Base de Datos Oracle 10*g*: Desarrollo de Unidades de Programa PL/SQL

Volumen 3: Prácticas Adicionales

D17169CS11
Edición 1.1
Agosto de 2004
D21978

ORACLE

Autores

Glenn Stokol
Aniket Raut

Colaboradores y Revisores Técnicos

Andrew Brannigan
Dr. Christoph Burandt
Kathryn Cunningham
Marjolein Dekkers
Janis Fleishman
Nancy Greenberg
Stefan Grenstad
Elizabeth Hall
Rosita Hanoman
Craig Hollister
Taj-ul Islam
Eric Lee
Bryn Llewellyn
Werner Nowatzky
Nagavalli Pataballa
Sunitha Patel
Denis Raphaely
Helen Robertson
Grant Spencer
Tone Thomas
Priya Vennapusa
Ken Woolfe
Cesljas Zarco

Editor

Joseph Fernandez

Contenido

Prácticas Adicionales

Prácticas Adicionales: Visión General

Estas prácticas adicionales se proporcionan como complemento del curso *Base de Datos Oracle 10g: Desarrollo de Unidades de Programa PL/SQL*. En estas prácticas se aplican los conceptos que ha aprendido a lo largo del curso.

Las prácticas adicionales constan de dos partes:

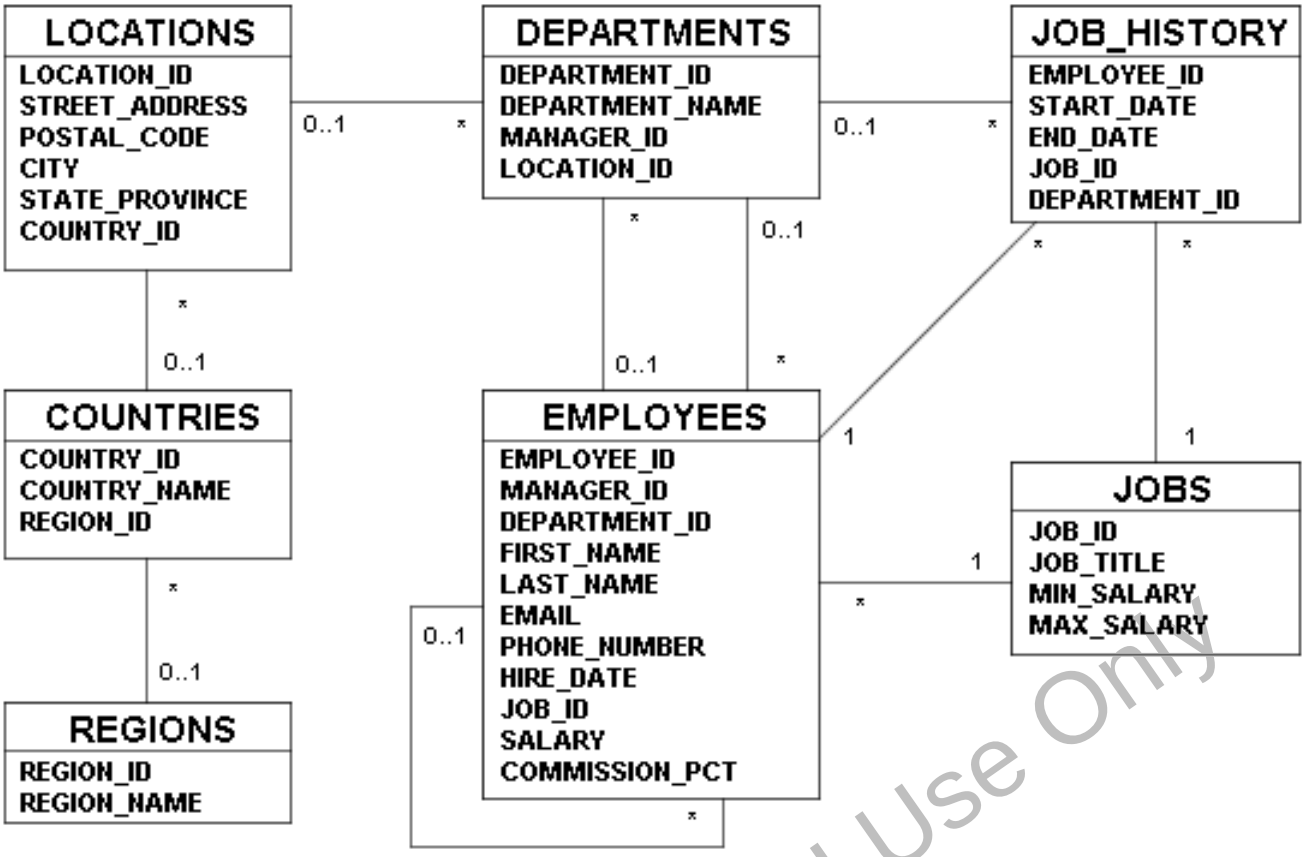
La parte A proporciona ejercicios complementarios para crear procedimientos almacenados, funciones, paquetes y disparadores así como para utilizar los paquetes proporcionados por Oracle con *i*SQL*Plus como entorno de desarrollo. Las tablas utilizadas en esta parte de las prácticas adicionales son `EMPLOYEES`, `JOBS`, `JOB_HISTORY` y `DEPARTMENTS`.

