history;
```

5. Write a compound query that lists the following:
- Last names and department ID of all the employees from the EMPLOYEES table, regardless of whether or not they belong to any department
 - Department ID and department name of all the departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them

```
SELECT last_name,department_id,TO_CHAR(null)
FROM    employees
UNION
SELECT TO_CHAR(null),department_id,department_name
FROM    departments;
```

Practice 16 Solutions

1. Alter the session to set the NLS_DATE_FORMAT to DD-MON-YYYY HH24:MI:SS.

```
ALTER SESSION SET NLS_DATE_FORMAT =  
'DD-MON-YYYY HH24:MI:SS';
```

2. a. Write queries to display the time zone offsets (TZ_OFFSET) for the following time zones.

US/Pacific-New

```
SELECT TZ_OFFSET ('US/Pacific-New') from dual;
```

Singapore

```
SELECT TZ_OFFSET ('Singapore') from dual;
```

Egypt

```
SELECT TZ_OFFSET ('Egypt') from dual;
```

- b. Alter the session to set the TIME_ZONE parameter value to the time zone offset of US/Pacific-New.

```
ALTER SESSION SET TIME_ZONE = '-7:00';
```

- c. Display the CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP for this session.

Note: The output might be different based on the date when the command is executed.

```
SELECT CURRENT_DATE, CURRENT_TIMESTAMP, LOCALTIMESTAMP  
FROM DUAL;
```

- d. Alter the session to set the TIME_ZONE parameter value to the time zone offset of Singapore.

```
ALTER SESSION SET TIME_ZONE = '+8:00';
```

- e. Display the CURRENT_DATE, CURRENT_TIMESTAMP, LOCALTIMESTAMP for this session.

Note: The output might be different, based on the date when the command is executed.

```
SELECT CURRENT_DATE, CURRENT_TIMESTAMP, LOCALTIMESTAMP  
FROM DUAL;
```

3. Write a query to display the DBTIMEZONE and SESSIONTIMEZONE.

```
SELECT DBTIMEZONE,SESSIONTIMEZONE  
FROM DUAL;
```

Practice 16 Solutions (continued)

4. Write a query to extract the YEAR from HIRE_DATE column of the EMPLOYEES table for those employees who work in department 80.

```
SELECT last_name, EXTRACT (YEAR FROM HIRE_DATE)
FROM employees
WHERE department_id = 80;
```

5. Alter the session to set the NLS_DATE_FORMAT to DD-MON-YYYY.

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY';
```

Practice 17 Solutions

1. Write a query to display the following for those employees whose manager ID is less than 120:

- Manager ID
- Job ID and total salary for every job ID for employees who report to the same manager
- Total salary of those managers
- Total salary of those managers, irrespective of the job IDs

```
SELECT manager_id,job_id,sum(salary)
FROM employees
WHERE manager_id < 120
GROUP BY ROLLUP(manager_id,job_id);
```

2. Observe the output from question 1. Write a query using the GROUPING function to determine whether the NULL values in the columns corresponding to the GROUP BY expressions are caused by the ROLLUP operation.

```
SELECT manager_id MGR ,job_id JOB,
sum(salary),GROUPING(manager_id),GROUPING(job_id)
FROM employees
WHERE manager_id < 120
GROUP BY ROLLUP(manager_id,job_id);
```

3. Write a query to display the following for those employees whose manager ID is less than 120 :

- Manager ID
- Job and total salaries for every job for employees who report to the same manager
- Total salary of those managers
- Cross-tabulation values to display the total salary for every job, irrespective of the manager
- Total salary irrespective of all job titles

```
SELECT manager_id, job_id, sum(salary)
FROM employees
WHERE manager_id < 120
GROUP BY CUBE(manager_id, job_id);
```


Practice 17 Solutions (continued)

4. Observe the output from question 3. Write a query using the GROUPING function to determine whether the NULL values in the columns corresponding to the GROUP BY expressions are caused by the CUBE operation.

```
SELECT manager_id MGR ,job_id JOB,
sum(salary),GROUPING(manager_id),GROUPING(job_id)
FROM employees
WHERE manager_id < 120
GROUP BY CUBE(manager_id,job_id);
```

5. Using GROUPING SETS, write a query to display the following groupings :
- department_id, manager_id, job_id
 - department_id, job_id
 - Manager_id, job_id

The query should calculate the sum of the salaries for each of these groups.

```
SELECT department_id, manager_id, job_id, SUM(salary)
FROM employees
GROUP BY
GROUPING SETS ((department_id, manager_id, job_id),
(department_id, job_id),(manager_id,job_id));
```

Practice 18 Solutions

1. Write a query to display the last name, department number, and salary of any employee whose department number and salary both match the department number and salary of any employee who earns a commission.

```
SELECT last_name, department_id, salary
FROM employees
WHERE (salary, department_id) IN
      (SELECT salary, department_id
       FROM employees
       WHERE commission_pct IS NOT NULL);
```

2. Display the last name, department name, and salary of any employee whose salary and commission match the salary and commission of any employee located in location ID1700.

```
SELECT e.last_name, d.department_name, e.salary
FROM employees e, departments d
WHERE e.department_id = d.department_id
AND (salary, NVL(commission_pct,0)) IN
     (SELECT salary, NVL(commission_pct,0)
      FROM employees e, departments d
      WHERE e.department_id = d.department_id
            AND d.location_id = 1700);
```

3. Create a query to display the last name, hire date, and salary for all employees who have the same salary and commission as Kochhar.

Note: Do not display Kochhar in the result set.

```
SELECT last_name, hire_date, salary
FROM employees
WHERE (salary, NVL(commission_pct,0)) IN
      (SELECT salary, NVL(commission_pct,0)
       FROM employees
       WHERE last_name = 'Kochhar')
AND last_name != 'Kochhar';
```

4. Create a query to display the employees who earn a salary that is higher than the salary of all of the sales managers (JOB_ID = 'SA_MAN'). Sort the results on salary from highest to lowest.

```
SELECT last_name, job_id, salary
FROM employees
WHERE salary > ALL
      (SELECT salary
       FROM employees
       WHERE job_id = 'SA_MAN')
ORDER BY salary DESC;
```

Practice 18 Solutions (continued)

5. Display the details of the employee ID, last name, and department ID of those employees who live in cities whose name begins with *T*.

```
SELECT employee_id, last_name, department_id
FROM employees
WHERE department_id IN (SELECT department_id
                        FROM departments
                        WHERE location_id IN
                        (SELECT location_id
                         FROM locations
                         WHERE city LIKE 'T%'));
```

6. Write a query to find all employees who earn more than the average salary in their departments. Display last name, salary, department ID, and the average salary for the department. Sort by average salary. Use aliases for the columns retrieved by the query as shown in the sample output.

```
SELECT e.last_name ename, e.salary salary,
       e.department_id deptno, AVG(a.salary) dept_avg
FROM employees e, employees a
WHERE e.department_id = a.department_id
AND e.salary > (SELECT AVG(salary)
                FROM employees
                WHERE department_id = e.department_id )
GROUP BY e.last_name, e.salary, e.department_id
ORDER BY AVG(a.salary);
```

7. Find all employees who are not supervisors.
a. First do this by using the NOT EXISTS operator.

```
SELECT outer.last_name
FROM employees outer
WHERE NOT EXISTS (SELECT 'X'
                  FROM employees inner
                  WHERE inner.manager_id =
                        outer.employee_id);
```

Practice 18 Solutions (continued)

- b. Can this be done by using the NOT IN operator? How, or why not?

```
SELECT outer.last_name
FROM   employees outer
WHERE  outer.employee_id
NOT IN (SELECT inner.manager_id
        FROM   employees inner);
```

This alternative solution is not a good one. The subquery picks up a NULL value, so the entire query returns no rows. The reason is that all conditions that compare a NULL value result in NULL. Whenever NULL values are likely to be part of the value set, *do not* use NOT IN as a substitute for NOT EXISTS.

8. Write a query to display the last names of the employees who earn less than the average salary in their departments.

```
SELECT last_name
FROM   employees outer
WHERE  outer.salary < (SELECT AVG(inner.salary)
                       FROM employees inner
                       WHERE inner.department_id
                             = outer.department_id);
```

9. Write a query to display the last names of employees who have one or more coworkers in their departments with later hire dates but higher salaries.

```
SELECT last_name
FROM   employees outer
WHERE EXISTS (SELECT 'X'
              FROM employees inner
              WHERE inner.department_id =
                    outer.department_id
              AND inner.hire_date > outer.hire_date
              AND inner.salary > outer.salary);
```

10. Write a query to display the employee ID, last names, and department names of all employees.

Note: Use a scalar subquery to retrieve the department name in the SELECT statement.

```
SELECT employee_id, last_name,
       (SELECT department_name
        FROM departments d
        WHERE e.department_id =
              d.department_id ) department
FROM employees e
ORDER BY department;
```

Practice 18 Solutions (continued)

11. Write a query to display the department names of those departments whose total salary cost is above one-eighth ($1/8$) of the total salary cost of the whole company. Use the WITH clause to write this query. Name the query SUMMARY.

```
WITH
summary AS (
    SELECT d.department_name, SUM(e.salary) AS dept_total
    FROM employees e, departments d
    WHERE e.department_id = d.department_id
    GROUP BY d.department_name)
SELECT department_name, dept_total
FROM summary
WHERE dept_total > (
    SELECT SUM(dept_total) * 1/8
    FROM summary )
ORDER BY dept_total DESC;
```

Practice 19 Solutions

1. Look at the following outputs. Are these outputs the result of a hierarchical query? Explain why or why not.

Exhibit 1: This is not a hierarchical query; the report simply has a descending sort on SALARY.

Exhibit 2: This is not a hierarchical query; there are two tables involved.

Exhibit 3: Yes, this is most definitely a hierarchical query as it displays the tree structure representing the management reporting line from the EMPLOYEES table.

2. Produce a report showing an organization chart for Mourgos's department. Print last names, salaries, and department IDs.

```
SELECT last_name, salary, department_id
FROM employees
START WITH last_name = 'Mourgos'
CONNECT BY PRIOR employee_id = manager_id;
```

3. Create a report that shows the hierarchy of the managers for the employee Lorentz. Display his immediate manager first.

```
SELECT last_name
FROM employees
WHERE last_name != 'Lorentz'
START WITH last_name = 'Lorentz'
CONNECT BY PRIOR manager_id = employee_id;
```

4. Create an indented report showing the management hierarchy starting from the employee whose LAST_NAME is Kochhar. Print the employee's last name, manager ID, and department ID. Give alias names to the columns as shown in the sample output.

```
COLUMN name FORMAT A20
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2, '_')
       name, manager_id mgr, department_id deptno
FROM employees
START WITH last_name = 'Kochhar'
CONNECT BY PRIOR employee_id = manager_id
/
COLUMN name CLEAR
```

Practice 19 Solutions (continued)

If you have time, complete the following exercises:

5. Produce a company organization chart that shows the management hierarchy. Start with the person at the top level, exclude all people with a job ID of IT_PROG, and exclude De Haan and those employees who report to De Haan.

```
SELECT last_name,employee_id, manager_id
FROM   employees
WHERE  job_id != 'IT_PROG'
START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id
AND last_name != 'De Haan';
```

Practice 20 Solutions

1. Run the `cre_sal_history.sql` script in the lab folder to create the `SAL_HISTORY` table.

```
@ \lab\cre_sal_history.sql
```

2. Display the structure of the `SAL_HISTORY` table.

```
DESC sal_history
```

3. Run the `cre_mgr_history.sql` script in the lab folder to create the `MGR_HISTORY` table.

```
@ \lab\cre_mgr_history.sql
```

4. Display the structure of the `MGR_HISTORY` table.

```
DESC mgr_history
```

5. Run the `cre_special_sal.sql` script in the lab folder to create the `SPECIAL_SAL` table.

```
@ \lab\cre_special_sal.sql
```

6. Display the structure of the `SPECIAL_SAL` table.

```
DESC special_sal
```

7. a. Write a query to do the following:

- Retrieve the details of the employee ID, hire date, salary, and manager ID of those employees whose employee ID is less than 125 from the `EMPLOYEES` table.
- If the salary is more than \$20,000, insert the details of employee ID and salary into the `SPECIAL_SAL` table.
- Insert the details of the employee ID, hire date, and salary into the `SAL_HISTORY` table.
- Insert the details of the employee ID, manager ID, and `SYSDATE` into the `MGR_HISTORY` table.

```
INSERT ALL
WHEN SAL > 20000 THEN
INTO special_sal VALUES (EMPID, SAL)
ELSE
INTO sal_history VALUES(EMPID,HIREDATE,SAL)
INTO mgr_history VALUES(EMPID,MGR,SAL)
SELECT employee_id EMPID, hire_date HIREDATE,
salary SAL, manager_id MGR
FROM employees
WHERE employee_id < 125;
```


Practice 20 Solutions (continued)

- b. Display the records from the SPECIAL_SAL table.

```
SELECT * FROM special_sal;
```

- c. Display the records from the SAL_HISTORY table.

```
SELECT * FROM sal_history;
```

- d. Display the records from the MGR_HISTORY table.

```
SELECT * FROM mgr_history;
```

8. a. Run the cre_sales_source_data.sql script in the lab folder to create the SALES_SOURCE_DATA table.

```
@ \lab\cre_sales_source_data.sql
```

- b. Run the ins_sales_source_data.sql script in the lab folder to insert records into the SALES_SOURCE_DATA table.

```
@ \lab\ins_sales_source_data.sql
```

- c. Display the structure of the SALES_SOURCE_DATA table.

```
DESC sales_source_data
```

- d. Display the records from the SALES_SOURCE_DATA table.

```
SELECT * FROM SALES_SOURCE_DATA;
```

- e. Run the cre_sales_info.sql script in the lab folder to create the SALES_INFO table.

```
@ \lab\cre_sales_info.sql
```

- f. Display the structure of the SALES_INFO table.

```
DESC sales_info
```

- g. Write a query to do the following:

- Retrieve the details of the employee ID, week ID, sales on Monday, sales on Tuesday, sales on Wednesday, sales on Thursday, and sales on Friday from the SALES_SOURCE_DATA table.
- Build a transformation such that each record retrieved from the SALES_SOURCE_DATA table is converted into multiple records for the SALES_INFO table.

Hint: Use a pivoting INSERT statement.

Practice 20 Solutions (continued)

```
INSERT ALL
INTO sales_info VALUES (employee_id, week_id, sales_MON)
INTO sales_info VALUES (employee_id, week_id, sales_TUE)
INTO sales_info VALUES (employee_id, week_id, sales_WED)
INTO sales_info VALUES (employee_id, week_id, sales_THUR)
INTO sales_info VALUES (employee_id, week_id, sales_FRI)
SELECT EMPLOYEE_ID, week_id, sales_MON, sales_TUE,
sales_WED, sales_THUR,sales_FRI FROM sales_source_data;
```

h. Display the records from the SALES_INFO table.

```
SELECT * FROM sales_info;
```

9. a. Create the DEPT_NAMED_INDEX table based on the following table instance chart. Name the index for the PRIMARY KEY column as DEPT_PK_IDX.

