Fidelity INVESTMENTS

WORKING WITH RELATIONAL DATABASES

EXERCISE MANUAL SOLUTIONS

Fidelity LEAP
Technology Immersion Program

© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

This page intentionally left blank.

© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Table of Contents

Chapter 1: What Is Structured Query Language? 1
Exercise 1.1: Using SQL Developer 1

Chapter 2: SQL Query Syntax 3
Exercise 2.1: Selecting Data 3

Chapter 3: SQL Scalar Functions 7
Exercise 3.1: Using Scalar Functions 7

Chapter 4: SQL Joins 11
Exercise 4.1: Working with `INNER JOIN`s 11
Exercise 4.2: Using `OUTER JOIN`s 13

Chapter 5: Additional SQL Functions 15
Exercise 5.1: Additional SQL Functions 15

Chapter 6: Data Manipulation Language 17
Exercise 6.1: Manipulating Data 17

Chapter 11: Aggregating Information 19
Exercise 11.1: Using the Aggregate Functions 19
Exercise 11.2: `GROUP BY` and `HAVING` 20
Exercise 11.3: Using Subqueries 23

Chapter 12: Set Operators 25
Exercise 12.1: Set Operators 25

Chapter 13: Programming with PL/SQL 27
Exercise 13.1: Building Anonymous Blocks 27
Exercise 13.2: Using Cursors 31

Chapter 14: Creating Stored Procedures, Functions, and Packages 33
Exercise 14.1: Stored Procedures, Functions, and Packages 33

Chapter 15: Testing PL/SQL 39
Exercise 15.1: Writing PL/SQL Tests With utPLSQL 39
Exercise 15.2: Testing Updates With utPLSQL 41

Chapter 16: Creating Triggers 47
Exercise 16.1: Working with Triggers 47

Chapter 17: Data Definition Language 57
Exercise 17.1: Table Management 57

© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

This page intentionally left blank.

© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 1: What Is Structured Query Language?

Exercise 1.1: Using SQL Developer

```
 7.  7

 9.  SELECT * FROM locations;

12.  SELECT * FROM countries;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

This page intentionally left blank.

© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 2: SQL Query Syntax

Exercise 2.1: Selecting Data

```
-- 1
SELECT dname, deptno
FROM   dept;

-- 2
SELECT *
FROM   dept;

-- 3
SELECT dname  AS "Name",
       deptno AS "DEPT#",
       loc    AS "Dept Location"
FROM   dept;

-- 4
SELECT deptno
FROM   emp;

-- 5
SELECT DISTINCT deptno
FROM   emp;

-- 6
SELECT deptno, job
FROM   emp;

-- 7
SELECT DISTINCT deptno, job
FROM   emp;

-- 8
SELECT ename
FROM   emp
WHERE  deptno = 30;

-- 9
SELECT ename
FROM   emp
WHERE  hiredate = DATE '1981-12-17';
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 10
SELECT ename
FROM   emp
WHERE  hiredate >= DATE '1981-12-17';

-- Other safe ways of specifying the date
SELECT ename
FROM   emp
WHERE  hiredate >= TO_DATE('12-17-1981', 'MM-DD-YYYY');

SELECT ename
FROM   emp
WHERE  hiredate >= TO_DATE( '19811217', 'YYYYMMDD' );

-- 11
SELECT ename
FROM   emp
WHERE  job = 'clerk';

-- 12
SELECT ename
FROM   emp
WHERE  job = 'CLERK';

-- 13
SELECT ename
FROM   emp
WHERE  sal > 2500;

-- 14
SELECT ename
FROM   emp
WHERE  sal BETWEEN 1000 AND 1600;

-- 15
SELECT ename
FROM   emp
WHERE  ename LIKE '%ER%';

-- 16
SELECT empno, ename
FROM   emp
WHERE  comm IS NULL;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 17
SELECT empno, ename, comm
FROM   emp
ORDER BY
       comm;

-- 18
SELECT empno, ename, comm
FROM   emp
ORDER BY
       comm DESC;

-- 19
SELECT empno, ename, comm
FROM   emp
ORDER BY
       comm DESC NULLS LAST;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

This page intentionally left blank.

© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 3: SQL Scalar Functions

Exercise 3.1: Using Scalar Functions

```
-- 1
SELECT first_name,
       last_name,
       TO_CHAR(salary, '$99,999') AS salary
FROM   employees
WHERE  department_id = 30;

-- 2
SELECT first_name,
       last_name,
       TO_CHAR(hire_date, 'YYYY-MM-DD') AS "Date Hired"
FROM   employees
WHERE  department_id = 30;

-- 3
SELECT first_name,
       last_name,
       ROUND(salary, -3) AS RDSAL,
       TRUNC(salary, -3) AS TSAL,
       salary
FROM   employees
WHERE  department_id = 30;

-- 4
SELECT LOWER(first_name) AS LNAME,
       UPPER(last_name) AS UNAME
FROM   employees
WHERE  department_id = 30
ORDER BY
       first_name, last_name;

-- 5
SELECT SUBSTR(first_name, 1, 1)
        || '. '
        || last_name AS NAME
FROM   employees
WHERE  department_id = 30
ORDER BY
       NAME;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 6
SELECT street_address,
       LTRIM(street_address, '0123456789 -') AS "Street Name"
FROM   locations
ORDER BY
       "Street Name";

-- Other approaches, using regular expressions (not covered)
SELECT street_address,
       REGEXP_SUBSTR(street_address, '[[:alpha:]].*')
              AS "Street Name"
FROM   locations
ORDER BY
       "Street Name";

SELECT street_address,
       REGEXP_REPLACE(street_address, '^[-[:digit:] ]*', '')
              AS "Street Name"
FROM   locations
ORDER BY
       "Street Name";

-- 7
SELECT street_address,
       LENGTH(street_address) AS "Street Length"
FROM   locations
ORDER BY
       "Street Length";

-- 8
SELECT location_id,
       street_address,
       city,
       state_province
FROM   locations
WHERE  INSTR(UPPER(street_address), 'RUE') > 0
OR     INSTR(UPPER(street_address), 'RUA') > 0
ORDER BY
       location_id DESC;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
SELECT location_id,
       street_address,
       city,
       state_province
FROM   locations
WHERE  UPPER(street_address) LIKE '%RUE%'
OR     UPPER(street_address) LIKE '%RUA%'
ORDER BY
       location_id DESC;

-- Using regular expression
SELECT location_id,
       street_address,
       city,
       state_province
FROM   locations
WHERE  REGEXP_LIKE(street_address, 'RU[AE]', 'i')
ORDER BY
       location_id DESC;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

This page intentionally left blank.

© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 4: SQL Joins

Exercise 4.1: Working with `INNER JOINS`

```
-- 1
SELECT l.city,
       l.location_id,
       d.department_name
FROM   departments d
JOIN   locations   l
ON     d.location_id = l.location_id;

-- 2
SELECT c.country_name,
       l.city
FROM   locations l
JOIN   countries c
ON     l.country_id = c.country_id;

-- 3
SELECT c.country_name,
       l.city,
       d.department_name
FROM   departments d
JOIN   locations   l
ON     d.location_id = l.location_id
JOIN   countries   c
ON     l.country_id  = c.country_id;

-- 4
SELECT e.employee_id,
       e.first_name,
       e.last_name,
       jh.job_id
FROM   employees   e
JOIN   job_history jh
ON     e.employee_id = jh.employee_id
ORDER BY
       e.employee_id;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 5
SELECT j.job_title,
       jh.employee_id,
       jh.start_date
FROM   job_history jh
JOIN   jobs        j
ON     jh.job_id     = j.job_id
WHERE  jh.start_date > DATE '1998-01-01';