La parte B es un caso práctico que se puede realizar al final del curso. Esta parte complementa las prácticas en el proceso de creación y gestión de unidades de programa. Las tablas utilizadas en este caso práctico están basadas en una base de datos de vídeos y contienen las tablas `TITLE`, `TITLE_COPY`, `RENTAL`, `RESERVATION` y `MEMBER`.

Al principio de las partes A y B se proporciona un diagrama de entidad/relación. Cada diagrama de entidad/relación muestra las entidades de tabla y sus relaciones. Para obtener definiciones más detalladas de las tablas y los datos que contienen las tablas, consulte el apéndice titulado “Prácticas Adicionales: Descripciones de las Tablas y Datos”.

Parte A: Diagrama de Entidad/Relación

Recursos Humanos



Parte A (continuación)

Nota: Estos ejercicios se pueden utilizar como práctica adicional al describir cómo crear procedimientos.

1. En este ejercicio, cree un programa para agregar un nuevo trabajo a la tabla JOBS.
   a. Cree un procedimiento almacenado denominado NEW_JOB para introducir un nuevo orden en la tabla JOBS. El procedimiento debe aceptar tres parámetros. El primero y el segundo proporcionan un identificador y un título de trabajo. El tercero proporciona el salario mínimo. Utilice el salario máximo del nuevo trabajo como el doble del salario mínimo proporcionado para el identificador de trabajo.
   b. Llame al procedimiento para agregar un nuevo trabajo con el identificador 'SY_ANAL', el título 'System Analyst'y un salario mínimo de 6,000.
   c. Verifique si se ha agregado una fila y anote el identificador del nuevo trabajo para utilizarlo en el siguiente ejercicio. Confirme los cambios.
2. En este ejercicio, cree un programa para agregar una nueva fila a la tabla JOB_HISTORY para un empleado existente.
   a. Cree un procedimiento almacenado denominado ADD_JOB_HIST para agregar una nueva fila a la tabla JOB_HISTORY para un empleado que está cambiando su trabajo al nuevo identificador de trabajo ('SY_ANAL') creado en el ejercicio 1b.
      El procedimiento debe proporcionar dos parámetros: uno para el identificador del empleado que está cambiando el trabajo y el otro para el identificador del nuevo trabajo. Consulte el identificador del empleado en la tabla EMPLOYEES e introdúzcalo en la tabla JOB_HISTORY. Indique la fecha de contratación de este empleado como fecha de inicio y la fecha de hoy como fecha final para esta fila en la tabla JOB_HISTORY.
      Cambie la fecha de contratación de este empleado en la tabla EMPLOYEES a la fecha de hoy. Actualice el identificador de trabajo de este empleado según el identificador de trabajo transferido como parámetro (utilice el identificador de trabajo 'SY_ANAL') y el salario igual al salario mínimo para ese identificador de trabajo + 500.
      Nota: Incluya el manejo de excepciones para manejar el intento de insertar un empleado no existente.
   b. Desactive todos los disparadores de las tablas EMPLOYEES, JOBS y JOB_HISTORY antes de llamar al procedimiento ADD_JOB_HIST.
   c. Ejecute el procedimiento con el identificador de empleado 106 y el identificador de trabajo 'SY_ANAL' como parámetros.
   d. Consulte las tablas JOB_HISTORY y EMPLOYEES para ver los cambios del empleado 106 y, a continuación, confirme los cambios.
   e. Vuelva a activar los disparadores de las tablas EMPLOYEES, JOBS y JOB_HISTORY.