COLUMN Name	Deptno	Dname
Primary Key	Yes	
Data type	Number	VARCHAR2
Length	4	30

```
CREATE TABLE DEPT_NAMED_INDEX
(deptno NUMBER(4)
PRIMARY KEY USING INDEX
(CREATE INDEX dept_pk_idx ON
DEPT_NAMED_INDEX(deptno),
dname VARCHAR2(30));
```

- b. Query the USER_INDEXES table to display the INDEX_NAME for the DEPT_NAMED_INDEX table.

```
SELECT INDEX_NAME, TABLE_NAME
FROM USER_INDEXES
WHERE TABLE_NAME = 'DEPT_NAMED_INDEX';
```

Practice D Solutions

1. Write a script to describe and select the data from your tables. Use `CHR(10)` in the select list with the concatenation operator (`||`) to generate a line feed in your report. Save the output of the script into `my_file1.sql`. To save the file, select the `FILE` option for the output, and execute the code. Remember to save the file with a `.sql` extension. To execute the `my_file1.sql`, browse to locate the script, load the script, and execute the script.

```
SET PAGESIZE 0

SELECT 'DESC ' || table_name || CHR(10) ||
       'SELECT * FROM ' || table_name || ';'
FROM   user_tables
/
SET PAGESIZE 24
SET LINESIZE 100
```

2. Use SQL to generate SQL statements that revoke user privileges. Use the data dictionary views `USER_TAB_PRIVS_MADE` and `USER_COL_PRIVS_MADE`.
 - a. Execute the script `\lab\privs.sql` to grant privileges to the user `SYSTEM`.
 - b. Query the data dictionary views to check the privileges. In the sample output shown, note that the data in the `GRANTOR` column can vary depending on who the `GRANTOR` is. Also the last column that has been truncated is the `GRANTABLE` column.

```
COLUMN      grantee  FORMAT  A10
COLUMN      table_name  FORMAT  A10
COLUMN      column_name  FORMAT  A10
COLUMN      grantor  FORMAT  A10
COLUMN      privilege  FORMAT  A10
SELECT *
FROM      user_tab_privs_made
WHERE     grantee = 'SYSTEM';

SELECT *
FROM      user_col_privs_made
WHERE     grantee = 'SYSTEM';
```

Practice D Solutions (continued)

- c. Produce a script to revoke the privileges. Save the output of the script into `my_file2.sql`. To save the file, select the `FILE` option for the output, and execute the code. Remember to save the file with a `.sql` extension. To execute the `my_file2.sql`, browse to locate the script, load the script, and execute the script.

```
SET VERIFY OFF
SET PAGESIZE 0

SELECT      'REVOKE ' || privilege || ' ON ' ||
table_name || ' FROM system;'
FROM      user_tab_privs_made
WHERE     grantee = 'SYSTEM'
/

SELECT      DISTINCT      'REVOKE ' || privilege || ' ON ' ||
table_name || ' FROM system;'
FROM      user_col_privs_made
WHERE     grantee = 'SYSTEM'
/

SET VERIFY ON
SET PAGESIZE 24
```

B

Table Descriptions and Data

COUNTRIES Table

```
DESCRIBE countries
```

Name	Null?	Type
COUNTRY_ID	NOT NULL	CHAR(2)
COUNTRY_NAME		VARCHAR2(40)
REGION_ID		NUMBER

```
SELECT * FROM countries;
```

CO	COUNTRY_NAME	REGION_ID
CA	Canada	2
DE	Germany	1
UK	United Kingdom	1
US	United States of America	2

DEPARTMENTS Table

DESCRIBE departments

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

SELECT * FROM departments;

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

8 rows selected.

EMPLOYEES Table

DESCRIBE employees

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

SELECT * FROM employees;

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
100	Steven	King	SKING	515.123.4567	17-JUN-87
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00
174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94
206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94

20 rows selected.

EMPLOYEES Table (continued)

JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
AD_PRES	24000			90
AD_VP	17000		100	90
AD_VP	17000		100	90
IT_PROG	9000		102	60
IT_PROG	6000		103	60
IT_PROG	4200		103	60
ST_MAN	5800		100	50
ST_CLERK	3500		124	50
ST_CLERK	3100		124	50
ST_CLERK	2600		124	50
ST_CLERK	2500		124	50
SA_MAN	10500	.2	100	80
SA_REP	11000	.3	149	80
SA_REP	8600	.2	149	80
SA_REP	7000	.15	149	
AD_ASST	4400		101	10
MK_MAN	13000		100	20
MK_REP	6000		201	20
AC_MGR	12000		101	110
AC_ACCOUNT	8300		205	110

20 rows selected.

JOBS Table

DESCRIBE jobs

Name	Null?	Type
JOB_ID	NOT NULL	VARCHAR2(10)
JOB_TITLE	NOT NULL	VARCHAR2(35)
MIN_SALARY		NUMBER(6)
MAX_SALARY		NUMBER(6)

SELECT * FROM jobs;

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AD_PRES	President	20000	40000
AD_VP	Administration Vice President	15000	30000
AD_ASST	Administration Assistant	3000	6000
AC_MGR	Accounting Manager	8200	16000
AC_ACCOUNT	Public Accountant	4200	9000
SA_MAN	Sales Manager	10000	20000
SA_REP	Sales Representative	6000	12000
ST_MAN	Stock Manager	5500	8500
ST_CLERK	Stock Clerk	2000	5000
IT_PROG	Programmer	4000	10000
MK_MAN	Marketing Manager	9000	15000
MK_REP	Marketing Representative	4000	9000

12 rows selected.

JOB_GRADES Table

```
DESCRIBE job_grades
```

Name	Null?	Type
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

```
SELECT * FROM job_grades;
```

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

6 rows selected.

JOB_HISTORY Table

```
DESCRIBE job_history
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID		NUMBER(4)

```
SELECT * FROM job_history;
```

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
102	13-JAN-93	24-JUL-98	IT_PROG	60
101	21-SEP-89	27-OCT-93	AC_ACCOUNT	110
101	28-OCT-93	15-MAR-97	AC_MGR	110
201	17-FEB-96	19-DEC-99	MK_REP	20
114	24-MAR-98	31-DEC-99	ST_CLERK	50
122	01-JAN-99	31-DEC-99	ST_CLERK	50
200	17-SEP-87	17-JUN-93	AD_ASST	90
176	24-MAR-98	31-DEC-98	SA_REP	80
176	01-JAN-99	31-DEC-99	SA_MAN	80
200	01-JUL-94	31-DEC-98	AC_ACCOUNT	90

10 rows selected.

LOCATIONS Table

DESCRIBE locations

Name	Null?	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

SELECT * FROM locations;

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	CO
1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
1500	2011 Interiors Blvd	99236	South San Francisco	California	US
1700	2004 Charade Rd	98199	Seattle	Washington	US
1800	460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

REGIONS Table

```
DESCRIBE regions
```

Name	Null?	Type
REGION_ID	NOT NULL	NUMBER
REGION_NAME		VARCHAR2(25)

```
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

C

Using SQL*Plus

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this appendix, you should be able to do the following:

- **Log in to SQL*Plus**
- **Edit SQL commands**
- **Format output using SQL*Plus commands**
- **Interact with script files**

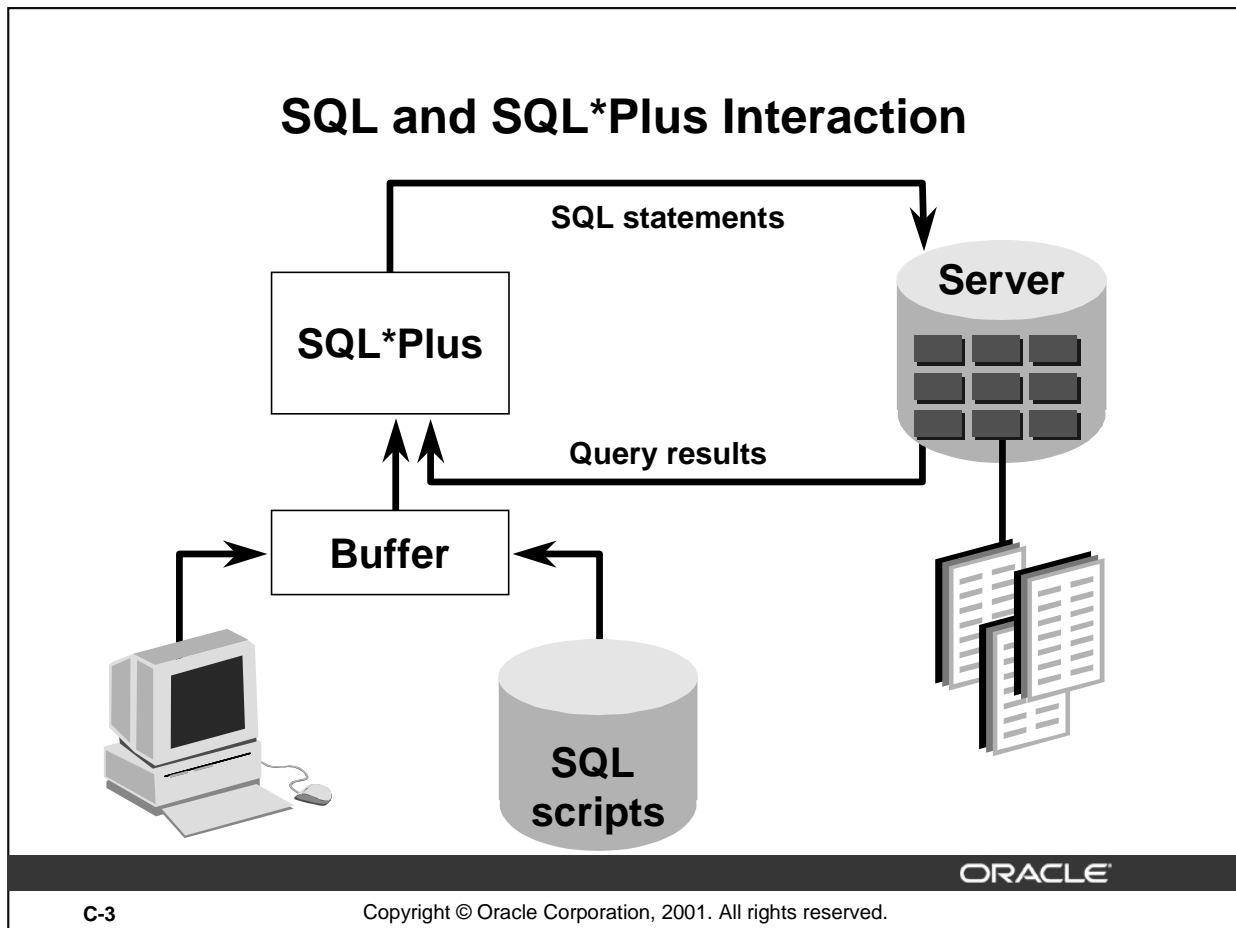
ORACLE

C-2

Copyright © Oracle Corporation, 2001. All rights reserved.

Lesson Aim

You may want to create `SELECT` statements that can be used again and again. This lesson also covers the use of SQL*Plus commands to execute SQL statements. You learn how to format output using SQL*Plus commands, edit SQL commands, and save scripts in SQL*Plus.



SQL and SQL*Plus

SQL is a command language for communication with the Oracle9i Server from any tool or application. Oracle SQL contains many extensions. When you enter a SQL statement, it is stored in a part of memory called the *SQL buffer* and remains there until you enter a new SQL statement.

SQL*Plus is an Oracle tool that recognizes and submits SQL statements to the Oracle9i Server for execution. It contains its own command language.

Features of SQL

- SQL can be used by a range of users, including those with little or no programming experience.
- It is a nonprocedural language.
- It reduces the amount of time required for creating and maintaining systems.
- It is an English-like language.

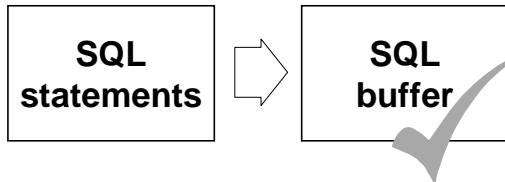
Features of SQL*Plus

- SQL*Plus accepts ad hoc entry of statements.
- It accepts SQL input from files.
- It provides a line editor for modifying SQL statements.
- It controls environmental settings.
- It formats query results into basic reports.
- It accesses local and remote databases.

SQL Statements versus SQL*Plus Commands

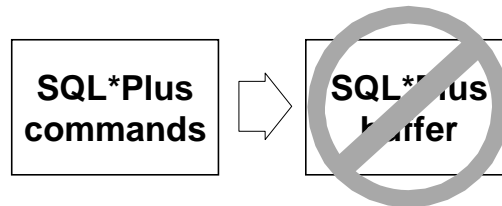
SQL

- A language
- ANSI standard
- Keywords cannot be abbreviated
- Statements manipulate data and table definitions in the database



SQL*Plus

- An environment
- Oracle proprietary
- Keywords can be abbreviated
- Commands do not allow manipulation of values in the database



ORACLE

C-4

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL and SQL*Plus (continued)

The following table compares SQL and SQL*Plus:

SQL	SQL*Plus
Is a language for communicating with the Oracle server to access data	Recognizes SQL statements and sends them to the server
Is based on American National Standards Institute (ANSI) standard SQL	Is the Oracle proprietary interface for executing SQL statements
Manipulates data and table definitions in the database	Does not allow manipulation of values in the database
Is entered into the SQL buffer on one or more lines	Is entered one line at a time, not stored in the SQL buffer
Does not have a continuation character	Uses a dash (-) as a continuation character if the command is longer than one line
Cannot be abbreviated	Can be abbreviated
Uses a termination character to execute commands immediately	Does not require termination characters; executes commands immediately
Uses functions to perform some formatting	Uses commands to format data

Overview of SQL*Plus

- **Log in to SQL*Plus.**
- **Describe the table structure.**
- **Edit your SQL statement.**
- **Execute SQL from SQL*Plus.**
- **Save SQL statements to files and append SQL statements to files.**
- **Execute saved files.**
- **Load commands from file to buffer to edit.**

ORACLE

C-5

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL*Plus

SQL*Plus is an environment in which you can do the following:

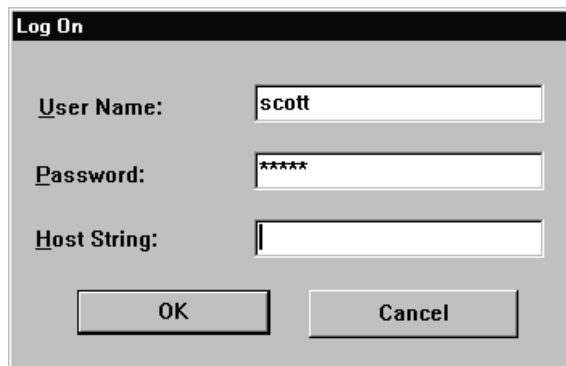
- Execute SQL statements to retrieve, modify, add, and remove data from the database
- Format, perform calculations on, store, and print query results in the form of reports
- Create script files to store SQL statements for repetitive use in the future

SQL*Plus commands can be divided into the following main categories:

Category	Purpose
Environment	Affect the general behavior of SQL statements for the session
Format	Format query results
File manipulation	Save, load, and run script files
Execution	Send SQL statements from SQL buffer to the Oracle server
Edit	Modify SQL statements in the buffer
Interaction	Create and pass variables to SQL statements, print variable values, and print messages to the screen
Miscellaneous	Connect to the database, manipulate the SQL*Plus environment, and display column definitions

Logging In to SQL*Plus

- From a Windows environment:



- From a command line:

```
sqlplus [username[/password  
        [@database]]]
```

ORACLE

C-6

Copyright © Oracle Corporation, 2001. All rights reserved.