-- 6
SELECT j.job_title,
       jh.employee_id,
       jh.start_date,
       e.first_name,
       e.last_name
FROM   job_history jh
JOIN   jobs        j
ON     jh.job_id      = j.job_id
JOIN   employees   e
ON     jh.employee_id = e.employee_id;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Exercise 4.2: Using `OUTER JOINS`

```
-- 1
SELECT e.employee_id,
       e.first_name,
       e.last_name,
       jh.job_id
FROM   employees   e
LEFT OUTER JOIN
       job_history jh
ON     e.employee_id = jh.employee_id
ORDER BY
       e.employee_id;

-- 2
SELECT j.job_title,
       jh.employee_id
FROM   jobs        j
LEFT OUTER JOIN
       job_history jh
ON     j.job_id = jh.job_id;

-- 3
SELECT j.job_title,
       jh.employee_id
FROM   jobs        j
LEFT OUTER JOIN
       job_history jh
ON     j.job_id     = jh.job_id
WHERE  j.min_salary > 9000;

-- 4
SELECT j.job_title,
       jh.employee_id,
       jh.start_date
FROM   jobs        j
LEFT OUTER JOIN
       job_history jh
ON     j.job_id = jh.job_id
AND    jh.start_date > DATE '1998-01-01';
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 5
SELECT j.job_title,
       jh.employee_id,
       jh.start_date,
       e.first_name,
       e.last_name
FROM   jobs        j
LEFT OUTER JOIN
       job_history jh
ON     j.job_id       = jh.job_id
LEFT OUTER JOIN
       employees   e
ON     jh.employee_id = e.employee_id
ORDER BY e.employee_id;

-- BONUS

-- 6
SELECT j.job_title,
       jh.employee_id,
       jh.start_date,
       e.first_name,
       e.last_name
FROM   employees   e
LEFT OUTER JOIN
       job_history jh
ON     jh.employee_id = e.employee_id
LEFT OUTER JOIN
       jobs        j
ON     j.job_id       = jh.job_id;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 5: Additional SQL Functions

Exercise 5.1: Additional SQL Functions

```
-- 1
-- using MONTHS_BETWEEN
SELECT department_id,
       first_name,
       last_name,
       hire_date,
       NVL(commission_pct, 0) AS commission
FROM   employees
WHERE  manager_id = 100
AND    ABS(MONTHS_BETWEEN(hire_date, DATE '2007-01-01'))
               <= 24
ORDER BY
       department_id, hire_date;

-- using BETWEEN
SELECT department_id,
       first_name,
       last_name,
       hire_date,
       NVL(commission_pct, 0) AS commission
FROM   employees
WHERE  manager_id = 100
AND    hire_date
         BETWEEN ADD_MONTHS(DATE '2007-01-01', -24)
         AND     ADD_MONTHS(DATE '2007-01-01',  24)
ORDER BY
       department_id, hire_date;

-- 2
SELECT department_id,
       first_name,
       last_name,
       hire_date,
       COALESCE(commission_pct, 0) AS commission
FROM   employees
WHERE  manager_id = 100
AND    ABS(MONTHS_BETWEEN(hire_date, DATE '2007-01-01'))
               <= 24
ORDER BY
       ABS(MONTHS_BETWEEN(hire_date, DATE '2007-01-01'));
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

This page intentionally left blank.

© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 6: Data Manipulation Language

Exercise 6.1: Manipulating Data

```
-- 1.
SELECT *
FROM   regions;

-- 2
INSERT INTO
       regions
       ( region_id, region_name )
VALUES ( 5, 'Central America' );

-- 3
SELECT *
FROM   regions;

-- 4
INSERT INTO
       regions
       ( region_id, region_name )
VALUES ( 6, 'South America' );

-- 5
SELECT *
FROM   regions;

-- 6
UPDATE regions
SET    region_name = 'South and Central America'
WHERE  region_name = 'Central America';

-- 7
SELECT *
FROM   regions;

-- 8
DELETE regions
WHERE  region_id = 6;

-- 9
SELECT *
FROM   regions;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 10
ROLLBACK;

SELECT *
FROM   regions;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 11: Aggregating Information

Exercise 11.1: Using the Aggregate Functions

```
-- 1
SELECT COUNT(*) AS "Count"
FROM   emp;

-- 2
SELECT empno, ename, sal, comm
FROM   emp
ORDER BY
       sal;

-- 3
SELECT COUNT(sal) AS "Count",
       COUNT(DISTINCT sal) AS "CDistinct"
FROM   emp;

-- 4
SELECT COUNT(comm) AS "Count",
       SUM(comm)   AS "Sum",
       AVG(comm)   AS "Average"
FROM emp;

-- 5
SELECT COUNT(comm) AS "Count",
       SUM(comm)   AS "Sum",
       AVG(comm)   AS "Average",
       ROUND(AVG(COALESCE(comm, 0)), 3)
                   AS "Average of all Records"
FROM   emp;

-- 6
SELECT MAX(sal) AS "Maximum Salary",
       MIN(sal) AS "Minimum Salary"
FROM   emp;

-- 7
SELECT MAX(hiredate) "Maximum Hire Date",
       MIN(hiredate) "Minimum Hire Date"
FROM   emp;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Exercise 11.2: GROUP BY and HAVING

```
-- 1
SELECT  department_id,
        COUNT(*),
        MIN(salary),
        MAX(salary),
        SUM(salary) AS "Total Salary",
        ROUND(AVG(salary ), 0) AS "Avg Salary"
FROM    employees
GROUP BY
        department_id
-- optional, but achieves same output sequence
ORDER BY
        department_id;

-- 2
SELECT  department_id,
        COUNT(*),
        MIN(salary),
        MAX(salary),
        SUM(salary) AS "Total Salary",
        ROUND(AVG(salary), 0) AS "Avg Salary"
FROM    employees
GROUP BY
        department_id
ORDER BY
        AVG(salary);

-- 3
SELECT  department_id,
        COUNT(*),
        MIN(salary),
        MAX(salary),
        SUM(salary) AS "Total Salary",
        ROUND(AVG(salary), 0) AS "Avg Salary",
        ROUND(AVG(salary) - MIN(salary), 0) AS "Below Avg"
FROM    employees
GROUP BY
        department_id
ORDER BY
        AVG(salary) - MIN(salary) DESC;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 4
SELECT  manager_id,
        COUNT(*),
        MIN(salary),
        MAX(salary),
        SUM(salary) AS "Total Salary",
        ROUND(AVG(salary), 0) AS "Avg Salary",
        ROUND(AVG(salary) - MIN(salary), 0) AS "Below Avg"
FROM    employees
GROUP BY
        manager_id
ORDER BY
        AVG(salary) - MIN(salary) DESC;

-- 5
SELECT  department_id AS deptid,
        manager_id AS mgrid,
        COUNT(*),
        MIN(salary),
        MAX(salary),
        SUM(salary) AS "Total Salary",
        ROUND(AVG(salary), 0) AS "Avg Salary",
        ROUND(AVG(salary) - MIN(salary), 0) AS "Below Avg"
FROM    employees
GROUP BY
        department_id, manager_id
ORDER BY
        AVG(salary) - MIN(salary) DESC;

-- 6
SELECT  department_id AS deptid,
        manager_id AS mgrid,
        COUNT(*),
        MIN(salary),
        MAX(salary),
        SUM(salary) AS "Total Salary",
        ROUND(AVG(salary), 0) AS "Avg Salary",
        ROUND(AVG(salary) - MIN(salary), 0) AS "Below Avg"
FROM    employees
GROUP BY
        department_id, manager_id
HAVING  COUNT(*) > 5
ORDER BY
        AVG(salary) - MIN(salary) DESC;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- Bonus

-- 7
SELECT  TRUNC(department_id, -2) AS "Depts by 100s",
        SUM(salary),
        AVG(salary),
        COUNT(*)
FROM    employees
GROUP BY
        TRUNC(department_id, -2)
ORDER BY
        TRUNC(department_id, -2);

-- 8

-- Use two aggregate expressions (GROUP BY only affects the
first)
SELECT ROUND(AVG(AVG(salary))) AS "Avg of Dept Avgs"
FROM   employees
GROUP BY
       department_id;

-- Use subquery as source of query (per next section)
SELECT ROUND(AVG("Average")) AS "Avg of Dept Avgs"
FROM   (
           SELECT AVG(salary) AS "Average"
           FROM   employees
           GROUP BY
                  department_id
        );

-- 9
SELECT ROUND(AVG( salary ))
FROM   employees;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Exercise 11.3: Using Subqueries