Parte A (continuación)

3. En este ejercicio, cree un programa para actualizar los salarios mínimo y máximo de un trabajo de la tabla JOBS.
   a. Cree un procedimiento almacenado denominado UPD_JOBSAL para actualizar los salarios mínimo y máximo de un identificador de trabajo concreto de la tabla JOBS. El procedimiento debe proporcionar tres parámetros: el identificador de trabajo, un nuevo salario mínimo y un nuevo salario máximo. Agregue el manejo de excepciones para justificar un identificador de trabajo no válido en la tabla JOBS. Emita una excepción si el salario máximo proporcionado es menor que el salario mínimo. Proporcione un mensaje que se mostrará si la fila de la tabla JOBS está bloqueada.
   Indicación: El número de error de recurso bloqueado/ocupado es –54.
   b. Ejecute el procedimiento UPD_JOBSAL utilizando un identificador de trabajo 'SY_ANAL', un salario mínimo de 7000 y un salario máximo de 140.
   Nota: Esto debe generar un mensaje de excepción.
   c. Desactive los disparadores de las tablas EMPLOYEES y JOBS.
   d. Ejecute el procedimiento UPD_JOBSAL utilizando un identificador de trabajo 'SY_ANAL', un salario mínimo de 7000 y un salario máximo de 14000.
   e. Consulte la tabla JOBS para ver los cambios y, a continuación, confírmelos.
   f. Active los disparadores de las tablas EMPLOYEES y JOBS.
4. En este ejercicio, cree un procedimiento para controlar si los salarios de los empleados han excedido la media para el tipo de trabajo.
   a. Desactive el disparador SECURE_EMPLOYEES.
   b. En la tabla EMPLOYEES, agregue una columna EXCEED_AVGSAL para almacenar hasta tres caracteres y un valor por defecto 'NO'. Utilice una restricción de control para permitir los valores 'YES' o 'NO'.
   c. Escriba un procedimiento almacenado denominado CHECK_AVGSAL que comprueba si el salario de cada empleado excede el salario medio para JOB_ID. El salario medio de un trabajo se calcula a partir de la información de la tabla JOBS. Si el salario del empleado excede la media para el trabajo, actualice la columna EXCEED_AVGSAL de la tabla EMPLOYEES con el valor YES; de lo contrario, defina el valor en NO. Utilice un cursor para seleccionar las filas de los empleados con la opción FOR UPDATE de la consulta. Agregue el manejo de excepciones para justificar el bloqueo de un registro.
   Indicación: El número de error de recurso bloqueado/ocupado es –54. Escriba y utilice una función local denominada GET_JOB_AVGSAL para determinar el salario medio de un identificador de trabajo especificado como parámetro.
   d. Ejecute el procedimiento CHECK_AVGSAL. A continuación, para ver los resultados de las modificaciones, escriba una consulta para mostrar el identificador del empleado, el trabajo, el salario medio del trabajo, el salario del empleado y la columna del indicador exceed_avgsal para empleados cuyos salarios exceden la media para el trabajo. Por último, confirme los cambios.