Logging In to SQL*Plus

How you invoke SQL*Plus depends on which type of operating system or Windows environment you are running.

To log in through a Windows environment:

1. Select Start > Programs > Oracle for Windows NT > SQL*Plus.
2. Fill in the username, password, and database name.

To log in through a command line environment:

1. Log on to your machine.
2. Enter the SQL*Plus command shown in the slide.

In the syntax:

username your database username.
password your database password (if you enter your password here, it is visible.)
@database the database connect string.

Note: To ensure the integrity of your password, do not enter it at the operating system prompt. Instead, enter only your username. Enter your password at the Password prompt.

After you log in to SQL*Plus, you see the following message (if you are using SQL*Plus version 9i):

```
SQL*Plus: Release 9.0.1.0.0 - Development on Tue Jan 9 08:44:28 2001  
  (c) Copyright 2000 Oracle Corporation. All rights reserved.
```

Displaying Table Structure

Use the SQL*Plus DESCRIBE command to display the structure of a table.

```
DESC[RIBE] tablename
```

ORACLE

C-7

Copyright © Oracle Corporation, 2001. All rights reserved.

Displaying Table Structure

In SQL*Plus you can display the structure of a table using the DESCRIBE command. The result of the command is a display of column names and data types as well as an indication if a column must contain data.

In the syntax:

tablename the name of any existing table, view, or synonym that is accessible to the user

To describe the JOB_GRADES table, use this command:

```
SQL> DESCRIBE job_grades
Name                               Null?    Type
-----
GRADE_LEVEL                        VARCHA2 ( 3 )
LOWEST_SAL                          NUMBER
HIGHEST_SAL                         NUMBER
```

Displaying Table Structure

```
SQL> DESCRIBE departments
```

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

ORACLE

C-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Displaying Table Structure (continued)

The example in the slide displays the information about the structure of the DEPARTMENTS table.

In the result:

- Null?** specifies whether a column *must* contain data; NOT NULL indicates that a column must contain data
- Type** displays the data type for a column

The following table describes the data types:

Data type	Description
NUMBER(<i>p</i> , <i>s</i>)	Number value that has a maximum number of digits <i>p</i> , the number of digits to the right of the decimal point <i>s</i>
VARCHAR2(<i>s</i>)	Variable-length character value of maximum size <i>s</i>
DATE	Date and time value between January 1, 4712 B.C., and December 31, 9999 A.D.
CHAR(<i>s</i>)	Fixed-length character value of size <i>s</i>

SQL*Plus Editing Commands

- **A[PPEND] *text***
- **C[HANGE] / *old* / *new***
- **C[HANGE] / *text* /**
- **CL[EAR] BUFF[ER]**
- **DEL**
- **DEL *n***
- **DEL *m n***

ORACLE

C-9

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL*Plus Editing Commands

SQL*Plus commands are entered one line at a time and are not stored in the SQL buffer.

Command	Description
A[PPEND] <i>text</i>	Adds text to the end of the current line
C[HANGE] / <i>old</i> / <i>new</i>	Changes <i>old</i> text to <i>new</i> in the current line
C[HANGE] / <i>text</i> /	Deletes <i>text</i> from the current line
CL[EAR] BUFF[ER]	Deletes all lines from the SQL buffer
DEL	Deletes current line
DEL <i>n</i>	Deletes line <i>n</i>
DEL <i>m n</i>	Deletes lines <i>m</i> to <i>n</i> inclusive

Guidelines

- If you press [Enter] before completing a command, SQL*Plus prompts you with a line number.
- You terminate the SQL buffer either by entering one of the terminator characters (semicolon or slash) or by pressing [Enter] twice. The SQL prompt then appears.

SQL*Plus Editing Commands

- I[NPUT]
- I[NPUT] *text*
- L[IST]
- L[IST] *n*
- L[IST] *m n*
- R[UN]
- *n*
- *n text*
- 0 *text*

ORACLE

C-10

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL*Plus Editing Commands (continued)

Command	Description
I[NPUT]	Inserts an indefinite number of lines
I[NPUT] <i>text</i>	Inserts a line consisting of <i>text</i>
L[IST]	Lists all lines in the SQL buffer
L[IST] <i>n</i>	Lists one line (specified by <i>n</i>)
L[IST] <i>m n</i>	Lists a range of lines (<i>m</i> to <i>n</i>) inclusive
R[UN]	Displays and runs the current SQL statement in the buffer
<i>n</i>	Specifies the line to make the current line
<i>n text</i>	Replaces line <i>n</i> with <i>text</i>
0 <i>text</i>	Inserts a line before line 1

Note: You can enter only one SQL*Plus command per SQL prompt. SQL*Plus commands are not stored in the buffer. To continue a SQL*Plus command on the next line, end the first line with a hyphen (-).

Using LIST, n, and APPEND

```
SQL> LIST
```

```
1  SELECT last_name  
2* FROM   employees
```

```
SQL> 1
```

```
1* SELECT last_name
```

```
SQL> A , job_id
```

```
1* SELECT last_name, job_id
```

```
SQL> L
```

```
1  SELECT last_name, job_id  
2* FROM   employees
```

ORACLE

C-11

Copyright © Oracle Corporation, 2001. All rights reserved.

Using LIST, n, and APPEND

- Use the L[IST] command to display the contents of the SQL buffer. The * beside line 2 in the buffer indicates that line 2 is the current line. Any edits that you made apply to the current line.
- Change the number of the current line by entering the number of the line you want to edit. The new current line is displayed.
- Use the A[PPEND] command to add text to the current line. The newly edited line is displayed. Verify the new contents of the buffer by using the LIST command.

Note: Many SQL*Plus commands including LIST and APPEND can be abbreviated to just their first letter. LIST can be abbreviated to L, APPEND can be abbreviated to A.

Using the CHANGE Command

```
SQL> L
```

```
1* SELECT * from employees
```

```
SQL> c/employees/departments
```

```
1* SELECT * from departments
```

```
SQL> L
```

```
1* SELECT * from departments
```

ORACLE

C-12

Copyright © Oracle Corporation, 2001. All rights reserved.

Using the CHANGE Command

- Use L[IST] to display the contents of the buffer.
- Use the C[HANGE] command to alter the contents of the current line in the SQL buffer. In this case, replace the employees table with the departments table. The new current line is displayed.
- Use the L[IST] command to verify the new contents of the buffer.

SQL*Plus File Commands

- **SAVE *filename***
- **GET *filename***
- **START *filename***
- **@ *filename***
- **EDIT *filename***
- **SPOOL *filename***
- **EXIT**

ORACLE

C-13

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL*Plus File Commands

SQL statements communicate with the Oracle server. SQL*Plus commands control the environment, format query results, and manage files. You can use the commands described in the following table:

Command	Description
SAV[E] <i>filename</i> [.ext] [REP[LACE]APP[END]]	Saves current contents of SQL buffer to a file. Use APPEND to add to an existing file; use REPLACE to overwrite an existing file. The default extension is .sql.
GET <i>filename</i> [.ext]	Writes the contents of a previously saved file to the SQL buffer. The default extension for the filename is .sql.
STA[RT] <i>filename</i> [.ext]	Runs a previously saved command file
@ <i>filename</i>	Runs a previously saved command file (same as START)
ED[IT]	Invokes the editor and saves the buffer contents to a file named <code>afiedt.buf</code>
ED[IT] [<i>filename</i> [.ext]]	Invokes the editor to edit contents of a saved file
SPO[OL] [<i>filename</i> [.ext]] OFF OUT]	Stores query results in a file. OFF closes the spool file. OUT closes the spool file and sends the file results to the system printer.
EXIT	Leaves SQL*Plus

Using the SAVE and START Commands

```
SQL> L
  1  SELECT last_name, manager_id, department_id
  2* FROM   employees
SQL> SAVE my_query
```

```
Created file my_query
```

```
SQL> START my_query
```

```
LAST_NAME                MANAGER_ID DEPARTMENT_ID
-----
King                      100          90
Kochhar                   100          90
...
20 rows selected.
```

ORACLE

C-14

Copyright © Oracle Corporation, 2001. All rights reserved.

SAVE

Use the `SAVE` command to store the current contents of the buffer in a file. In this way, you can store frequently used scripts for use in the future.

START

Use the `START` command to run a script in SQL*Plus.

EDIT

Use the `EDIT` command to edit an existing script. This opens an editor with the script file in it. When you have made the changes, exit the editor to return to the SQL*Plus command line.

Summary

Use SQL*Plus as an environment to:

- **Execute SQL statements**
- **Edit SQL statements**
- **Format output**
- **Interact with script files**

ORACLE

C-15

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

SQL*Plus is an execution environment that you can use to send SQL commands to the database server and to edit and save SQL commands. You can execute commands from the SQL prompt or from a script file.

D

Writing Advanced Scripts

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this appendix, you should be able to do the following:

- Describe the types of problems that are solved by using SQL to generate SQL
- Write a script that generates a script of **DROP TABLE** statements
- Write a script that generates a script of **INSERT INTO** statements

ORACLE

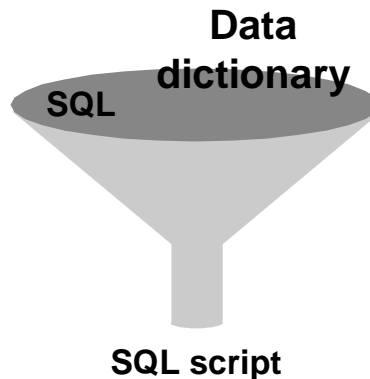
D-2

Copyright © Oracle Corporation, 2001. All rights reserved.

Lesson Aim

In this appendix, you learn how to write a SQL script to generates a SQL script.

Using SQL to Generate SQL



- **SQL can be used to generate scripts in SQL**
- **The data dictionary**
 - **Is a collection of tables and views that contain database information**
 - **Is created and maintained by the Oracle server**

ORACLE

D-3

Copyright © Oracle Corporation, 2001. All rights reserved.

Using SQL to Generate SQL

SQL can be a powerful tool to generate other SQL statements. In most cases this involves writing a script file. You can use SQL from SQL to:

- Avoid repetitive coding
- Access information from the data dictionary
- Drop or re-create database objects
- Generate dynamic predicates that contain run-time parameters

The examples used in this lesson involve selecting information from the data dictionary. The data dictionary is a collection of tables and views that contain information about the database. This collection is created and maintained by the Oracle Server. All data dictionary tables are owned by the SYS user. Information stored in the data dictionary includes names of the Oracle Server users, privileges granted to users, database object names, table constraints, and audition information. There are four categories of data dictionary views. Each category has a distinct prefix that reflects its intended use.

Prefix	Description
USER_	Contains details of objects owned by the user
ALL_	Contains details of objects to which the user has been granted access rights, in addition to objects owned by the user
DBA_	Contains details of users with DBA privileges to access any object in the database
V\$_	Stored information about database server performance and locking; available only to the DBA

Creating a Basic Script

```
SELECT 'CREATE TABLE ' || table_name || '_test '
      || 'AS SELECT * FROM ' || table_name
      || ' WHERE 1=2;'
      AS "Create Table Script"
FROM   user_tables;
```

Create Table Script
CREATE TABLE COUNTRIES_test AS SELECT * FROM COUNTRIES WHERE 1=2;
CREATE TABLE DEPARTMENTS_test AS SELECT * FROM DEPARTMENTS WHERE 1=2;
CREATE TABLE EMPLOYEES_test AS SELECT * FROM EMPLOYEES WHERE 1=2;
CREATE TABLE JOBS_test AS SELECT * FROM JOBS WHERE 1=2;
CREATE TABLE JOB_GRADES_test AS SELECT * FROM JOB_GRADES WHERE 1=2;
CREATE TABLE JOB_HISTORY_test AS SELECT * FROM JOB_HISTORY WHERE 1=2;
CREATE TABLE LOCATIONS_test AS SELECT * FROM LOCATIONS WHERE 1=2;
CREATE TABLE REGIONS_test AS SELECT * FROM REGIONS WHERE 1=2;

8 rows selected.

ORACLE

D-4

Copyright © Oracle Corporation, 2001. All rights reserved.

A Basic Script

The example in the slide produces a report with `CREATE TABLE` statements from every table you own. Each `CREATE TABLE` statement produced in the report includes the syntax to create a table using the table name with a suffix of `_test` and having only the structure of the corresponding existing table. The old table name is obtained from the `TABLE_NAME` column of the data dictionary view `USER_TABLES`.

The next step is to enhance the report to automate the process.

Note: You can query the data dictionary tables to view various database objects that you own. The data dictionary views frequently used include:

- `USER_TABLES`: Displays description of the user's own tables
- `USER_OBJECTS`: Displays all the objects owned by the user
- `USER_TAB_PRIVS_MADE`: Displays all grants on objects owned by the user
- `USER_COL_PRIVS_MADE`: Displays all grants on columns of objects owned by the user

Controlling the Environment

```
SET ECHO OFF  
SET FEEDBACK OFF  
SET PAGESIZE 0
```

← Set system variables
to appropriate values.

```
SPOOL dropem.sql
```

```
SQL STATEMENT
```

```
SPOOL OFF
```

```
SET FEEDBACK ON  
SET PAGESIZE 24  
SET ECHO ON
```

← Set system variables
back to the default
value.

ORACLE

D-5

Copyright © Oracle Corporation, 2001. All rights reserved.

Controlling the Environment

In order to execute the SQL statements that are generated, you must capture them in a spool file that can then be run. You must also plan to clean up the output that is generated and make sure that you suppress elements such as headings, feedback messages, top titles, and so on. You can accomplish all of this by using *iSQL*Plus* commands.

The Complete Picture

```
SET ECHO OFF
SET FEEDBACK OFF
SET PAGESIZE 0

SELECT 'DROP TABLE ' || object_name || ';'
FROM   user_objects
WHERE  object_type = 'TABLE'
/

SET FEEDBACK ON
SET PAGESIZE 24
SET ECHO ON
```

ORACLE

D-6

Copyright © Oracle Corporation, 2001. All rights reserved.

The Complete Picture

The output of the command on the slide is saved into a file called `dropem.sql` using the File Output option in *iSQL*Plus*. This file contains the following data. This file can now be started from the *iSQL*Plus* by locating the script file, loading it, and executing it.

'DROPTABLE' OBJECT_NAME ';'
DROP TABLE COUNTRIES;
DROP TABLE DEPARTMENTS;
DROP TABLE EMPLOYEES;
DROP TABLE JOBS;
DROP TABLE JOB_GRADES;
DROP TABLE JOB_HISTORY;
DROP TABLE LOCATIONS;
DROP TABLE REGIONS;

Note: By default, files are spooled into the `ORACLE_HOME\ORANT\BIN` folder in Windows NT.