```
--1
SELECT department_id,
       department_name
FROM   departments
WHERE  department_id IN (
         SELECT department_id
         FROM employees
       )
ORDER BY
       department_id;

--2
SELECT employee_id,
       first_name,
       last_name,
       salary
FROM   employees
WHERE  salary > (
         SELECT AVG(salary)
         FROM employees
       )
ORDER BY
       salary desc;

--3
SELECT employee_id,
       first_name,
       last_name,
       salary
FROM   employees
WHERE  salary = (
         SELECT MAX(salary)
         FROM employees
       );
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
--4
SELECT employee_id,
       first_name,
       last_name,
       salary,
       commission_pct
FROM   employees
WHERE  salary > (
         SELECT AVG(salary)
         FROM employees
       )
AND    commission_pct > (
         SELECT AVG(commission_pct)
         FROM employees
       )
ORDER BY
       last_name;

-- BONUS

--5
SELECT employee_id,
       first_name,
       last_name
FROM   employees
WHERE  department_id IN (
         SELECT department_id
         FROM   departments
         WHERE location_id IN (
           SELECT location_id
           FROM   locations
           WHERE  city = 'London'
         )
       );
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 12: Set Operators

Exercise 12.1: Set Operators

```
-- 1
SELECT 'Employees who earn commission' AS "Type",
       COUNT(*) AS "Count"
FROM   employees
WHERE  commission_pct IS NOT NULL
UNION ALL
SELECT 'Employees who do not earn commission',
       COUNT(*)
FROM   employees
WHERE  commission_pct IS NULL;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

This page intentionally left blank.

© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 13: Programming with PL/SQL

Exercise 13.1: Building Anonymous Blocks

```
SET SERVEROUTPUT ON;

-- 1 to 6
SELECT *
FROM   employees
WHERE  last_name IN ('Austin', 'Lee', 'King');

DECLARE
    emp_rec employees%ROWTYPE;
BEGIN
    SELECT *
    INTO   emp_rec
    FROM   employees
    WHERE  last_name = 'Austin';

    DBMS_OUTPUT.PUT_LINE('old salary: ' || emp_rec.salary);

    IF COALESCE(emp_rec.commission_pct, 0) = 0 THEN
        emp_rec.salary := emp_rec.salary + 500;
    ELSIF emp_rec.commission_pct < 0.02 THEN
        emp_rec.salary := emp_rec.salary + 300;
    ELSE
        emp_rec.salary := emp_rec.salary + 100;
    END IF;

    DBMS_OUTPUT.PUT_LINE('new salary: ' || emp_rec.salary);

    UPDATE  employees
    SET     salary = emp_rec.salary
    WHERE   employee_id = emp_rec.employee_id;
END;
/

SELECT *
FROM   employees
WHERE  last_name IN ('Austin', 'Lee', 'King');

-- 8
ROLLBACK;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 9
DECLARE
    emp_rec employees%ROWTYPE;
BEGIN
    SELECT *
    INTO   emp_rec
    FROM   employees
    WHERE  last_name = 'Austin';

    DBMS_OUTPUT.PUT_LINE('old salary: ' || emp_rec.salary);

    CASE
        WHEN COALESCE(emp_rec.commission_pct, 0) = 0 THEN
            emp_rec.salary := emp_rec.salary + 500;
        WHEN emp_rec.commission_pct < 0.02 THEN
            emp_rec.salary := emp_rec.salary + 300;
        ELSE
            emp_rec.salary := emp_rec.salary + 100;
    END CASE;

    DBMS_OUTPUT.PUT_LINE('new salary: ' || emp_rec.salary);

    UPDATE  employees
    SET     salary = emp_rec.salary
    WHERE   employee_id = emp_rec.employee_id;
END;
/

-- 10
ROLLBACK;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- BONUS

-- case expression, and #11, 12 (exception handling)
DECLARE
    emp_rec employees%ROWTYPE;
BEGIN
    SELECT *
    INTO   emp_rec
    FROM   employees
    WHERE  last_name = 'King';

    DBMS_OUTPUT.PUT_LINE('old salary: ' || emp_rec.salary);

    emp_rec.salary := emp_rec.salary +
        CASE
            WHEN COALESCE(emp_rec.commission_pct, 0) = 0 THEN
                500
            WHEN emp_rec.commission_pct < 0.02 THEN
                300
            ELSE
                100
        END;

    DBMS_OUTPUT.PUT_LINE('new salary: ' || emp_rec.salary);

    UPDATE  employees
    SET     salary = emp_rec.salary
    WHERE   employee_id = emp_rec.employee_id;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20999, 'Employee does not
exist');
    WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR(-20999, 'Multiple employees
found');
    WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20999, 'Contact Support: ' ||
SQLERRM);
END;
/


ROLLBACK;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 13
SELECT *
FROM   employees
WHERE  last_name IN ( 'Austin', 'Lee', 'King', 'Smith',
                      'Howard' );


-- To show this could be done in a single UPDATE
UPDATE employees
SET    salary = salary +
         CASE
           WHEN COALESCE(commission_pct, 0) = 0 THEN
             500
           WHEN commission_pct < 0.02 THEN
             300
           ELSE
             100
         END
WHERE  last_name = 'Austin';

ROLLBACK;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Exercise 13.2: Using Cursors

```
SET SERVEROUTPUT ON;

SELECT *
FROM   employees
WHERE  hire_date < DATE '2003-01-01';

-- 1-8
DECLARE
    CURSOR emp_cur(
      in_hire_date DATE
    )
    IS
        SELECT *
        FROM   employees
        WHERE  hire_date < in_hire_date
        FOR UPDATE;
    emp_rec emp_cur%ROWTYPE;
    raise   employees.salary%TYPE := 5000;
    v_date  DATE                  := DATE '2003-01-01';
BEGIN
    OPEN emp_cur( v_date );
    LOOP
        FETCH emp_cur INTO emp_rec;
        EXIT WHEN emp_cur%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE('Updating '
                             || emp_rec.employee_id
                             || ' '
                             || emp_rec.last_name
                             || ' from '
                             || emp_rec.salary);

        UPDATE employees
        SET    salary = salary + raise
        WHERE CURRENT OF emp_cur;
    END LOOP;

    CLOSE emp_cur;
END;
/

SELECT *
FROM   employees
WHERE  hire_date < DATE '2003-01-01';
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 9
ROLLBACK;


-- 10
DECLARE
    CURSOR emp_cur(
      in_hire_date DATE
    )
    IS
        SELECT *
        FROM   employees
        WHERE  hire_date < in_hire_date
        FOR UPDATE;
    new_salary employees.salary%TYPE := 11000;
    v_date     DATE                  := DATE '2003-01-01';
BEGIN
    FOR emp_rec IN emp_cur( v_date ) LOOP
        DBMS_OUTPUT.PUT_LINE('Updating '
                             || emp_rec.employee_id
                             || ' '
                             || emp_rec.last_name
                             || ' from '
                             || emp_rec.salary);

        UPDATE employees
        SET    salary = new_salary
        WHERE CURRENT OF emp_cur;
    END LOOP;
END;
/

SELECT *
FROM   employees
WHERE  hire_date < DATE '2003-01-01';

-- 11
ROLLBACK;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 14:
Creating Stored Procedures, Functions, and Packages

Exercise 14.1: Stored Procedures, Functions, and Packages