Dumping the Contents of a Table to a File

```
SET HEADING OFF ECHO OFF FEEDBACK OFF
SET PAGESIZE 0

SELECT
  'INSERT INTO departments_test VALUES
  (' || department_id || ', ''' || department_name ||
  ''', ''' || location_id || ''');'
  AS "Insert Statements Script"
FROM   departments
/

SET PAGESIZE 24
SET HEADING ON ECHO ON FEEDBACK ON
```

ORACLE

D-7

Copyright © Oracle Corporation, 2001. All rights reserved.

Dumping Table Contents to a File

Sometimes it is useful to have the values for the rows of a table in a text file in the format of an `INSERT INTO VALUES` statement. This script can be run to populate the table, in case the table has been dropped accidentally.

The example in the slide produces `INSERT` statements for the `DEPARTMENTS_TEST` table, captured in the `data.sql` file using the File Output option in *iSQL*Plus*.

The contents of the `data.sql` script file are as follows:

```
INSERT INTO departments_test VALUES
  (10, 'Administration', 1700);
INSERT INTO departments_test VALUES
  (20, 'Marketing', 1800);
INSERT INTO departments_test VALUES
  (50, 'Shipping', 1500);
INSERT INTO departments_test VALUES
  (60, 'IT', 1400);
...
```

Dumping the Contents of a Table to a File

Source	Result
<code>'''x'''</code>	<code>'x'</code>
<code>''''</code>	<code>'</code>
<code>'''' department_name ''''</code>	<code>'Administration'</code>
<code>''', '''</code>	<code>','</code>
<code>''') ;'</code>	<code>');</code>

ORACLE

D-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Dumping Table Contents to a File (continued)

You may have noticed the large number of single quotes in the slide on the previous page. A set of four single quotes produces one single quote in the final statement. Also remember that character and date values must be surrounded by quotes.

Within a string, to display one single quote, you need to prefix it with another single quote. For example, in the fifth example in the slide, the surrounding quotes are for the entire string. The second quote acts as a prefix to display the third quote. Thus the result is one single quote followed by the parenthesis followed by the semicolon.

Generating a Dynamic Predicate

```
COLUMN my_col NEW_VALUE dyn_where_clause

SELECT DECODE('&&deptno', null,
DECODE ('&&hiredate', null, ' ',
'WHERE hire_date=TO_DATE('' || '&&hiredate'', 'DD-MON-YYYY'')),
DECODE ('&&hiredate', null,
'WHERE department_id = ' || '&&deptno',
'WHERE department_id = ' || '&&deptno' ||
' AND hire_date = TO_DATE('' || '&&hiredate'', 'DD-MON-YYYY''))
AS my_col FROM dual;
```

```
SELECT last_name FROM employees &dyn_where_clause;
```

```
;
```

ORACLE

D-9

Copyright © Oracle Corporation, 2001. All rights reserved.

Generating a Dynamic Predicate

The example in the slide generates a SELECT statement that retrieves data of all employees in a department who were hired on a specific day. The script generates the WHERE clause dynamically.

Note: Once the user variable is in place, you need to use the UNDEFINE command to delete it.

The first SELECT statement prompts you to enter the department number. If you do not enter any department number, the department number is treated as null by the DECODE function, and the user is then prompted for the hire date. If you do not enter any hire date, the hire date is treated as null by the DECODE function and the dynamic WHERE clause that is generated is also a null, which causes the second SELECT statement to retrieve all rows from the EMPLOYEES table.

Note: The NEW_V[ALUE] variable specifies a variable to hold a column value. You can reference the variable in TTITLE commands. Use NEW_VALUE to display column values or the date in the top title. You must include the column in a BREAK command with the SKIP PAGE action. The variable name cannot contain a pound sign (#). NEW_VALUE is useful for master/detail reports in which there is a new master record for each page.

Generating a Dynamic Predicate (continued)

Note: Here, the hire date must be entered in DD-MON-YYYY format.

The SELECT statement in the previous slide can be interpreted as follows:

```
IF (<<deptno>> is not entered) THEN
  IF (<<hiredate>> is not entered) THEN
    return empty string
  ELSE
    return the string 'WHERE hire_date = TO_DATE('<<hiredate>>', 'DD-MON-YYYY)'
```

```
ELSE
  IF (<<hiredate>> is not entered) THEN
    return the string 'WHERE department_id = <<deptno>> entered'
```

```
ELSE
  return the string 'WHERE department_id = <<deptno>> entered
    AND hire_date = TO_DATE('<<hiredate>>', 'DD-MON-YYYY)'
```

```
END IF
```

The returned string becomes the value of the variable DYN_WHERE_CLAUSE, that will be used in the second SELECT statement.

When the first example on the slide is executed, the user is prompted for the values for DEPTNO and HIREDATE:



Define Substitution Variables

"deptno"	<input type="text" value="10"/>
"hiredate"	<input type="text" value="17-SEP-1987"/>

The following value for MY_COL is generated:

MY_COL
WHERE department_id = 10AND hire_date = TO_DATE('17-SEP-1987','DD-MON-YYYY)

When the second example on the slide is executed, the following output is generated:

LAST_NAME
Whalen

Summary

In this appendix, you should have learned the following:

- **You can write a SQL script to generate another SQL script.**
- **Script files often use the data dictionary.**
- **You can capture the output in a file.**

ORACLE

D-11

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

SQL can be used to generate SQL scripts. These scripts can be used to avoid repetitive coding, drop or re-create objects, get help from the data dictionary, and generate dynamic predicates that contain run-time parameters.

*iSQL*Plus* commands can be used to capture the reports generated by the SQL statements and clean up the output that is generated, such as suppressing headings, feedback messages, and so on.

Practice D Overview

This practice covers the following topics:

- **Writing a script to describe and select the data from your tables**
- **Writing a script to revoke user privileges**

ORACLE

D-12

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice D Overview

In this practice, you gain practical experience in writing SQL to generate SQL.

Practice D

1. Write a script to describe and select the data from your tables. Use `CHR(10)` in the select list with the concatenation operator (`||`) to generate a line feed in your report. Save the output of the script into `my_file1.sql`. To save the file, select `File` option for the output and execute the code. Remember to save the file with a `.sql` extension. To execute the `my_file1.sql`, browse to locate the script, load the script, and execute the script.
2. Use SQL to generate SQL statements that revoke user privileges. Use the data dictionary views `USER_TAB_PRIVS_MADE` and `USER_COL_PRIVS_MADE`.
 - a. Execute the script `\Lab\privs.sql` to grant privileges to the user `SYSTEM`.
 - b. Query the data dictionary views to check the privileges. In the sample output shown, note that the data in the `GRANTOR` column can vary depending on who the `GRANTOR` is. Also the last column that has been truncated is the `GRANTABLE` column.

GRANTEE	TABLE_NAME	GRANTOR	PRIVILEGE	GRA	HIE
SYSTEM	DEPARTMENT S	SQL2	ALTER	NO	NO
SYSTEM	DEPARTMENT S	SQL2	DELETE	NO	NO
SYSTEM	DEPARTMENT S	SQL2	INDEX	NO	NO
SYSTEM	DEPARTMENT S	SQL2	INSERT	NO	NO
SYSTEM	DEPARTMENT S	SQL2	SELECT	NO	NO
SYSTEM	DEPARTMENT S	SQL2	UPDATE	NO	NO
SYSTEM	DEPARTMENT S	SQL2	REFERENCES	NO	NO
SYSTEM	DEPARTMENT S	SQL2	ON COMMIT REFRESH	NO	NO
SYSTEM	DEPARTMENT S	SQL2	QUERY REWR ITE	NO	NO
SYSTEM	DEPARTMENT S	SQL2	DEBUG	NO	NO

10 rows selected.

GRANTEE	TABLE_NAME	COLUMN_NAM	GRANTOR	PRIVILEGE	GRA
SYSTEM	EMPLOYEES	JOB_ID	SQL2	UPDATE	NO
SYSTEM	EMPLOYEES	SALARY	SQL2	UPDATE	NO

Practice D (continued)

- c. Produce a script to revoke the privileges. Save the output of the script into `my_file2.sql`. To save the file, select the `File` option for the output, and execute the code. Remember to save the file with a `.sql` extension. To execute the `my_file2.sql`, browse to locate the script, load the script, and execute the script.

'REVOKE' PRIVILEGE 'ON' TABLE_NAME 'FROM SYSTEM;'
REVOKE ALTER ON DEPARTMENTS FROM system;
REVOKE DELETE ON DEPARTMENTS FROM system;
REVOKE INDEX ON DEPARTMENTS FROM system;
REVOKE INSERT ON DEPARTMENTS FROM system;
REVOKE SELECT ON DEPARTMENTS FROM system;
REVOKE UPDATE ON DEPARTMENTS FROM system;
REVOKE REFERENCES ON DEPARTMENTS FROM system;
REVOKE ON COMMIT REFRESH ON DEPARTMENTS FROM system;
REVOKE QUERY REWRITE ON DEPARTMENTS FROM system;
REVOKE DEBUG ON DEPARTMENTS FROM system;

10 rows selected.

'REVOKE' PRIVILEGE 'ON' TABLE_NAME 'FROM SYSTEM;'
REVOKE UPDATE ON EMPLOYEES FROM system;

Oracle Architectural Components

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

After completing this appendix, you should be able to do the following:

- **Describe the Oracle Server architecture and its main components**
- **List the structures involved in connecting a user to an Oracle instance**
- **List the stages in processing:**
 - **Queries**
 - **DML statements**
 - **Commits**

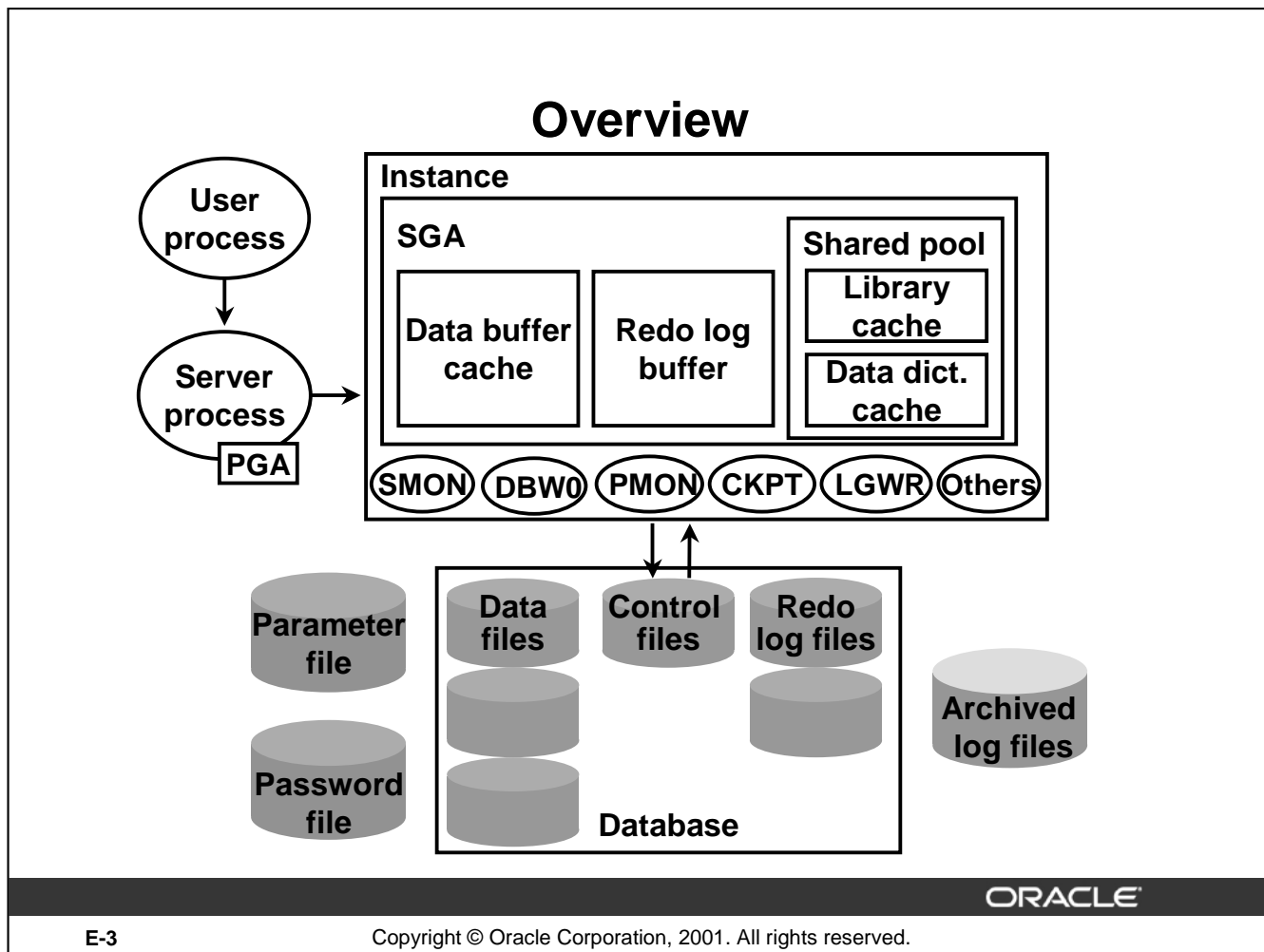
ORACLE

E-2

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

This appendix introduces Oracle Server architecture by describing the files, processes, and memory structures involved in establishing a database connection and executing a SQL command.



Overview

The Oracle Server is an object relational database management system that provides an open, comprehensive, integrated approach to information management.

Primary Components

There are several processes, memory structures, and files in an Oracle Server; however, not all of them are used when processing a SQL statement. Some are used to improve the performance of the database, ensure that the database can be recovered in the event of a software or hardware error, or perform other tasks necessary to maintain the database. The Oracle Server consists of an Oracle instance and an Oracle database.

Oracle Instance

An Oracle instance is the combination of the background processes and memory structures. The instance must be started to access the data in the database. Every time an instance is started, a system global area (SGA) is allocated and Oracle background processes are started. The SGA is a memory area used to store database information that is shared by database processes.

Background processes perform functions on behalf of the invoking process. They consolidate functions that would otherwise be handled by multiple Oracle programs running for each user. The background processes perform I/O and monitor other Oracle processes to provide increased parallelism for better performance and reliability.

Primary Components (continued)

Other Processes

The user process is the application program that originates SQL statements. The server process executes the SQL statements sent from the user process.

Database Files

Database files are operating system files that provide the actual physical storage for database information. The database files are used to ensure that the data is kept consistent and can be recovered in the event of a failure of the instance.

Other Files

Nondatabase files are used to configure the instance, authenticate privileged users, and recover the database in the event of a disk failure.

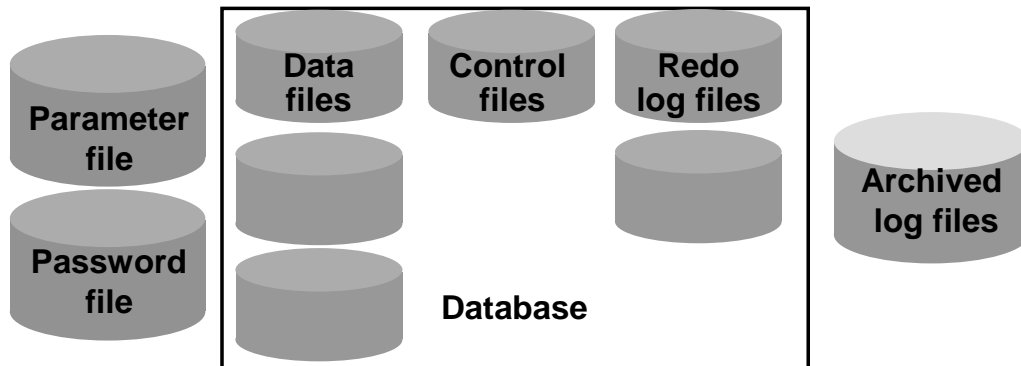
SQL Statement Processing

The user and server processes are the primary processes involved when a SQL statement is executed; however, other processes may help the server complete the processing of the SQL statement.

Oracle Database Administrators

Database administrators are responsible for maintaining the Oracle Server so that the server can process user requests. An understanding of the Oracle architecture is necessary to maintain it effectively.

Oracle Database Files



ORACLE

E-5

Copyright © Oracle Corporation, 2001. All rights reserved.

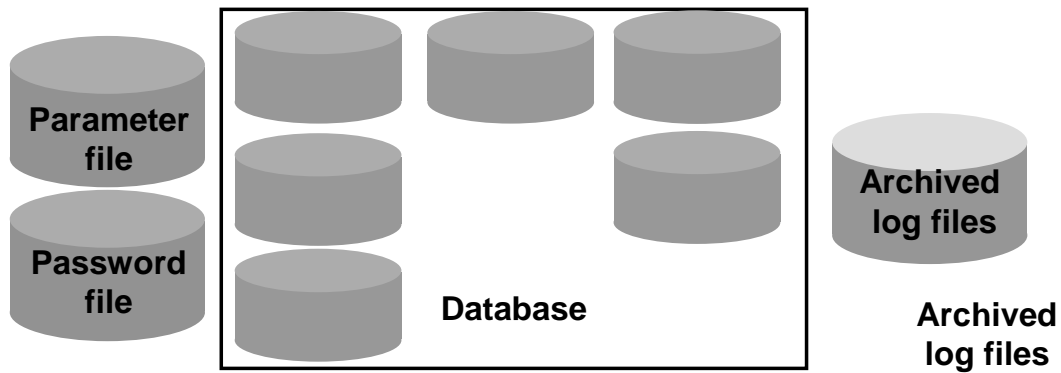
Oracle Database Files

An Oracle database is a collection of data that is treated as a unit. The general purpose of a database is to store and retrieve related information. The database has a logical structure and a physical structure. The physical structure of the database is the set of operating system files in the database. An Oracle database consists of three file types:

Data files contain the actual data in the database. The data is stored in user-defined tables, but data files also contain the data dictionary, before-images of modified data, indexes, and other types of structures. A database has at least one data file. The characteristics of data files are:

- A data file can be associated with only one database. Data files can have certain characteristics set so they can automatically extend when the database runs out of space. One or more data files form a logical unit of database storage called a tablespace. Redo logs contain a record of changes made to the database to enable recovery of the data in case of failures. A database requires at least two redo log files.
- Control files contain information necessary to maintain and verify database integrity. For example, a control file is used to identify the data files and redo log files. A database needs at least one control file.

Other Key Physical Structures



ORACLE

E-6

Copyright © Oracle Corporation, 2001. All rights reserved.

Other Key Files

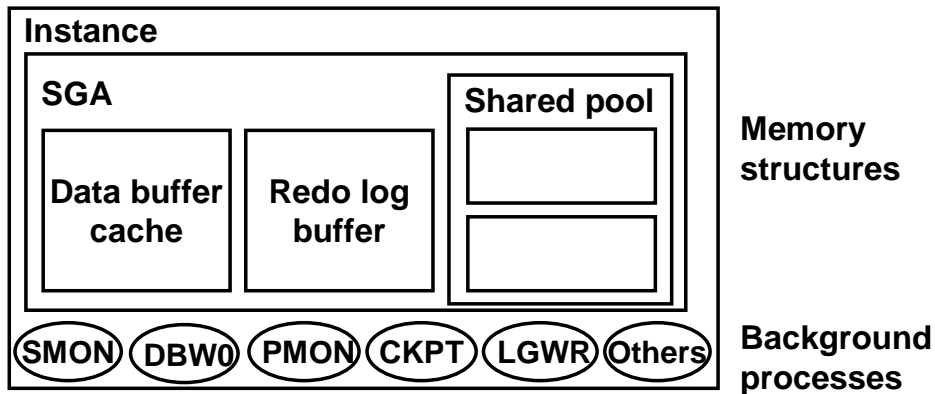
The Oracle Server also uses other files that are not part of the database:

- The parameter file defines the characteristics of an Oracle instance. For example, it contains parameters that size some of the memory structures in the SGA.
- The password file authenticates which users are permitted to start up and shut down an Oracle instance.
- Archived redo log files are offline copies of the redo log files that may be necessary to recover from media failures.

Oracle Instance

An Oracle instance:

- Is a means to access an Oracle database
- Always opens one and only one database



ORACLE

E-7

Copyright © Oracle Corporation, 2001. All rights reserved.

Oracle Instance

An Oracle instance consists of the SGA memory structure and the background processes used to manage a database. An instance is identified by using methods specific to each operating system. The instance can open and use only one database at a time.

System Global Area

The SGA is a memory area used to store database information that is shared by database processes. It contains data and control information for the Oracle Server. It is allocated in the virtual memory of the computer where the Oracle server resides. The SGA consists of several memory structures:

- The shared pool is used to store the most recently executed SQL statements and the most recently used data from the data dictionary. These SQL statements may be submitted by a user process or, in the case of stored procedures, read from the data dictionary.
- The database buffer cache is used to store the most recently used data. The data is read from, and written to, the data files.
- The redo log buffer is used to track changes made to the database by the server and background processes.

Oracle Instance

System Global Area (continued)

The purpose of these structures is discussed in detail in later sections of this lesson.

There are also two optional memory structures in the SGA:

- Java pool: Used to store Java code
- Large pool: Used to store large memory structures not directly related to SQL statement processing; for example, data blocks copied during backup and restore operations

Background Processes

The background processes in an instance perform common functions that are needed to service requests from concurrent users without compromising the integrity and performance of the system. They consolidate functions that would otherwise be handled by multiple Oracle programs running for each user. The background processes perform I/O and monitor other Oracle processes to provide increased parallelism for better performance and reliability.

Depending on its configuration, an Oracle instance may include several background processes, but every instance includes these five required background processes:

- Database Writer (DBW0) is responsible for writing changed data from the database buffer cache to the data files.
- Log Writer (LGWR) writes changes registered in the redo log buffer to the redo log files.
- System Monitor (SMON) checks for consistency of the database and, if necessary, initiates recovery of the database when the database is opened.
- Process Monitor (PMON) cleans up resources if one of the Oracle processes fails.
- The Checkpoint Process (CKPT) is responsible for updating database status information in the control files and data files whenever changes in the buffer cache are permanently recorded in the database.

The following sections of this lesson explain how a server process uses some of the components of the Oracle instance and database to process SQL statements submitted by a user process.

Processing a SQL Statement

- **Connect to an instance using:**
 - The user process
 - The server process
- **The Oracle Server components that are used depend on the type of SQL statement:**
 - Queries return rows
 - DML statements log changes
 - Commit ensures transaction recovery
- **Some Oracle Server components do not participate in SQL statement processing.**

ORACLE

E-9

Copyright © Oracle Corporation, 2001. All rights reserved.

Components Used to Process SQL

Not all of the components of an Oracle instance are used to process SQL statements. The user and server processes are used to connect a user to an Oracle instance. These processes are not part of the Oracle instance, but are required to process a SQL statement.

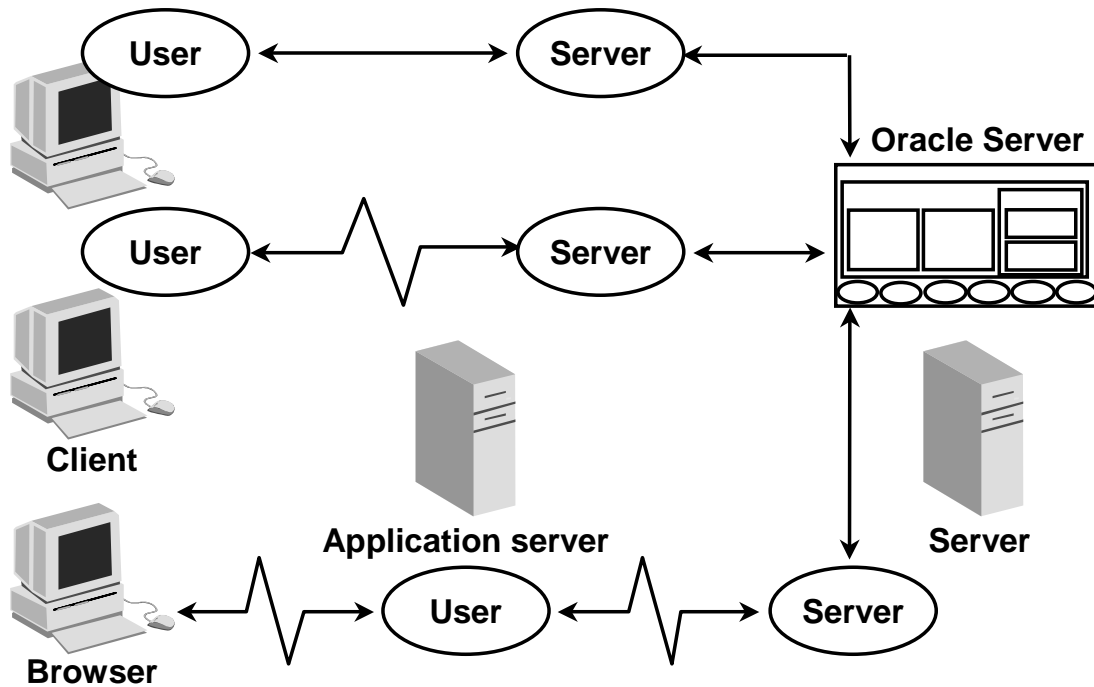
Some of the background processes, SGA structures, and database files are used to process SQL statements. Depending on the type of SQL statement, different components are used:

- Queries require additional processing to return rows to the user
- Data manipulation language (DML) statements require additional processing to log the changes made to the data
- Commit processing ensures that the modified data in a transaction can be recovered

Some required background processes do not directly participate in processing a SQL statement but are used to improve performance and to recover the database.

The optional background process, ARC0, is used to ensure that a production database can be recovered.

Connecting to an Instance



E-10

Copyright © Oracle Corporation, 2001. All rights reserved.

Processes Used to Connect to an Instance

Before users can submit SQL statements to the Oracle Server, they must connect to an instance.

The user starts a tool such as *iSQL*Plus* or runs an application developed using a tool such as Oracle Forms. This application or tool is executed in a *user process*.

In the most basic configuration, when a user logs on to the Oracle Server, a process is created on the computer running the Oracle Server. This process is called a server process. The server process communicates with the Oracle instance on behalf of the user process that runs on the client. The server process executes SQL statements on behalf of the user.

Connection

A connection is a communication pathway between a user process and an Oracle Server. A database user can connect to an Oracle Server in one of three ways:

- The user logs on to the operating system running the Oracle instance and starts an application or tool that accesses the database on that system. The communication pathway is established using the interprocess communication mechanisms available on the host operating system.

Processes Used to Connect to an Instance

Connection (continued)

- The user starts the application or tool on a local computer and connects over a network to the computer running the Oracle instance. In this configuration, called client-server, network software is used to communicate between the user and the Oracle Server.
- In a three-tiered connection, the user's computer communicates over the network to an application or a network server, which is connected through a network to the machine running the Oracle instance. For example, the user runs a browser on a network computer to use an application residing on an NT server that retrieves data from an Oracle database running on a UNIX host.

Sessions

A session is a specific connection of a user to an Oracle Server. The session starts when the user is validated by the Oracle Server, and it ends when the user logs out or when there is an abnormal termination. For a given database user, many concurrent sessions are possible if the user logs on from many tools, applications, or terminals at the same time. Except for some specialized database administration tools, starting a database session requires that the Oracle Server be available for use.

Note: The type of connection explained here, where there is a one-to-one correspondence between a user and server process, is called a dedicated server connection.

Processing a Query

- **Parse:**
 - Search for identical statement
 - Check syntax, object names, and privileges
 - Lock objects used during parse
 - Create and store execution plan
- **Execute: Identify rows selected**
- **Fetch: Return rows to user process**

ORACLE

E-12

Copyright © Oracle Corporation, 2001. All rights reserved.

Query Processing Steps

Queries are different from other types of SQL statements because, if successful, they return data as results. Whereas other statements simply return success or failure, a query can return one row or thousands of rows.

There are three main stages in the processing of a query:

- Parse
- Execute
- Fetch

Parsing a SQL Statement

During the *parse* stage, the SQL statement is passed from the user process to the server process, and a parsed representation of the SQL statement is loaded into a shared SQL area.

During the parse, the server process performs the following functions:

- Searches for an existing copy of the SQL statement in the shared pool
- Validates the SQL statement by checking its syntax
- Performs data dictionary lookups to validate table and column definitions

The Shared Pool



- **The library cache contains the SQL statement text, parsed code, and execution plan.**
- **The data dictionary cache contains table, column, and other object definitions and privileges.**
- **The shared pool is sized by `SHARED_POOL_SIZE`.**

ORACLE

E-13

Copyright © Oracle Corporation, 2001. All rights reserved.

Shared Pool Components

During the parse stage, the server process uses the area in the SGA known as the shared pool to compile the SQL statement. The shared pool has two primary components:

- Library cache
- Data dictionary cache

Library Cache

The library cache stores information about the most recently used SQL statements in a memory structure called a shared SQL area. The shared SQL area contains:

- The text of the SQL statement
- The parse tree: A compiled version of the statement
- The execution plan: The steps to be taken when executing the statement

The optimizer is the function in the Oracle Server that determines the optimal execution plan.

If a SQL statement is reexecuted and a shared SQL area already contains the execution plan for the statement, the server process does not need to parse the statement. The library cache improves the performance of applications that reuse SQL statements by reducing parse time and memory requirements. If the SQL statement is not reused, it is eventually aged out of the library cache.

Shared Pool Components (continued)

Data Dictionary Cache

The data dictionary cache, also known as the dictionary cache or row cache, is a collection of the most recently used definitions in the database. It includes information about database files, tables, indexes, columns, users, privileges, and other database objects.

During the parse phase, the server process looks for the information in the dictionary cache to resolve the object names specified in the SQL statement and to validate the access privileges. If necessary, the server process initiates the loading of this information from the data files.

Sizing the Shared Pool

The size of the shared pool is specified by the initialization parameter `SHARED_POOL_SIZE`.