```
SET SERVEROUTPUT ON;

-- 1-6
CREATE OR REPLACE PROCEDURE update_emp(
    parm_employee_id IN employees.employee_id%TYPE,
    parm_last_name   IN employees.last_name%TYPE,
    parm_email       IN employees.email%TYPE,
    parm_hire_date   IN employees.hire_date%TYPE,
    parm_job_id      IN employees.job_id%TYPE,
    parm_salary      IN employees.salary%TYPE
)
IS
    emp_count NUMBER(5);
BEGIN
    UPDATE employees
    SET    salary      = parm_salary
    WHERE  employee_id = parm_employee_id
    AND    last_name   = parm_last_name
    AND    hire_date   = parm_hire_date
    AND    job_id      = parm_job_id;

    IF SQL%NOTFOUND THEN
    -- No row was updated. There are two possible reasons: either the
    -- details don't match, or the employee doesn't exist. It is VERY
    -- important that you do not assume the UPDATE failed just because
    -- the employee doesn't exist

    -- There are a number of ways to structure this code
        SELECT COUNT(*)
        INTO   emp_count
        FROM   employees
        WHERE  employee_id = parm_employee_id;

        IF emp_count = 0 THEN
        -- Employee doesn't exist
            INSERT INTO
                   employees (
                       employee_id,
                       last_name,
                       email,
                       hire_date,
                       job_id,
                       salary
                   )
            VALUES (
                       parm_employee_id,
                       parm_last_name,
                       parm_email,
                       parm_hire_date,
                       parm_job_id,
                       parm_salary
                   );
            DBMS_OUTPUT.PUT_LINE( 'Inserted new employee' );
        ELSE
        -- Details don't match
            DBMS_OUTPUT.PUT_LINE( 'Employee already exists, details do not match' );
        END IF;
    ELSE
        DBMS_OUTPUT.PUT_LINE( 'Employee updated' );
    END IF;
END;
/
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 7
SELECT *
FROM   employees
WHERE  last_name IN ( 'Chen', 'Johnston' );

-- This update should succeed
BEGIN
    update_emp( 110,
                'Chen',
                'JCHEN',
                DATE '2005-09-28',
                'FI_ACCOUNT',
                2000 );
END;
/

-- This update should fail because the details don't match
BEGIN
    update_emp( 110,
                'XXXX',
                'JCHEN',
                DATE '2005-09-28',
                'FI_ACCOUNT',
                5000 );
END;
/

-- This update should insert Johnson instead
BEGIN
    update_emp( 999,
                'Johnston',
                'JJ',
                DATE '2018-06-08',
                'IT_PROG',
                20000 );
END;
/

-- 8
ROLLBACK;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- BONUS

-- 9
DROP PROCEDURE update_emp;

-- 10-12
CREATE OR REPLACE PACKAGE pack_employee
IS
    PROCEDURE update_emp(
        parm_employee_id   IN employees.employee_id%TYPE,
        parm_last_name     IN employees.last_name%TYPE,
        parm_email         IN employees.email%TYPE,
        parm_hire_date     IN employees.hire_date%TYPE,
        parm_job_id        IN employees.job_id%TYPE,
        parm_salary        IN employees.salary%TYPE,
        parm_department_id IN employees.department_id%TYPE
    );
END pack_employee;
/

-- 13
CREATE OR REPLACE PACKAGE BODY pack_employee
IS

-- 14-16
    FUNCTION get_manager(
        parm_department_id IN employees.department_id%TYPE
    )
    RETURN NUMBER
    IS
        dept_count NUMBER(5);
        mgr_id     departments.manager_id%TYPE;
    BEGIN
        SELECT COUNT(*)
        INTO   dept_count
        FROM   departments
        WHERE  department_id = parm_department_id;

        -- Here we distinguish between the department not existing
        -- and there being no manager. When we use the function, we
        -- don't make that distinction, so we could simplify this.
        IF dept_count = 0 THEN
            RETURN 0;
        ELSE
            SELECT manager_id
            INTO   mgr_id
            FROM   departments
            WHERE  department_id = parm_department_id;

            RETURN mgr_id;
        END IF;

        RETURN NULL;
    END get_manager;

-- 17
    PROCEDURE update_emp(
        parm_employee_id   IN employees.employee_id%TYPE,
        parm_last_name     IN employees.last_name%TYPE,
        parm_email         IN employees.email%TYPE,
        parm_hire_date     IN employees.hire_date%TYPE,
        parm_job_id        IN employees.job_id%TYPE,
        parm_salary        IN employees.salary%TYPE,
        parm_department_id IN employees.department_id%TYPE
    )
    IS
        emp_count NUMBER(5);
    BEGIN
        UPDATE employees
        SET    salary      = parm_salary
        WHERE  employee_id = parm_employee_id
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
        AND    last_name   = parm_last_name
        AND    hire_date   = parm_hire_date
        AND    job_id      = parm_job_id;

        IF SQL%NOTFOUND THEN
            SELECT COUNT(*)
            INTO   emp_count
            FROM   employees
            WHERE  employee_id = parm_employee_id;

            IF emp_count = 0 THEN
            -- Don't care whether the department doesn't exist, or no manager
                IF COALESCE(get_manager(parm_department_id), 0) = 0 THEN
                    DBMS_OUTPUT.PUT_LINE('Department '
                                    || parm_department_id
                                    || ' does not exist, or has no manager');
                ELSE
                    INSERT INTO
                           employees (
                               employee_id,
                               last_name,
                               email,
                               hire_date,
                               job_id,
                               salary,
                               department_id
                           )
                    VALUES (
                               parm_employee_id,
                               parm_last_name,
                               parm_email,
                               parm_hire_date,
                               parm_job_id,
                               parm_salary,
                               parm_department_id
                           );
                    DBMS_OUTPUT.PUT_LINE('Inserted new employee');
                END IF;
            ELSE
                DBMS_OUTPUT.PUT_LINE('Employee already exists, details do not match');
            END IF;
        ELSE
            DBMS_OUTPUT.PUT_LINE('Employee updated');
        END IF;
    END update_emp;

-- 18
END pack_employee;
/
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 20
SELECT *
FROM   employees
WHERE  last_name IN ( 'Chen', 'Johnston' );

-- Chen, details match, update salary, success
BEGIN
    pack_employee.update_emp( 110,
                'Chen',
                'JCHEN',
                DATE '2005-09-28',
                'FI_ACCOUNT',
                2000,
                100 );
END;
/

-- Chen, details do not match, fail
BEGIN
    pack_employee.update_emp( 110,
                'XXXX',
                'JCHEN',
                DATE '2005-09-28',
                'FI_ACCOUNT',
                2000,
                100 );
END;
/

-- Johnston, dept does not exist, fail
BEGIN
    pack_employee.update_emp( 999,
                'Johnston',
                'JJ',
                DATE '2018-06-08',
                'IT_PROG',
                20000,
                999 );
END;
/

-- Johnston, department has no manager, fail
BEGIN
    pack_employee.update_emp( 999,
                'Johnston',
                'JJ',
                DATE '2018-06-08',
                'IT_PROG',
                20000,
                120 );
END;
/

-- Johnston, valid dept, insert new employee, success
BEGIN
    pack_employee.update_emp( 999,
                'Johnston',
                'JJ',
                DATE '2018-06-08',
                'IT_PROG',
                20000,
                100 );
END;
/

ROLLBACK;

DROP PACKAGE pack_employee;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

This page intentionally left blank.

© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 15: Testing PL/SQL

Exercise 15.1: Writing PL/SQL Tests With utPLSQL

Function

```
CREATE OR REPLACE FUNCTION func_check_salary(
    job_id IN jobs.job_id%TYPE,
    salary IN jobs.min_salary%TYPE
)
RETURN BOOLEAN
IS
    job_count NUMBER(5);
BEGIN
    IF job_id IS NULL OR salary IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'parameters may not be null');
    END IF;

    SELECT COUNT(*)
    INTO   job_count
    FROM   jobs j
    WHERE  j.job_id = func_check_salary.job_id
    AND    func_check_salary.salary BETWEEN min_salary AND max_salary;

    IF job_count <> 1 THEN
        RETURN FALSE;
    END IF;

    RETURN TRUE;
END func_check_salary;
/
```

Tests

```
CREATE OR REPLACE PACKAGE test_check_salary
IS
    --%suite(Tests for func_check_salary)

    --%test(Returns true if salary is in range)
    PROCEDURE salary_in_range;

    --%test(Returns false if salary is below minimum)
    PROCEDURE salary_below_minimum;

    --%test(Returns false if salary is above maximum)
    PROCEDURE salary_above_maximum;

    --%test(Checks a second job_id)
    PROCEDURE check_second_job_id;

    --%test(Throws exception for NULL job_id)
    --%throws(-20001)
    PROCEDURE exception_for_null_job_id;

    --%test(Throws exception for NULL salary)
    --%throws(-20001)
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
    PROCEDURE exception_for_null_salary;

    --%test(Returns false if job_id does not exist)
    PROCEDURE job_id_does_not_exist;

END test_check_salary;
/

CREATE OR REPLACE PACKAGE BODY test_check_salary
IS

    PROCEDURE salary_in_range
    IS
    BEGIN
        ut.expect(func_check_salary('AD_VP', 20000)).to_equal(TRUE);
    END salary_in_range;

    PROCEDURE salary_below_minimum
    IS
    BEGIN
        ut.expect(func_check_salary('AD_VP', 10000)).to_equal(FALSE);
    END salary_below_minimum;

    PROCEDURE salary_above_maximum
    IS
    BEGIN
        ut.expect(func_check_salary('AD_VP', 40000)).to_equal(FALSE);
    END salary_above_maximum;

    PROCEDURE check_second_job_id
    IS
    BEGIN
        ut.expect(func_check_salary('AD_ASST', 4000)).to_equal(TRUE);
    END check_second_job_id;

    PROCEDURE exception_for_null_job_id
    IS
        result BOOLEAN;
    BEGIN
        result := func_check_salary(NULL, 4000);
    END exception_for_null_job_id;

    PROCEDURE exception_for_null_salary
    IS
        result BOOLEAN;
    BEGIN
        result := func_check_salary('AD_VP', NULL);
    END exception_for_null_salary;

    PROCEDURE job_id_does_not_exist
    IS
    BEGIN
        ut.expect(func_check_salary('XX_XXX', 4000)).to_equal(FALSE);
    END job_id_does_not_exist;

END test_check_salary;
/
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Exercise 15.2: Testing Updates With utPLSQL

Procedure

```
CREATE OR REPLACE PROCEDURE proc_update_salary(
    employee_id IN employees.employee_id%TYPE,
    salary IN employees.salary%TYPE
)
IS
    job_id employees.job_id%TYPE;
BEGIN
    -- the function already throws the exception if the salary is NULL
    IF employee_id IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'employee_id may not be NULL');
    END IF;

    SELECT job_id
    INTO   proc_update_salary.job_id
    FROM   employees
    WHERE  employee_id = proc_update_salary.employee_id;

    IF func_check_salary(job_id => job_id,
            salary => salary) THEN
        UPDATE employees e
        SET    salary        = proc_update_salary.salary
        WHERE  e.employee_id = proc_update_salary.employee_id;
    END IF;

-- If the employee does not exist, just ignore the error
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        NULL;
END proc_update_salary;
/
```

Tests

```
CREATE OR REPLACE PACKAGE test_update_salary
IS
    --%suite(Tests for proc_update_salary)

    --%test(Updates if salary is in range)
    PROCEDURE salary_in_range;

    --%test(Does not update if salary is below minimum)
    PROCEDURE salary_below_minimum;

    --%test(Returns false if salary is above maximum)
    PROCEDURE salary_above_maximum;

    --%test(Checks a second employee_id)
    PROCEDURE check_second_employee_id;

    --%test(Throws exception for NULL employee_id)
    --%throws(-20001)
    PROCEDURE exception_for_null_employee_id;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
    -- could use "throws", as above, but this is an alternative
    -- that allows us to check no data was updated
    --%test(Throws exception for NULL salary)
    PROCEDURE exception_for_null_salary;

    --%test(Does nothing if employee_id does not exist)
    PROCEDURE employee_id_does_not_exist;

END test_update_salary;
/

CREATE OR REPLACE PACKAGE BODY test_update_salary
IS

    PROCEDURE check_normal_function(
        test_emp_id employees.employee_id%TYPE,
        test_salary employees.salary%TYPE
    );

    PROCEDURE salary_in_range
    IS
        test_emp_id CONSTANT employees.employee_id%TYPE := 110;
        test_salary CONSTANT employees.salary%TYPE := 8500;
    BEGIN
        check_normal_function(
            test_emp_id => test_emp_id,
            test_salary => test_salary
        );
    END salary_in_range;

    FUNCTION get_all_as_cursor RETURN SYS_REFCURSOR;

    PROCEDURE salary_below_minimum
    IS
        test_emp_id CONSTANT employees.employee_id%TYPE := 110;
        test_salary CONSTANT employees.salary%TYPE := 4000;
        unchanged_before SYS_REFCURSOR;
        unchanged_after  SYS_REFCURSOR;
    BEGIN
        -- prepare expected and before data
        unchanged_before := get_all_as_cursor;

        -- execute
        proc_update_salary(employee_id => test_emp_id,
                    salary => test_salary);

        -- check results
        unchanged_after := get_all_as_cursor;

        ut.expect(unchanged_after).to_equal(unchanged_before);

    END salary_below_minimum;

    PROCEDURE salary_above_maximum
    IS
        test_emp_id CONSTANT employees.employee_id%TYPE := 110;
        test_salary CONSTANT employees.salary%TYPE := 10000;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
    unchanged_before SYS_REFCURSOR;
    unchanged_after  SYS_REFCURSOR;
BEGIN
    -- prepare expected and before data
    unchanged_before := get_all_as_cursor;

    -- execute
    proc_update_salary(employee_id => test_emp_id,
                salary => test_salary);

    -- check results
    unchanged_after := get_all_as_cursor;

    ut.expect(unchanged_after).to_equal(unchanged_before);

END salary_above_maximum;

PROCEDURE check_second_employee_id
IS
    test_emp_id CONSTANT employees.employee_id%TYPE := 107;
    test_salary CONSTANT employees.salary%TYPE := 5000;
BEGIN
    check_normal_function(
        test_emp_id => test_emp_id,
        test_salary => test_salary
    );
END check_second_employee_id;

PROCEDURE exception_for_null_employee_id
IS
BEGIN
    proc_update_salary(NULL, 4000);
END exception_for_null_employee_id;

PROCEDURE exception_for_null_salary
IS
    test_emp_id CONSTANT employees.employee_id%TYPE := 110;
    test_salary CONSTANT employees.salary%TYPE := NULL;
    unchanged_before SYS_REFCURSOR;
    unchanged_after  SYS_REFCURSOR;
BEGIN
    -- prepare expected and before data
    unchanged_before := get_all_as_cursor;

    -- execute
    proc_update_salary(employee_id => test_emp_id,
                salary => test_salary);

    -- there is no fail function
    ut.expect(1).to_equal(1);
EXCEPTION
    WHEN OTHERS THEN
        ut.expect(SQLCODE).to_equal(-20001);