Database Buffer Cache



- **Stores the most recently used blocks**
- **Size of a buffer based on `DB_BLOCK_SIZE`**
- **Number of buffers defined by `DB_BLOCK_BUFFERS`**

ORACLE

E-15

Copyright © Oracle Corporation, 2001. All rights reserved.

Function of the Database Buffer Cache

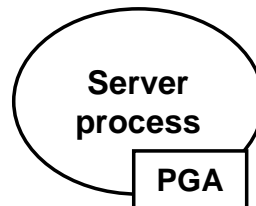
When a query is processed, the server process looks in the database buffer cache for any blocks it needs. If the block is not found in the database buffer cache, the server process reads the block from the data file and places a copy in the buffer cache. Because subsequent requests for the same block may find the block in memory, the requests may not require physical reads. The Oracle Server uses a least recently used algorithm to age out buffers that have not been accessed recently to make room for new blocks in the buffer cache.

Sizing the Database Buffer Cache

The size of each buffer in the buffer cache is equal to the size of an Oracle block, and it is specified by the `DB_BLOCK_SIZE` parameter. The number of buffers is equal to the value of the `DB_BLOCK_BUFFERS` parameter.

Program Global Area (PGA)

- **Not shared**
- **Writable only by the server process**
- **Contains:**
 - **Sort area**
 - **Session information**
 - **Cursor state**
 - **Stack space**



ORACLE

E-16

Copyright © Oracle Corporation, 2001. All rights reserved.

Program Global Area Components

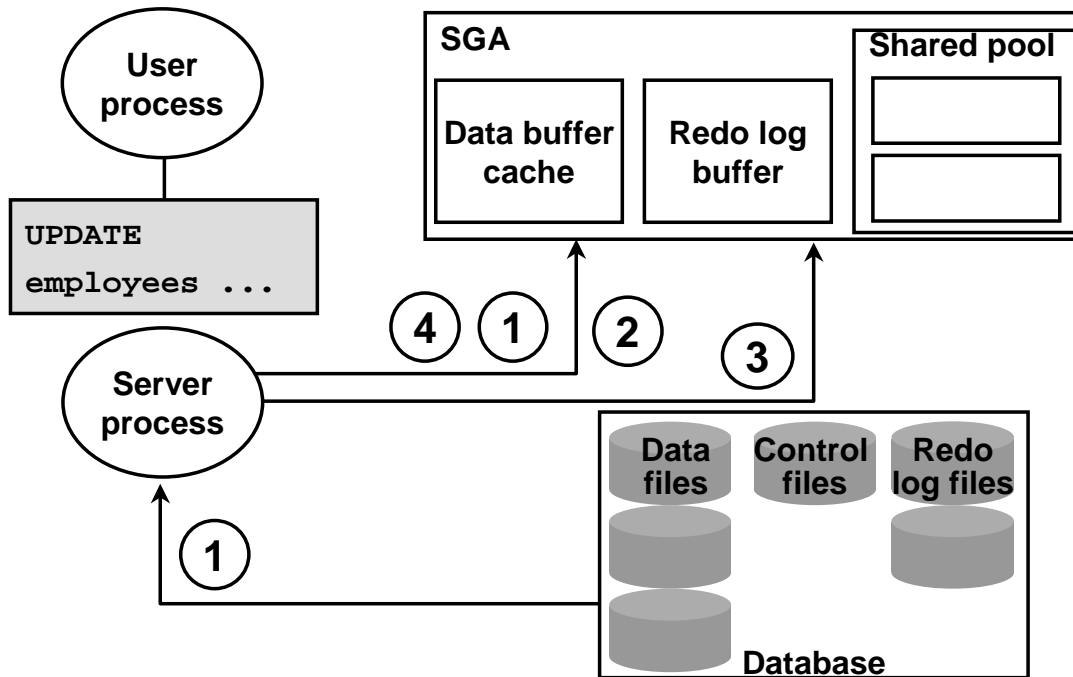
A program global area (PGA) is a memory region that contains data and control information for a server process. It is a nonshared memory created by Oracle when a server process is started. Access to it is exclusive to that server process and is read and written only by the Oracle Server code acting on behalf of it. The PGA memory allocated by each server process attached to an Oracle instance is referred to as the aggregated PGA memory allocated by the instance.

In a dedicated server configuration, the PGA of the server includes these components:

- **Sort area:** Used for any sorts that may be required to process the SQL statement
- **Session information:** Includes user privileges and performance statistics for the session
- **Cursor state:** Indicates the stage in the processing of the SQL statements that are currently used by the session
- **Stack space:** Contains other session variables

The PGA is allocated when a process is created and deallocated when the process is terminated.

Processing a DML Statement



ORACLE

E-17

Copyright © Oracle Corporation, 2001. All rights reserved.

DML Processing Steps

A data manipulation language (DML) statement requires only two phases of processing:

- Parse is the same as the parse phase used for processing a query
- Execute requires additional processing to make data changes

DML Execute Phase

To execute a DML statement:

- If the data and rollback blocks are not already in the buffer cache, the server process reads them from the data files into the buffer cache.
- The server process places locks on the rows that are to be modified.
- In the redo log buffer, the server process records the changes to be made to the rollback and data.
- The rollback block changes record the values of the data before it is modified. The rollback block is used to store the before image of the data, so that the DML statements can be rolled back if necessary.
- The data blocks changes record the new values of the data.

DML Processing Steps

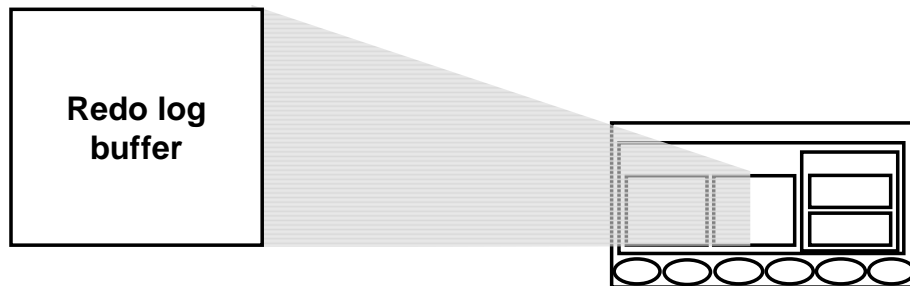
DML Execute Phase (continued)

The server process records the before image to the rollback block and updates the data block. Both of these changes are done in the database buffer cache. Any changed blocks in the buffer cache are marked as dirty buffers: that is, buffers that are not the same as the corresponding blocks on the disk.

The processing of a DELETE or INSERT command uses similar steps. The before image for a DELETE contains the column values in the deleted row, and the before image of an INSERT contains the row location information.

Because the changes made to the blocks are only recorded in memory structures and are not written immediately to disk, a computer failure that causes the loss of the SGA can also lose these changes.

Redo Log Buffer



- **Has its size defined by LOG_BUFFER**
- **Records changes made through the instance**
- **Is used sequentially**
- **Is a circular buffer**

ORACLE

E-19

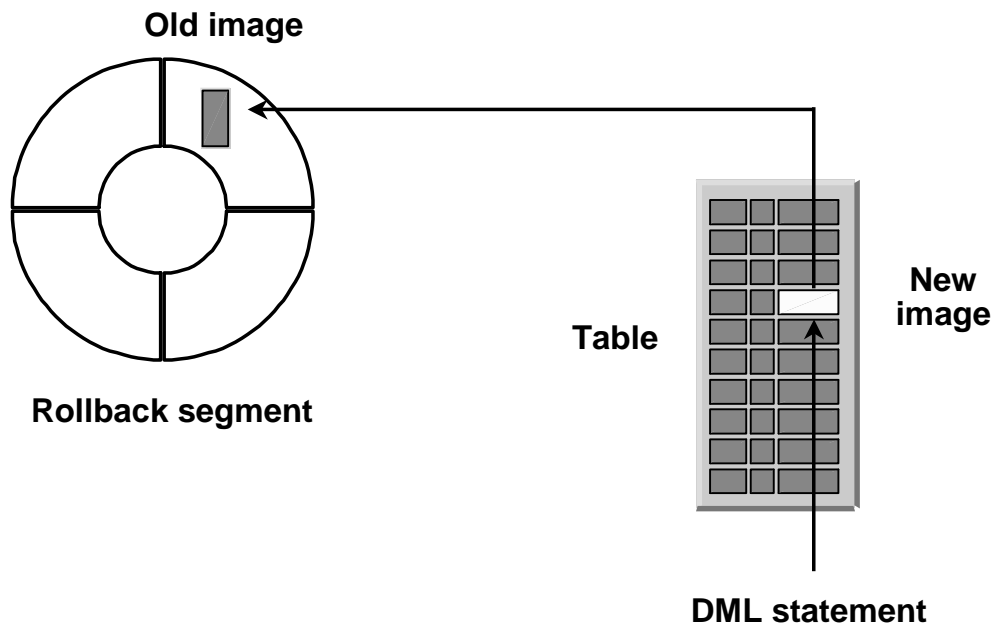
Copyright © Oracle Corporation, 2001. All rights reserved.

Redo Log Buffer Characteristics

The server process records most of the changes made to data file blocks in the redo log buffer, which is a part of the SGA. The redo log buffer has the following characteristics:

- Its size in bytes is defined by the LOG_BUFFER parameter.
- It records the block that is changed, the location of the change, and the new value in a redo entry. A redo entry makes no distinction between the type of block that is changed; it simply records which bytes are changed in the block.
- The redo log buffer is used sequentially, and changes made by one transaction may be interleaved with changes made by other transactions.
- It is a circular buffer that is reused after it is filled, but only after all the old redo entries are recorded in the redo log files.

Rollback Segment



ORACLE

E-20

Copyright © Oracle Corporation, 2001. All rights reserved.

Rollback Segment

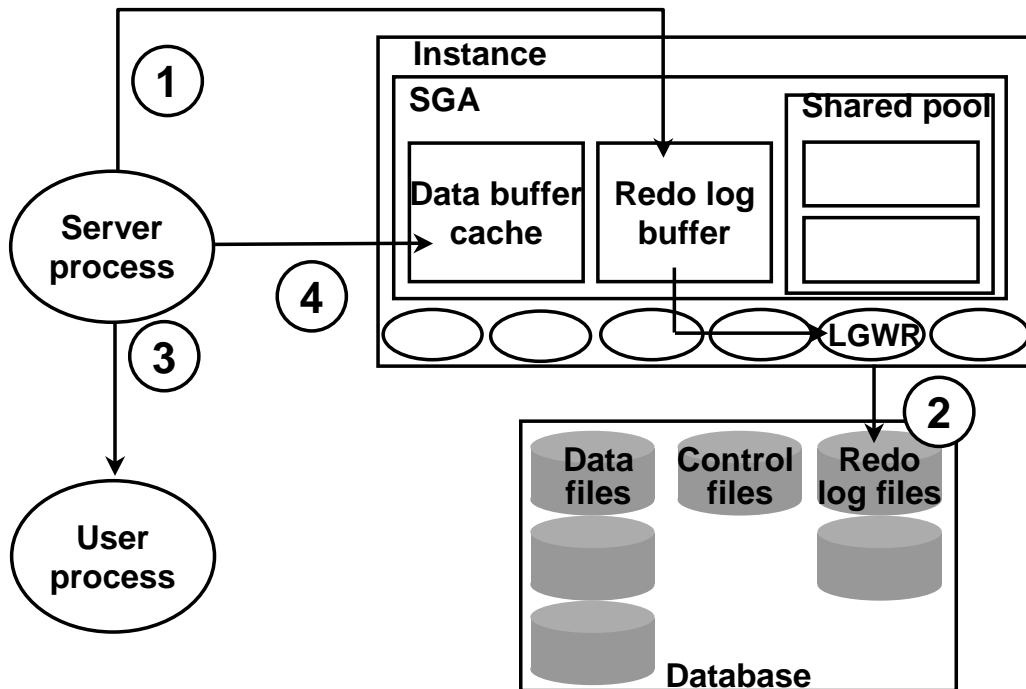
Before making a change, the server process saves the old data value into a rollback segment. This before image is used to:

- Undo the changes if the transaction is rolled back
- Provide read consistency by ensuring that other transactions do not see uncommitted changes made by the DML statement
- Recover the database to a consistent state in case of failures

Rollback segments, like tables and indexes, exist in data files, and rollback blocks are brought into the database buffer cache as required. Rollback segments are created by the DBA.

Changes to rollback segments are recorded in the redo log buffer.

COMMIT Processing



ORACLE

E-21

Copyright © Oracle Corporation, 2001. All rights reserved.

Fast COMMIT

The Oracle Server uses a fast commit mechanism that guarantees that the committed changes can be recovered in case of instance failure.

System Change Number

Whenever a transaction commits, the Oracle Server assigns a commit system change number (SCN) to the transaction. The SCN is monotonically incremented and is unique within the database. It is used by the Oracle Server as an internal time stamp to synchronize data and to provide read consistency when data is retrieved from the data files. Using the SCN enables the Oracle Server to perform consistency checks without depending on the date and time of the operating system.

Steps in Processing COMMITs

When a COMMIT is issued, the following steps are performed:

- The server process places a commit record, along with the SCN, in the redo log buffer.
- LGWR performs a contiguous write of all the redo log buffer entries up to and including the commit record to the redo log files. After this point, the Oracle Server can guarantee that the changes will not be lost even if there is an instance failure.

Fast COMMIT

Steps in Processing COMMITs (continued)

- The user is informed that the COMMIT is complete.
- The server process records information to indicate that the transaction is complete and that resource locks can be released.

Flushing of the dirty buffers to the data file is performed independently by DBW0 and can occur either before or after the commit.

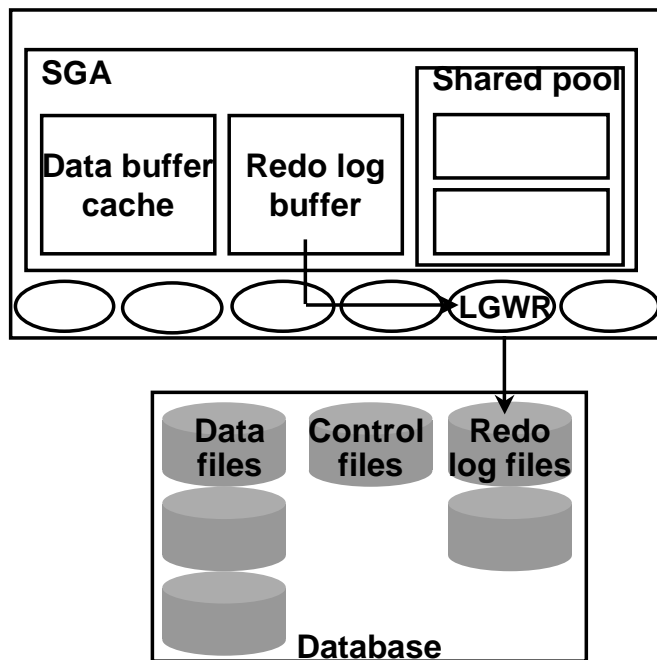
Advantages of the Fast COMMIT

The fast commit mechanism ensures data recovery by writing changes to the redo log buffer instead of the data files. It has the following advantages:

- Sequential writes to the log files are faster than writing to different blocks in the data file.
- Only the minimal information that is necessary to record changes is written to the log files, whereas writing to the data files would require whole blocks of data to be written.
- If multiple transactions request to commit at the same time, the instance piggybacks redo log records into a single write.
- Unless the redo log buffer is particularly full, only one synchronous write is required per transaction. If piggybacking occurs, there can be less than one synchronous write per transaction.
- Because the redo log buffer may be flushed before the COMMIT, the size of the transaction does not affect the amount of time needed for an actual COMMIT operation.

Note: Rolling back a transaction does not trigger LGWR to write to disk. The Oracle Server always rolls back uncommitted changes when recovering from failures. If there is a failure after a rollback, before the rollback entries are recorded on disk, the absence of a commit record is sufficient to ensure that the changes made by the transaction are rolled back.

Log Writer (LGWR)



LGWR writes when:

- There is a COMMIT
- The redo buffer log is one-third full
- There is more than 1 MB of redo
- Before DBW0 writes

ORACLE

E-23

Copyright © Oracle Corporation, 2001. All rights reserved.

LOG Writer

LGWR performs sequential writes from the redo log buffer to the redo log file under the following situations:

- When a transaction commits
- When the redo log buffer is one-third full
- When there is more than a megabyte of changes recorded in the redo log buffer
- Before DBW0 writes modified blocks in the database buffer cache to the data files

Because the redo is needed for recovery, LGWR confirms the COMMIT only after the redo is written to disk.

Other Instance Processes

- **Other required processes:**
 - **Database Writer (DBW0)**
 - **Process Monitor (PMON)**
 - **System Monitor (SMON)**
 - **Checkpoint (CKPT)**
- **The archive process (ARC0) is usually created in a production database**

ORACLE

E-24

Copyright © Oracle Corporation, 2001. All rights reserved.

Other Required Processes

Four other required processes do not participate directly in processing SQL statements:

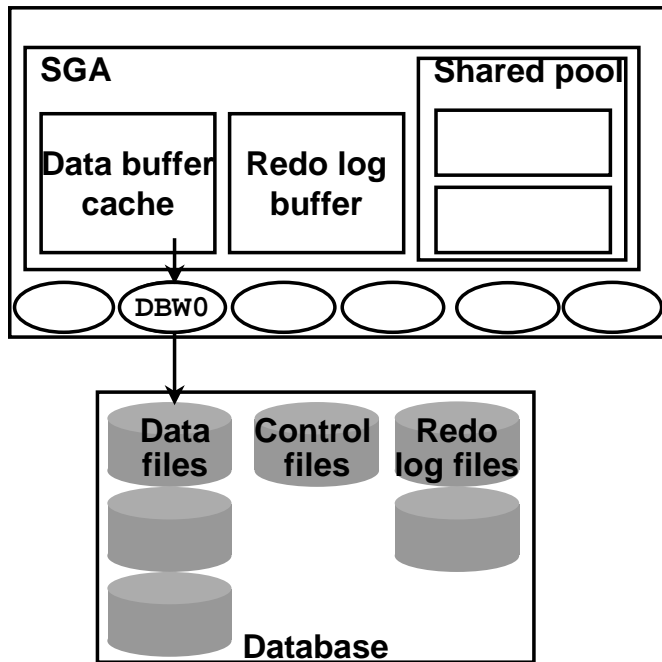
- Database Writer (DBW0)
- Process Monitor (PMON)
- System Monitor (SMON)
- Checkpoint (CKPT)

The checkpoint process is used to synchronize database files.

The Archiver Process

All other background processes are optional, depending on the configuration of the database; however, one of them, ARC0, is crucial to recovering a database after the loss of a disk. The ARC0 process is usually created in a production database.

Database Writer (DBW0)



DBW0 writes when:

- There are many dirty buffers
- There are few free buffers
- Timeout occurs
- Checkpoint occurs

ORACLE

E-25

Copyright © Oracle Corporation, 2001. All rights reserved.

Database Writer

The server process records changes to rollback and data blocks in the buffer cache. The Database Writer (DBW0) writes the dirty buffers from the database buffer cache to the data files. It ensures that a sufficient number of free buffers (buffers that can be overwritten when server processes need to read in blocks from the data files) are available in the database buffer cache. Database performance is improved because server processes make changes only in the buffer cache, and the DBW0 defers writing to the data files until one of the following events occurs:

- The number of dirty buffers reaches a threshold value
- A process scans a specified number of blocks when scanning for free buffers and cannot find any
- A timeout occurs (every three seconds)
- A checkpoint occurs (a checkpoint is a means of synchronizing the database buffer cache with the data file)

SMON: System Monitor

- **Automatically recovers the instance:**
 - **Rolls forward changes in the redo logs**
 - **Opens the database for user access**
 - **Rolls back uncommitted transactions**
- **Coalesces free space**
- **Deallocates temporary segments**

ORACLE

E-26

Copyright © Oracle Corporation, 2001. All rights reserved.

SMON: System Monitor

If the Oracle instance fails, any information in the SGA that has not been written to disk is lost. For example, the failure of the operating system causes an instance failure. After the loss of the instance, the background process SMON automatically performs instance recovery when the database is reopened. Instance recovery consists of the following steps:

- Rolling forward to recover data that has not been recorded in the data files but that has been recorded in the online redo log. This data has not been written to disk because of the loss of the SGA during instance failure. During this process, SMON reads the redo log files and applies the changes recorded in the redo log to the data blocks. Because all committed transactions have been written to the redo logs, this process completely recovers these transactions.
- Opening the database so users can log on. Any data that is not locked by unrecovered transactions is immediately available.
- Rolling back uncommitted transactions. They are rolled back by SMON or by the individual server processes as they access locked data.

SMON also performs some space maintenance functions:

- It combines, or coalesces, adjacent areas of free space in the data files.
- It deallocates temporary segments to return them as free space in data files. Temporary segments are used to store data during SQL statement processing.

PMON: Process Monitor

Cleans up after failed processes by:

- **Rolling back the transaction**
- **Releasing locks**
- **Releasing other resources**

ORACLE

E-27

Copyright © Oracle Corporation, 2001. All rights reserved.

PMON Functionality

The background process PMON cleans up after failed processes by:

- Rolling back the user's current transaction
- Releasing all currently held table or row locks
- Freeing other resources currently reserved by the user

Summary

In this appendix, you should have learned how to:

- **Identify database files: data files, control files, online redo logs**
- **Describe SGA memory structures: DB buffer cache, shared SQL pool, and redo log buffer**
- **Explain primary background processes: DBW0, LGWR, CKPT, PMON, SMON, and ARC0**
- **List SQL processing steps: parse, execute, fetch**

ORACLE

E-28

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

The Oracle database includes these files:

- **Control files:** Contain information required to verify the integrity of the database, including the names of the other files in the database (The control files are usually mirrored.)
- **Data files:** Contain the data in the database, including tables, indexes, rollback segments, and temporary segments
- **Online redo logs:** Contain the changes made to the data files (Online redo logs are used for recovery and are usually mirrored.)

Other files commonly used with the database include:

- **Parameter file:** Defines the characteristics of an Oracle instance
- **Password file:** Authenticates privileged database users
- **Archived redo logs:** Are backups of the online redo logs

SGA Memory Structures

The System Global Area (SGA) has three primary structures:

- Shared pool: Stores the most recently executed SQL statements and the most recently used data from the data dictionary
- Database buffer cache: Stores the most recently used data
- Redo log buffer: Records changes made to the database using the instance

Background Processes

A production Oracle instance includes these processes:

- Database Writer (DBW0): Writes changed data to the data files
- Log Writer (LGWR): Records changes to the data files in the online redo log files
- System Monitor (SMON): Checks for consistency and initiates recovery of the database when the database is opened
- Process Monitor (PMON): Cleans up the resources if one of the processes fails
- Checkpoint Process (CKPT): Updates the database status information after a checkpoint
- Archiver (ARC0): Backs up the online redo log to ensure recovery after a media failure (This process is optional, but is usually included in a production instance.)

Depending on its configuration, the instance may also include other processes.

SQL Statement Processing Steps

The steps used to process a SQL statement include:

- Parse: Compiles the SQL statement
- Execute: Identifies selected rows or applies DML changes to the data
- Fetch: Returns the rows queried by a SELECT statement

Index

Note: A bolded number or letter refers to an entire lesson or appendix.

A

APPEND Command C-11

ACCESS PARAMETER 20-19

Adding Data through a View 11-16

ADD_MONTHS Function 3-21

ALL Operator 6-16

Alias 1-4, 1-17, 1-16, 2-7, 2-24, 11-9

 Table Aliases 4-12

ALL INSERT (Conditional) 20-7

ALL_COL_COMMENT Data Dictionary View 9-32

ALL_TAB_COMMENT Data Dictionary View 9-32

ALTER SEQUENCE Statement 12-12

ALTER TABLE Statement 9-22, 9-23, 10-17, 10-20, 10-21, 13-11

ALTER USER Statement 13-11

Ambiguous Column Names 4-11

American National Standards Institute I-24

ANSI I-24

ANY Operator 6-15

Application Server I-5

Archived Redo Log File E-6

Arguments 3-3, 3-5

Arithmetic Expression 1-9

Arithmetic Operator 1-9

AS Subquery Clause 9-20

Assigning Privileges 13-7

Attributes I-16, I-19

AVG Function 5-6, 5-7

Index

B

Background Processes E-3, E-7

BETWEEN Operator 2-10

BREAK Command 7-18

BTITLE Command 7-19

C

CHANGE Command C-12

Caching Sequence 12-1

Calculations in Expressions 1-9

Cardinality 1-18

Cartesian Product 4-4, 4-5

CASE Expression 3-51, 3-52, 18-12

CASCADE CONSTRAINTS Clause 10-22

Character Data Type in Functions 3-4

Character Strings 2-5, 2-6

CHECK Constraint 10-16

Checkpoint Process E-8

Child Node 19-10

CLEAR BREAK Command 7-18

COALESCE Function 3-49

COLUMN Command 7-16, 7-17

Column Level Constraints 10-8

Command or Script Files 7-20

COMMENT Statement 9-32

COMMIT Statement 8-2, 8-33, 8-35, 8-39, 8-40, 9-8

Comparison Operator, Comparison Conditions 2-7, 18-4

Composite Column 17-17

Composite Unique Key 10-10

CONCAT Function 3-11

Concatenated Groupings 17-21

Concatenation Operator 1-18

Index

C

- Conditional `FIRST INSERT` 20-7, 20-13, 20-14
- Conditional If-Then-Else Logic 3-51
- Conditional `INSERT ALL` 20-7, 20-11
- Conditional Processing 3-51
- Conditions, Logical 2-15
- `CONNECT BY` Clause 19-5, 19-7, 19-13
- CONSTRAINTS 10**
 - `CASCADE CONSTRAINTS` Clause 10-22
 - `CHECK` Constraint 10-16
 - Column-Level Constraints 10-8
 - Defining Constraints 10-5
 - Deleting a Record with an Integrity Constraint 8-22
 - Disabling 10-20
 - Dropping a Constraint 10-19
 - `FOREIGN KEY` 10-13, 10-14, 10-15, 1-19
 - `NOT NULL` Constraint 10-7
 - Primary Key 10-11
 - `READ ONLY` Constraint 11-19
 - `REFERENCE` Constraint 10-15
 - Referential Integrity Constraint 10-13
 - Table-Level Constraints 10-8
 - `UNIQUE` Constraint 10-9, 10-10
- Controlling Database Access **13**
- Control File E-5
- Correlated Subquery 18-2, 18-13, 18-14, 18-15, 18-21, 18-24
- Correlated `UPDATE` 18-22
- Correlation 18-17
- `COUNT` Function 5-8

C

CREATE DATABASE Statement 16-9
CREATE DIRECTORY Statement 20-20
CREATE INDEX Statement 12-17, 20-24
Creating Scripts 1-26
CREATE SEQUENCE Statement 12-5
CREATE TABLE Statement **9**
CREATE USER Statement 13-6
CREATE VIEW Statement 11-7
Cross Tabular Reports 17-9
Cross Tabulation Rows 17-6
Cross Tabulation Values 17-10
CUBE Operator 17-2, 17-6, 17-9
CURRENT_DATE Function 16-6
CURRENT_TIMESTAMP Function 16-7
CURRVAL 9-7, 12-8
CYCLE Clause (Sequences) 12-6

D

Date Functions 3-6
Data Control Language (DCL) Statements 8-33, **9**
Data Definition Language (DDL) Statements 8-33, 9-5, **13**
Data Manipulation Language (DML) Statements **8**
 DML Operations through a View 11-14
Data Dictionary Tables 9-9, D-3
Data Dictionary Cache E-13, E-14
Data File E-5
Data from More than One Table (Joins) **4**
Data Structures in the Oracle Database 9-3, 9-5

Index

D

- Data Types 3-25
- Data Warehouse Applications 1-8
- Database Links 13-19
- Database Writer E-8
- Date Conversion Functions 3-4, 3-35
- Datetime Data Type 9-14
- Datetime Functions 16-2
- Daylight Savings Time 16-5
- DBTIMEZONE Function 16-9
- DECODE Expression 3-51, 3-54
- DEFAULT Clause 8-26, 8-27, 9-7
- Default Date Display 2-6, 3-17
- DEFAULT DIRECTORY 20-19
- Default Sort Order 2-23
- DEFINE Command 7-5, 7-11
- Defining Constraints 10-5
- DELETE Statement 8-19, 8-20, 13-16
- DESCRIBE Command 1-29, 8-7, 10-24, 11-13, C-7
- DISABLE Clause 10-20
- DISTINCT Keyword 1-4, 1-23, 5-5, 5-10
- Dropping a Constraint 10-19
- DROP ANY INDEX Statement 12-2
- DROP ANY VIEW Statement 11-20

D

DROP COLUMN Clause 9-27

DROP INDEX Statement 12

DROP SEQUENCE Statement 12-14

DROP SYNONYM 12-24

DROP TABLE Statement 9-29

DROP UNUSED COLUMNS Clause 9-28

DROP VIEW Statement 11-20

DUAL Table 3-14, 3-18

Duplicate Records 15-11

E

E-business 19-6, I-3

EDIT Command C-14

Entity I-16, I-17, I-18

Entity Relationship Diagram I-16, I-17, I-16

Equijoins 4-8, 4-27

ESCAPE Option 2-13

Exclusive Locks 8-46

E

Execute Button (in *iSQL*Plus*) 1-7, 1-32

Executing SQL 1-26

EXISTS Operator 18-18, 18-19

Explicit Data Type Conversion 3-25

Expressions

 Calculations in Expressions 1-9

 CASE Expression 3-51, 3-52, 18-12

 DECODE Expression 3-51, 3-54

 If-Then-Else Logic 3-51

External Tables **20**

 Conditional FIRST INSERT 20-7, 20-13, 20-14

 Conditional INSERT ALL 20-7, 20-11

 ORGANIZATION EXTERNAL Clause 20-18, 20-19

 Pivoting INSERT 20-7, 20-15

 Unconditional INSERT 20-7, 20-10

 REJECT LIMIT Clause 20-19

 TYPE ACCESS_DRIVER_TYPE 20-19

EXTRACT Function 16-10

F

FOREIGN KEY Constraint 10-13, 10-14, 10-15, I-19

Format Mode (fm) 3-31

FRACTIONAL_SECONDS_PRECISION 9-15

FROM Clause **1**

FROM Clause Query 11-21, 18-2, 18-10

FROM_TZ Function 16-11

Index

F

Functions 3, 5

- AVG (Average) 5-6, 5-7
- Character Data Type in Functions 3-4
- COALESCE Function 3-49
- CONCAT Function 3-11
- COUNT Function 5-8
- CURRENT_DATE Function 16-6
- CURRENT_TIMESTAMP Function 16-7
- Date Conversion Functions 3-4, 3-35
- Datetime Functions 16-2
- DBTIMEZONE Function 16-9
- EXTRACT Function 16-10
 - TIMEZONE_ABBR 16-10
 - TIMEZONE_REGION 16-10
- FROM_TZ Function 16-11
- INITCAP Function 3-9
- INSTR Function 3-11
- LAST_DAY Function 3-21
- LENGTH Function 3-11
- LOCALTIMESTAMP Function 16-8
- LOWER Function 3-9
- LPAD Function 3-11
- MAX Function 5-6, 5-7
- MIN Function 5-6, 5-7
- MONTHS_BETWEEN Function 3-6, 3-21
- Multiple-row Function 3-4