        -- check results
        unchanged_after := get_all_as_cursor;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
        ut.expect(unchanged_after).to_equal(unchanged_before);
END exception_for_null_salary;

PROCEDURE employee_id_does_not_exist
IS
    test_emp_id CONSTANT employees.employee_id%TYPE := -1;
    test_salary CONSTANT employees.salary%TYPE := 5000;
    unchanged_before SYS_REFCURSOR;
    unchanged_after  SYS_REFCURSOR;
BEGIN
    -- prepare expected and before data
    unchanged_before := get_all_as_cursor;

    -- execute
    proc_update_salary(employee_id => test_emp_id,
                salary => test_salary);

    -- check results
    unchanged_after := get_all_as_cursor;

    ut.expect(unchanged_after).to_equal(unchanged_before);

END employee_id_does_not_exist;

-- Utilities from here
FUNCTION get_all_as_cursor
RETURN SYS_REFCURSOR
IS
    cur SYS_REFCURSOR;
BEGIN
    OPEN cur FOR
    SELECT *
    FROM   employees
    ORDER BY
           employee_id;

    RETURN cur;
END;

PROCEDURE check_normal_function(
    test_emp_id employees.employee_id%TYPE,
    test_salary employees.salary%TYPE
)
IS
    unchanged_before SYS_REFCURSOR;
    unchanged_after  SYS_REFCURSOR;
    expected         SYS_REFCURSOR;
    actual           SYS_REFCURSOR;
    actual_salary    employees.salary%TYPE;
BEGIN
    -- prepare expected and before data
    OPEN unchanged_before FOR
    SELECT *
    FROM   employees
    WHERE  employee_id != test_emp_id
    ORDER BY employee_id;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
        OPEN expected FOR
        SELECT *
        FROM   employees
        WHERE  employee_id = test_emp_id;

        -- execute
        proc_update_salary(employee_id => test_emp_id,
                    salary => test_salary);

        -- check results
        OPEN actual FOR
        SELECT *
        FROM   employees
        WHERE  employee_id = test_emp_id;

        SELECT salary
        INTO   actual_salary
        FROM   employees
        WHERE  employee_id = test_emp_id;

        OPEN unchanged_after FOR
        SELECT *
        FROM   employees
        WHERE  employee_id != test_emp_id
        ORDER BY employee_id;

        ut.expect(actual).to_equal(expected).exclude('SALARY');
        ut.expect(actual_salary).to_equal(test_salary);
        ut.expect(unchanged_after).to_equal(unchanged_before);

    END check_normal_function;

END test_update_salary;
/
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

This page intentionally left blank.

© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 16: Creating Triggers

Exercise 16.1: Working with Triggers

Trigger

```
CREATE OR REPLACE TRIGGER emp_after_insert
    AFTER INSERT
    ON employees
    FOR EACH ROW
BEGIN
    IF NOT func_check_salary(job_id => :NEW.job_id,
            salary => :NEW.salary) THEN
        RAISE_APPLICATION_ERROR(-20001, 'Salary out of range for job');
    END IF;
END;
/

-- Drop at end
DROP TRIGGER emp_after_insert;
```

Execute Tests

```
BEGIN
    ut.run('test_insert_trigger');
END;
/
```

Tests

```
CREATE OR REPLACE PACKAGE test_insert_trigger
IS
    --%suite(Tests for insert trigger on employees)

    --%test(Test failing insert)
    --%throws(-20001)
    PROCEDURE salary_out_of_range;

    --%test(Test normal insert)
    PROCEDURE salary_in_range;
END test_insert_trigger;
/

CREATE OR REPLACE PACKAGE BODY test_insert_trigger
IS
    test_emp_id CONSTANT employees.employee_id%TYPE := 999;
    test_job_id CONSTANT employees.job_id%TYPE := 'PU_CLERK';

    PROCEDURE perform_insert(
        salary employees.salary%TYPE
    );

    PROCEDURE salary_out_of_range
    IS
        test_salary CONSTANT employees.salary%TYPE := 10000;
        unchanged_before SYS_REFCURSOR;
        unchanged_after  SYS_REFCURSOR;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
    count_after      PLS_INTEGER;
    count_before     PLS_INTEGER;
    salary_after     employees.salary%TYPE;
BEGIN
    -- INSERT new row (set non NULL columns and salary out of range)
    perform_insert(salary => test_salary);
END salary_out_of_range;

FUNCTION get_unchanged(
    employee_id employees.employee_id%TYPE
)
RETURN SYS_REFCURSOR;

PROCEDURE salary_in_range
IS
    test_salary CONSTANT employees.salary%TYPE := 4000;
    unchanged_before SYS_REFCURSOR;
    unchanged_after  SYS_REFCURSOR;
    count_after      PLS_INTEGER;
    count_before     PLS_INTEGER;
    salary_after     employees.salary%TYPE;
BEGIN
    -- Get data before INSERT
    unchanged_before := get_unchanged(
        employee_id => test_emp_id);

    SELECT COUNT(*)
    INTO   count_before
    FROM   employees;

    -- INSERT new row (set non NULL columns and salary in range)
    perform_insert(salary => test_salary);

    -- Get data after INSERT
    unchanged_after := get_unchanged(
        employee_id => test_emp_id);

    SELECT COUNT(*)
    INTO   count_after
    FROM   employees;

    -- Assert
    ut.expect(unchanged_after)
        .to_equal(unchanged_before)
        .join_by('EMPLOYEE_ID');

    ut.expect(count_after).to_equal(count_before + 1);

    -- This SELECT could throw NO_DATA_FOUND, which would be a failure
    SELECT e.salary
    INTO   salary_after
    FROM   employees e
    WHERE  e.employee_id = test_emp_id;

    ut.expect(salary_after).to_equal(test_salary);

EXCEPTION
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
        -- We could let this fail normally,
        -- but this allows us to customize the message
        WHEN NO_DATA_FOUND THEN
            ut.expect(TRUE,
                'There was no employee with id ' || test_emp_id)
                    .not_to_equal(TRUE);
    END;

    -- Test utilities from here

    -- Get all employees except the one identified
    FUNCTION get_unchanged(
        employee_id employees.employee_id%TYPE
    )
    RETURN SYS_REFCURSOR
    IS
        cur SYS_REFCURSOR;
    BEGIN
        OPEN cur FOR
        SELECT e.*
        FROM   employees e
        WHERE  e.employee_id !=get_unchanged.employee_id
        ORDER BY
               e.employee_id;

        RETURN cur;
    END get_unchanged;

    -- INSERT new row (set non NULL columns and salary)
    PROCEDURE perform_insert(
        salary employees.salary%TYPE
    )
    IS
    BEGIN
        INSERT INTO
            employees(
                employee_id,
                last_name,
                email,
                hire_date,
                job_id,
                salary
        )
        VALUES (
                test_emp_id,
                'SAWYER',
                'TSAWYER',
                SYSDATE,
                test_job_id,
                perform_insert.salary
        );
    END;

END test_insert_trigger;
/
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

BONUS

Trigger

```
CREATE OR REPLACE TRIGGER emp_after_insert_update
    AFTER INSERT
    OR UPDATE OF job_id, salary
    ON employees
    FOR EACH ROW
BEGIN
    IF NOT func_check_salary(job_id => :NEW.job_id,
            salary => :NEW.salary) THEN
        IF INSERTING THEN
            RAISE_APPLICATION_ERROR(-20001, 'Salary out of range for job');
        ELSIF UPDATING THEN
            RAISE_APPLICATION_ERROR(-20002, 'Salary out of range for job');
        END IF;
    END IF;
END;
/