F

Functions 3, 5

NEXT_DAY Function 3-21

NULLIF Function 3-48

Number Functions 3-13

NVL Function 3-45, 3-46, 5-5, 5-12

NVL2 Function 3-47

Returning a Value 3-3

ROUND Function 3-14, 3-21, 3-23

SESSIONTIMEZONE Function 16-9

STDDEV Function 5-7

SUBSTR Function 3-11

SUM Function 5-6, 5-7

SYS Function 9-9

SYSDATE Function 3-18, 3-20, 9-7

TO_CHAR Function 3-31, 3-37, 3-39

TO_DATE Function 3-39

TO_NUMBER Function 3-39

TO_TIMESTAMP Function 16-12

TO_YMINTERVAL Function 16-13

TRIM Function 3-11

TRUNC Function 3-15, 3-21, 3-23

TZOFFSET 16-14 Function

UPPER Function 3-9, 3-10

USER Function 9-7

Function-based Indexes 12-21

G

- Generating Unique Numbers 12-3
- GRANT Statement **13**
- Greenwich Mean Time 16-3
- Gregorian Calendar 16-10
- GROUP BY Clause 5-13, 5-14, 5-15, 5-16, 17-3, 17-4
- GROUP BY ROLLUP 17-17
- Grouping Data **5**, 17-2
- Group Functions **5**
- Group Functions in a Subquery 6-10
- Group Functions and NULL Values 5-11
- GROUPING SETS Clause 17-12, 17-11, 17-13
- Guidelines for Creating a View 11-8

H

- Hash Sign 3-38
- HAVING Clause 5-21, 5-22, 5-23, 6-11, 17-5
- Hierarchical Queries **19**
 - Child Node 19-10
 - CONNECT BY Clause 19-5, 19-7, 19-13
 - PRIOR Clause 19-7
 - Pruning the Tree 19-13
 - START WITH Clause 19-5, 19-6

I

- If-Then-Else Logic 3-51
- Implicit Data Type Conversion 3-25
- Indexes 9-3, **12**
 - CREATE INDEX Statement 12-17, 20-24
 - Naming Indexes 20-2
 - Non-unique Indexes 12-16
 - Unique Index 10-10, 12-6
 - When to Create an Index 12-18

Index

I

- INITCAP Function 3-9
- Inline Views 11-2, 11-21
- Inner Query 6-3, 6-4, 6-5, 18-5
- INSERT Statement 8-5, 8-6, 8-11, 13-18, 20-2, 20-7
 - Conditional FIRST INSERT 20-7, 20-13, 20-14
 - Conditional INSERT ALL 20-7, 20-11
 - Pivoting INSERT 20-7, 20-15
 - Unconditional INSERT 20-7, 20-10
 - VALUES Clause 8-5
- INSTR Function 3-11
- Integrity Constraints 8-17, 10-2
- International Standards Organization (ISO) 1-24
- Internet Features 1-7
- INTERSECT Operator 15-12
- INTERVAL YEAR TO MONTH Data Type 9-17
- IS NOT NULL Operator 2-14
- IS NULL Operator 2-14
- iSQL*Plus 1-24

J

- Java 1-23
- Joining Tables 1-3, 4
 - Cartesian Product 4-4, 4-5
 - Equijoins 4-8, 4-27
 - Joining a Table to Itself 4-19
 - Joining More than Two Tables 4-13
 - Joining When there is No Matching Record 4-34

J

- Joining Tables 1-3, 4
 - Left Table 4-32
 - Natural Joins 4-24, 4-26
 - Nonequijoins 4-14, 4-15
 - ON Clause 4-28, 4-29
 - Outer Join 4-17, 4-18
 - RIGHT Table 4-33
 - Three-Way Join 4-30

K

- Keywords 1-4, 1-7

L

- LAST_DAY Function 3-21
- LENGTH Function 3-11
- LEVEL Psuedocolumn 19-10
- Library Cache E-13
- LIKE Operator 2-12
- LIST Command C-11
- Literal Values 1-20
- Loading Scripts 1-32
- LOCALTIMESTAMP Function 16-8
- Locks 8-45
 - Exclusive Locks 8-46
- Logical Conditions 2-15
- Logical Subsets 11-4
- LogWriter (LGWR) E-6, E-8
- LOWER Function 3-9
- LPAD Function 3-11

M

MAX Function 5-6, 5-7

MERGE Statement 8-28, 8-29

 WHEN NOT MATCHED Clause 8-31

MIN Function 5-6, 5-7

MINUS Operator 15-14

MODIFY Clause 9-26

Modify Column 9-25

MONTHS_BETWEEN Function 3-6, 3-21

Multiple Column Subquery 6-7, 18-2, 18-8

Multiple-row Function 3-4

Multiple-row Subquery 6-2, 6-7, 6-14, 18-6

Multitable Inserts 20-2, 20-5, 20-7

N

Naming Conventions for Tables 9-4

Naming Indexes 20-2

Natural Joins 4-24, 4-26

Nested Queries 6-4, 18-4

Nested Functions 3-42

NEXT_DAY Function 3-21

NEXTVAL Psuedocolumn 9-7, 12-8

Nonequi Joins 4-14, 4-15

Nonpairwise Comparisons 18-7

Non-unique Indexes 12-16

NOT EXISTS Operator 18-20

NOT IN Operator 18-20

NOT NULL Constraint 10-7

NULL 1-14, 1-15, 2-14, I-19

NULLIF Function 3-48

Number Functions 3-13

NVL Function 3-45, 3-46, 5-5, 5-12

NVL2 Function 3-47

O

- Object Privileges 13-2
- Object Relational Database Management System (ORDBMS) I-2, I-7, I-12
- Object-oriented Programming I-7
- ON Clause 4-28, 4-29
- ON DELETE CASCADE Clause 10-15
- ON DELETE SET NULL Clause 10-15
- On Line Transaction Processing I-8
- OR REPLACE Clause 11-12
- Oracle Instance E-3, E-7, I-23
- Oracle9i Application Server I-4
- Oracle9i Database I-4
- ORDER BY Clause **2**, 15-20
 - Default Sort Order 2-23
- Order of Precedence 1-12
- ORGANIZATION EXTERNAL Clause 20-18, 20-19
- Outer Join 4-17, 4-18
- Outer Query 6-5, 18-5

P

- Pairwise Comparisons 18-7
- Parameter File E-6
- Parent-child Relationship 19-4
- Password File E-6
- Pivoting INSERT 20-7, 20-15
- Primary Key 10-11
- PRIOR Clause 19-7
- Privileges **13**
 - Object Privileges 13-2
- Process Monitor E-8
- Program Global Area E-16
- Projection 1-3
- PUBLIC Keyword 13-5

R

Read Consistency 8-43, 8-44
READ ONLY Constraint 11-19
REM Command 7-21
REFERENCE Constraint 10-13, 10-15
Referential Integrity Constraint 10-13
REJECT LIMIT Clause 20-19
Relational Database Management System (RDBMS) 1-2, 1-13, 1-14
Relationships 1-16
RENAME Command 9-28
Restricting Rows 2-2
Retrieving Data from a View 11-10
Returning a Value 3-3
REVOKE Command 13-17
ROLLBACK Statement 8-2, 8-33, 8-35, 8-38, 8-41, E-20
Rollback Segment E-20
ROLLUP Clause 17-2, 17-6, 17-7, 17-8
Root Node 19-10
ROUND Function 3-14, 3-21, 3-23
Row 1-19, 17-8
ROWNUMBER Psuedocolumn
RR Date Format 3-41
Rules of Precedence 1-13, 2-19

Index

S

- SAVE Command C-14
- SAVEPOINT Statement 8-2, 8-35, 8-36
- Scalar Subquery 18-11
- Schema 9-6, 13-4
- Script or Command Files 7-20, 7-22, C-2
 - Creating Scripts 1-26
 - Loading Scripts 1-32
- Search 2-12
- SELECT Statement 1
- Selection 1-3
- Sequences 9-13, 12
 - Caching Sequence Values 12-11
 - CREATE SEQUENCE Statement 12-5
 - CURRVAL 9-7, 12-8
 - CYCLE Clause 12-6
 - Generating Unique Numbers 12-3
 - NEXTVAL 9-7, 12-8
- Server Architecture E-2
- SESSIONTIMEZONE Function 16-9
- SET Command 7-12
- SET Clause 8-15
- SET Operators 15-2, 15-3
- SET TIME_ZONE Clause 16-9
- SET UNUSED Clause 9-28
- SET VERIFY ON Command 7-7
- Sets of Rows 5-3
- Shared Global Area I-23, E-7
- Shared SQL Area E-14
- Single Ampersand Substitution 7-4

S

Single Row Function 3-4

Single Row Operators 6-8

Single Row Subqueries 6-2, 6-7

SMON Process E-8

SOME Operator 6-15

Sorting Results with the ORDER BY Clause **2**

 Default Sort Order 2-23

Spool File D-5

Structured Query Language (SQL) 1-2, 1-21, 1-22, 1-2, 1-24, 1-25

SQL Buffer C-3

SQL Scripts D-2

SQL*Plus **C**

SQL*Plus Commands C-2

SQL*Plus Script File 7-3

SQL: 1999 Compliance 4-6, 4-22, 4-30

START Command C-14

START WITH Clause 19-5, 19-6

Statement 1-4

Statement Level Rollback 8-42

STDDEV Function 5-7

S

- Subqueries 6, 8-16, 8-21, 8-23, 9-18, 11-21, 18-2, 18-3, 18-10
 - AS Subquery Clause 9-20
 - Correlated Subquery 18-2, 18-13, 18-14, 18-15, 18-21, 18-24
 - Correlated UPDATE 18-22
 - FROM Clause Query 11-21, 18-2, 18-10
 - Group Functions in a Subquery 6-10
 - Inner Query 6-3, 6-4, 6-5, 18-5
 - Multiple Column Subquery 6-7, 18-2, 18-8
 - Multiple-row Subquery 6-2, 6-7, 6-14, 18-6
 - Nested Queries 6-4, 18-4
 - No Rows Returned from the Subquery 6-13
 - Outer Query 6-5, 18-5
 - Placement of the Subquery 6-4
 - Scalar Subquery 18-11
 - Single Row Subqueries 6-2, 6-7
- Subsets, Logical 11-4
- Substitution Variables 7-2, 7-3
- SUBSTR Function 3-11
- SUM Function 5-6, 5-7
- Summary Results for Groups of Rows 5-18
- Superaggregate Rows 17-7, 17-8, 17-9
- SYS Function 9-9
- Synonym 9-3, 12-2, 12-3, 12-23, 13-3
- SYSDATE Function 3-18, 3-20, 9-7
- System Development Life Cycle I-10
- System Global Area I-23, E-3, E-8

T

Table Aliases 4-12
Table Level Constraints 10-8
Table Prefixes 4-11
Three-Way Join 4-30
Time Zone 16-3
TIMESTAMP Data Type 9-16
 TIMESTAMP WITH TIME ZONE 9-15
 TIMESTAMP WITH LOCAL TIME 9-16
 INTERVAL YEAR TO MONTH 9-17
TIMEZONE_ABBR 16-10
TIMEZONE_REGION 16-10
TO_CHAR Function 3-31, 3-37, 3-39
TO_DATE Function 3-39
TO_NUMBER Function 3-39
TO_TIMESTAMP Function 16-12
TO_YMINTERVAL Function 16-13
Top-*n* Analysis 11-2, 11-22, 11-23, 11-24
Transactions 8-32
Tree Structured Report **19**
TRIM Function 3-11
TRUNC Function 3-15, 3-21, 3-23
TRUNCATE TABLE Statement 9-31
TTITLE Command 7-19
Tuple I-19
TYPE ACCESS_DRIVER_TYPE 20-19
TZOFFSET Function 16-14

U

- UNDEFINE Command 7-11
- UNION Operator 15-7, 15-8, 15-11
- UNION Operator 15-10, 15-11
- UNIQUE Constraint 10-9, 10-10
- Unique Identifier I-18
- Unique Index 10-10, 12-6
- UPDATE Statement 8, 13-14
 - SET Clause 8-15
 - Correlated UPDATE 18-22
- UPPER Function 3-9, 3-10
- Users - (Creating) 13-6
- USER Function 9-7
- User Process E-10
- USER_CATALOG Dictionary View 9-10
- USER_COL_COMMENTS Dictionary View 9-32
- USER_COL_PRIVS_MADE Dictionary View D-4
- USER_CONS_COLUMNS Dictionary View 10-19, 10-25
- USER_CONSTRAINTS Dictionary View 10-4, 10-19, 10-24
- USER_DB_LINKS Dictionary View 13-19
- USER_INDEXES Dictionary View 12-20
- USER_OBJECTS Dictionary View 9-10, D-4
- USER_SEQUENCES Dictionary View 12-7
- USER_TAB_COMMENTS Dictionary View 9-30
- USER_TAB_PRIVS_MADE Dictionary View D-4
- USER_TABLES Dictionary View 9-10, D-4
- USER_UNUSED_COL_TABS Dictionary View 9-28
- USING Clause 4-26, 13-20
- UTC (Coordinated Universal Time) 9-15

V

VALUES Clause 8-5

Variance 5-7

VERIFY Command 7-7

Views 9-3, **11**

 Guidelines for Creating a View 11-8

 Inline Views 11-2, 11-21

 OR REPLACE Clause 11-12

 Retrieving Data from a View 11-10

 Simple and Complex 11-6

 USING Clause 4-26

 WITH READ ONLY Clause 11-18

V\$TIMEZONE_NAME Dictionary View 16-11

W

WHEN NOT MATCHED Clause 8-31

WHERE Clause **2**

 Restricting Rows 2-2

Wildcard Symbol 2-12

WITH Clause 18-2, 18-26

WITH CHECK OPTION Clause 8-25, 11-17, 13-13, 13-14, 13-15, 13-18

WITH READ ONLY Clause 11-18

X

XML 1-23

Y

Year 2000 Compliance 3-17