-- Drop at end
DROP TRIGGER emp_after_insert_update;
```

Execute tests

```
BEGIN
    ut.run('test_insert_update_trigger');
END;
/
```

Tests

```
CREATE OR REPLACE PACKAGE test_insert_update_trigger
IS
    --%suite(Tests for insert trigger on employees)

    --%test(Test failing insert)
    --%throws(-20001)
    PROCEDURE insert_salary_out_of_range;

    --%test(Test normal insert)
    PROCEDURE insert_salary_in_range;

    --%test(Test normal update of salary)
    PROCEDURE update_salary_in_range;

    --%test(Test normal update of job_id)
    PROCEDURE update_job_id_in_range;

    --%test(Test normal update of salary and job_id)
    PROCEDURE update_both_in_range;

    --%test(Test failing update of salary)
    --%throws(-20002)
    PROCEDURE update_salary_out_of_range;

    --%test(Test failing update of job_id)
    --%throws(-20002)
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
    PROCEDURE update_job_id_out_of_range;

    --%test(Test failing update of salary and job_id)
    --%throws(-20002)
    PROCEDURE update_both_out_of_range;

END test_insert_update_trigger;
/

CREATE OR REPLACE PACKAGE BODY test_insert_update_trigger
IS
    test_new_emp_id CONSTANT employees.employee_id%TYPE := 999;
    test_new_job_id CONSTANT employees.job_id%TYPE := 'PU_CLERK';
    test_upd_emp_id CONSTANT employees.employee_id%TYPE := 107;

    PROCEDURE perform_insert(
        salary employees.salary%TYPE
    );

    PROCEDURE insert_salary_out_of_range
    IS
        test_salary CONSTANT employees.salary%TYPE := 10000;
    BEGIN
        -- INSERT new row (set non NULL columns and salary out of range)
        perform_insert(salary => test_salary);
    END insert_salary_out_of_range;

    FUNCTION get_unchanged(
        employee_id employees.employee_id%TYPE
    )
    RETURN SYS_REFCURSOR;

    PROCEDURE insert_salary_in_range
    IS
        test_salary CONSTANT employees.salary%TYPE := 4000;
        unchanged_before SYS_REFCURSOR;
        unchanged_after  SYS_REFCURSOR;
        count_after      PLS_INTEGER;
        count_before     PLS_INTEGER;
        salary_after     employees.salary%TYPE;
    BEGIN
        -- Get data before INSERT
        unchanged_before := get_unchanged(
            employee_id => test_new_emp_id);

        SELECT COUNT(*)
        INTO   count_before
        FROM   employees;

        -- INSERT new row (set non NULL columns and salary in range)
        perform_insert(salary => test_salary);

        -- Get data after INSERT
        unchanged_after := get_unchanged(
            employee_id => test_new_emp_id);

        SELECT COUNT(*)
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
    INTO   count_after
    FROM   employees;

    -- Assert
    ut.expect(unchanged_after)
        .to_equal(unchanged_before)
        .join_by('EMPLOYEE_ID');

    ut.expect(count_after).to_equal(count_before + 1);

    -- This SELECT could throw NO_DATA_FOUND, which would be a failure
    SELECT e.salary
    INTO   salary_after
    FROM   employees e
    WHERE  e.employee_id = test_new_emp_id;

    ut.expect(salary_after).to_equal(test_salary);

EXCEPTION
    -- We could let this fail normally,
    -- but this allows us to customize the message
    WHEN NO_DATA_FOUND THEN
        ut.expect(TRUE,
            'There was no employee with id ' || test_new_emp_id)
                .not_to_equal(TRUE);

END insert_salary_in_range;

PROCEDURE test_working_update(
    unchanged_before SYS_REFCURSOR,
    count_before     PLS_INTEGER,
    expected_salary  employees.salary%TYPE,
    expected_job_id  employees.job_id%TYPE
);

PROCEDURE update_salary_in_range
IS
    test_salary CONSTANT employees.salary%TYPE := 5000;
    unchanged_before SYS_REFCURSOR;
    count_before     PLS_INTEGER;
    salary_before    employees.salary%TYPE;
    job_id_before    employees.job_id%TYPE;
BEGIN
    -- Get data before UPDATE
    unchanged_before := get_unchanged(
        employee_id => test_upd_emp_id);

    SELECT COUNT(*)
    INTO   count_before
    FROM   employees;

    -- This SELECT could throw NO_DATA_FOUND, which would be a failure
    SELECT e.salary, e.job_id
    INTO   salary_before, job_id_before
    FROM   employees e
    WHERE  e.employee_id = test_upd_emp_id;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
    -- UPDATE and set salary to valid value
    UPDATE employees
    SET    salary      = test_salary
    WHERE  employee_id = test_upd_emp_id;

    -- All the checks are the same for each test
    test_working_update(
        unchanged_before => unchanged_before,
        count_before     => count_before,
        expected_salary  => test_salary,
        expected_job_id  => job_id_before -- unchanged
    );
EXCEPTION
    -- We could let this fail normally,
    -- but this allows us to customize the message
    WHEN NO_DATA_FOUND THEN
        ut.expect(TRUE,
            'There was no employee with id ' || test_upd_emp_id)
                .not_to_equal(TRUE);
END update_salary_in_range;

PROCEDURE update_job_id_in_range
IS
    test_job_id CONSTANT employees.job_id%TYPE := 'MK_REP';
    unchanged_before SYS_REFCURSOR;
    count_before     PLS_INTEGER;
    salary_before    employees.salary%TYPE;
    job_id_before    employees.job_id%TYPE;
BEGIN
    -- Get data before UPDATE
    unchanged_before := get_unchanged(
        employee_id => test_upd_emp_id);

    SELECT COUNT(*)
    INTO   count_before
    FROM   employees;

    -- This SELECT could throw NO_DATA_FOUND, which would be a failure
    SELECT e.salary, e.job_id
    INTO   salary_before, job_id_before
    FROM   employees e
    WHERE  e.employee_id = test_upd_emp_id;

    -- UPDATE and set salary to valid value
    UPDATE employees
    SET    job_id      = test_job_id
    WHERE  employee_id = test_upd_emp_id;

    -- All the checks are the same for each test
    test_working_update(
        unchanged_before => unchanged_before,
        count_before     => count_before,
        expected_salary  => salary_before,  -- unchanged
        expected_job_id  => test_job_id
    );
EXCEPTION
    -- We could let this fail normally,
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
    -- but this allows us to customize the message
    WHEN NO_DATA_FOUND THEN
        ut.expect(TRUE,
            'There was no employee with id ' || test_upd_emp_id)
                .not_to_equal(TRUE);
END update_job_id_in_range;

PROCEDURE update_both_in_range
IS
    test_salary CONSTANT employees.salary%TYPE := 10000;
    test_job_id CONSTANT employees.job_id%TYPE := 'MK_MAN';
    unchanged_before SYS_REFCURSOR;
    count_before     PLS_INTEGER;
    salary_before    employees.salary%TYPE;
    job_id_before    employees.job_id%TYPE;
BEGIN
    -- Get data before UPDATE
    unchanged_before := get_unchanged(
        employee_id => test_upd_emp_id);

    SELECT COUNT(*)
    INTO   count_before
    FROM   employees;

    -- This SELECT could throw NO_DATA_FOUND, which would be a failure
    SELECT e.salary, e.job_id
    INTO   salary_before, job_id_before
    FROM   employees e
    WHERE  e.employee_id = test_upd_emp_id;

    -- UPDATE and set salary to valid value
    UPDATE employees
    SET    job_id      = test_job_id,
           salary      = test_salary
    WHERE  employee_id = test_upd_emp_id;

    -- All the checks are the same for each test
    test_working_update(
        unchanged_before => unchanged_before,
        count_before     => count_before,
        expected_salary  => test_salary,
        expected_job_id  => test_job_id
    );
EXCEPTION
    -- We could let this fail normally,
    -- but this allows us to customize the message
    WHEN NO_DATA_FOUND THEN
        ut.expect(TRUE,
            'There was no employee with id ' || test_upd_emp_id)
                .not_to_equal(TRUE);
END update_both_in_range;

PROCEDURE update_salary_out_of_range
IS
    test_salary CONSTANT employees.salary%TYPE := 12000;
BEGIN
    UPDATE employees
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
    SET    salary      = test_salary
    WHERE  employee_id = test_upd_emp_id;
END;

PROCEDURE update_job_id_out_of_range
IS
    test_job_id CONSTANT employees.job_id%TYPE := 'ST_MAN';
BEGIN
    UPDATE employees
    SET    job_id      = test_job_id
    WHERE  employee_id = test_upd_emp_id;
END;

PROCEDURE update_both_out_of_range
IS
    test_salary CONSTANT employees.salary%TYPE := 12000;
    test_job_id CONSTANT employees.job_id%TYPE := 'ST_MAN';
BEGIN
    UPDATE employees
    SET    job_id      = test_job_id,
           salary      = test_salary
    WHERE  employee_id = test_upd_emp_id;
END;

-- Test utilities from here

-- Get all employees except the one identified
FUNCTION get_unchanged(
    employee_id employees.employee_id%TYPE
)
RETURN SYS_REFCURSOR
IS
    cur SYS_REFCURSOR;
BEGIN
    OPEN cur FOR
    SELECT e.*
    FROM   employees e
    WHERE  e.employee_id != get_unchanged.employee_id
    ORDER BY
           e.employee_id;

    RETURN cur;
END get_unchanged;

-- INSERT new row (set non NULL columns and salary)
PROCEDURE perform_insert(
    salary employees.salary%TYPE
)
IS
BEGIN
    INSERT INTO
        employees(
            employee_id,
            last_name,
            email,
            hire_date,
            job_id,
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
                salary
        )
        VALUES (
                test_new_emp_id,
                'SAWYER',
                'TSAWYER',
                SYSDATE,
                test_new_job_id,
                perform_insert.salary
        );
    END;

    -- Most checks for working updates are the same
    PROCEDURE test_working_update(
        unchanged_before SYS_REFCURSOR,
        count_before     PLS_INTEGER,
        expected_salary  employees.salary%TYPE,
        expected_job_id  employees.job_id%TYPE
    )
    IS
        unchanged_after  SYS_REFCURSOR;
        count_after      PLS_INTEGER;
        salary_after     employees.salary%TYPE;
        job_id_after     employees.job_id%TYPE;
    BEGIN
        -- UPDATE has already occurred at this point

        -- Get data after UPDATE
        unchanged_after := get_unchanged(
            employee_id => test_upd_emp_id);

        SELECT COUNT(*)
        INTO   count_after
        FROM   employees;

        -- Assert
        ut.expect(unchanged_after)
            .to_equal(unchanged_before)
            .join_by('EMPLOYEE_ID');

        ut.expect(count_after).to_equal(count_before);

        -- This SELECT could throw NO_DATA_FOUND, which would be a failure
        -- Exception is caught by calling procedure
        SELECT e.salary, e.job_id
        INTO   salary_after, job_id_after
        FROM   employees e
        WHERE  e.employee_id = test_upd_emp_id;

        ut.expect(salary_after).to_equal(expected_salary);
        ut.expect(job_id_after).to_equal(expected_job_id);
    END test_working_update;

END test_insert_update_trigger;
/
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

Chapter 17: Data Definition Language

Exercise 17.1: Table Management

```
-- Drop the benefits table to start fresh
DROP TABLE benefits;

-- 1
CREATE TABLE benefits (
    benefit_id             NUMBER(3)    NOT NULL,
    benefit_name           VARCHAR2(25),
    benefit_type           VARCHAR2(20) DEFAULT 'HEALTH CARE',
    benefit_effective_date DATE,
    benefit_max_allowance  NUMBER(8,2)
);

-- Make benefit_id the primary key
-- (could also have done this in the create table)
ALTER TABLE benefits
    ADD CONSTRAINT benefits_pk PRIMARY KEY(benefit_id);

-- 2
DESCRIBE benefits;

-- 3
-- drop the sequence so we can start fresh
DROP SEQUENCE seq_benefits;
CREATE SEQUENCE seq_benefits
    INCREMENT BY 1 START WITH 1;

-- 4
INSERT INTO
       benefits
VALUES (
       seq_benefits.NEXTVAL,
       '401K',
       'Retirement' ,
       DATE '2010-01-01',
       250000.00
);
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 5
INSERT INTO
       benefits (
           benefit_id,
           benefit_name,
           benefit_type,
           benefit_effective_date,
           benefit_max_allowance
       )
VALUES (
       seq_benefits.NEXTVAL,
       'Medical PPO',
       'Health' ,
       DATE '2011-01-01',
       100000.00
);

-- 6
INSERT INTO
       benefits(
           benefit_id,
           benefit_name,
           benefit_type,
           benefit_effective_date,
           benefit_max_allowance
       )
VALUES (
       seq_benefits.NEXTVAL,
       'Medical Ins',
       DEFAULT,
       DATE '2012-01-01',
       125000.00
);

-- 7
SELECT *
FROM   benefits;

-- 8
INSERT INTO
       benefits (
           benefit_id,
           benefit_name,
           benefit_effective_date,
           benefit_max_allowance
       )
VALUES (
       seq_benefits.NEXTVAL,
       'No default name provided' ,
       DATE '2013-01-01',
       150000.00
);
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 9
SELECT *
FROM   benefits;

-- 10
UPDATE benefits
SET    benefit_type = DEFAULT
WHERE  benefit_type LIKE 'H%';

-- 11
SELECT *
FROM   benefits;

-- 12
COMMIT;

-- 13
CREATE OR REPLACE VIEW vw_h_b
AS
    SELECT benefit_id ,
           benefit_name,
           benefit_type,
           benefit_max_allowance
    FROM   benefits
    WHERE  benefit_type LIKE 'HEALTH%';

-- 14
DESCRIBE vw_h_b

-- 15
SELECT *
FROM   vw_h_b;

-- 16
-- this will fail since the table is not empty
ALTER TABLE benefits
  ADD (max_dependents NUMBER(2) NOT NULL);

-- 17
ALTER TABLE benefits
  ADD (max_dependents NUMBER(2) DEFAULT 0 NOT NULL);

-- 18
SELECT *
FROM   benefits;

-- 19
SELECT *
FROM   vw_h_b;


-- BONUS
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 20
ALTER TABLE benefits
  MODIFY (benefit_name VARCHAR2(50));

DESCRIBE benefits

-- 21
-- This will fail because the size is too small
ALTER TABLE benefits
  MODIFY (benefit_name VARCHAR2(20));

-- 22
INSERT INTO
       benefits(
           benefit_id,
           benefit_name,
           benefit_type,
           benefit_effective_date,
           benefit_max_allowance,
           max_dependents
       )
SELECT seq_benefits.NEXTVAL,
       benefit_name,
       benefit_type,
       benefit_effective_date,
       benefit_max_allowance,
       max_dependents
FROM   benefits;

-- 23
SELECT *
FROM   benefits;

-- 24
ROLLBACK;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.

```
-- 25
INSERT INTO
       benefits(
           benefit_id,
           benefit_name,
           benefit_type,
           benefit_effective_date,
           benefit_max_allowance,
           max_dependents
       )
SELECT seq_benefits.NEXTVAL,
       benefit_name,
       benefit_type,
       benefit_effective_date,
       benefit_max_allowance,
       max_dependents
FROM   benefits;

-- 26
SELECT *
FROM   benefits;
```


© 2023 Copyright ROI Training, Inc.: All rights reserved. Not to be reproduced without prior written consent.