Nota: Estos ejercicios se pueden utilizar como práctica adicional al describir cómo crear funciones.

Parte A (continuación)

5. Cree un subprograma para recuperar el número de años de servicio de un empleado concreto.
   a. Cree una función almacenada denominada `GET_YEARS_SERVICE` para recuperar el número total de años de servicio de un empleado concreto. La función debe aceptar el identificador de empleado como parámetro y devolver el número de años de servicio. Agregue la gestión de errores para justificar un identificador de empleado no válido.
   b. Llame a la función `GET_YEARS_SERVICE` en una llamada a `DBMS_OUTPUT.PUT_LINE` para un empleado que tenga el identificador `999`.
   c. Muestre el número de años de servicio para el empleado 106 mediante la llamada de `DBMS_OUTPUT.PUT_LINE` a la función `GET_YEARS_SERVICE`.
   d. Consulte los datos del empleado concreto en las tablas `JOB_HISTORY` y `EMPLOYEES` para verificar que las modificaciones son precisas. Los valores representados en los resultados de esta página pueden variar de los valores obtenidos al ejecutar las consultas.
6. En este ejercicio, cree un programa para recuperar el número de trabajos diferentes que ha tenido un empleado durante su servicio.
   a. Cree una función almacenada denominada `GET_JOB_COUNT` para recuperar el número total de trabajos diferentes que ha tenido un empleado.
   La función debe aceptar el identificador de empleado en un parámetro y devolver el número de trabajos diferentes que ha tenido un empleado hasta la fecha, incluido el presente. Agregue el manejo de excepciones para justificar un identificador de empleado no válido.
   Indicación: Utilice los distintos identificadores de trabajo de la tabla `JOB_HISTORY` y excluya el identificador de trabajo actual, si se trata de uno de los identificadores de trabajo en los que ya ha trabajado el empleado. Una dos consultas con `UNION` y cuente las filas recuperadas en una tabla PL/SQL. Utilice `FETCH` con `BULK COLLECT INTO` para obtener los trabajos únicos del empleado.
   b. Llame a la función para el empleado con el identificador `176`.

Nota: Estos ejercicios se pueden utilizar como práctica adicional al describir cómo crear paquetes.

7. Cree un paquete denominado `EMPJOB_PKG` que contenga los procedimientos `NEW_JOB`, `ADD_JOB_HIST` y `UPD_JOBSAL`, así como las funciones `GET_YEARS_SERVICE` y `GET_JOB_COUNT`.
   a. Cree la especificación del paquete con toda la construcción del subprograma pública. Mueva cualquier tipo definido localmente del subprograma a la especificación del paquete.
   b. Cree el cuerpo del paquete con la implementación del subprograma; recuerde eliminar de las implementaciones del subprograma cualquier tipo que haya movido a la especificación del paquete.

Parte A (continuación)

c. Llame al procedimiento `EMPJOB_PKG.NEW_JOB` para crear un nuevo trabajo con el identificador `PR_MAN`, el título `Public Relations Manager` y el salario `6,250`.

d. Llame al procedimiento `EMPJOB_PKG.ADD_JOB_HIST` para modificar el trabajo del empleado con identificador `110` por el identificador de trabajo `PR_MAN`.
Nota: Debe desactivar el disparador `UPDATE_JOB_HISTORY` antes de ejecutar el procedimiento `ADD_JOB_HIST` y volver a activar el disparador después de ejecutar el procedimiento.

e. Consulte las tablas `JOBS`, `JOB_HISTORY` y `EMPLOYEES` para verificar los resultados.

Nota: Estos ejercicios se pueden utilizar como práctica adicional al describir cómo crear disparadores de base de datos.

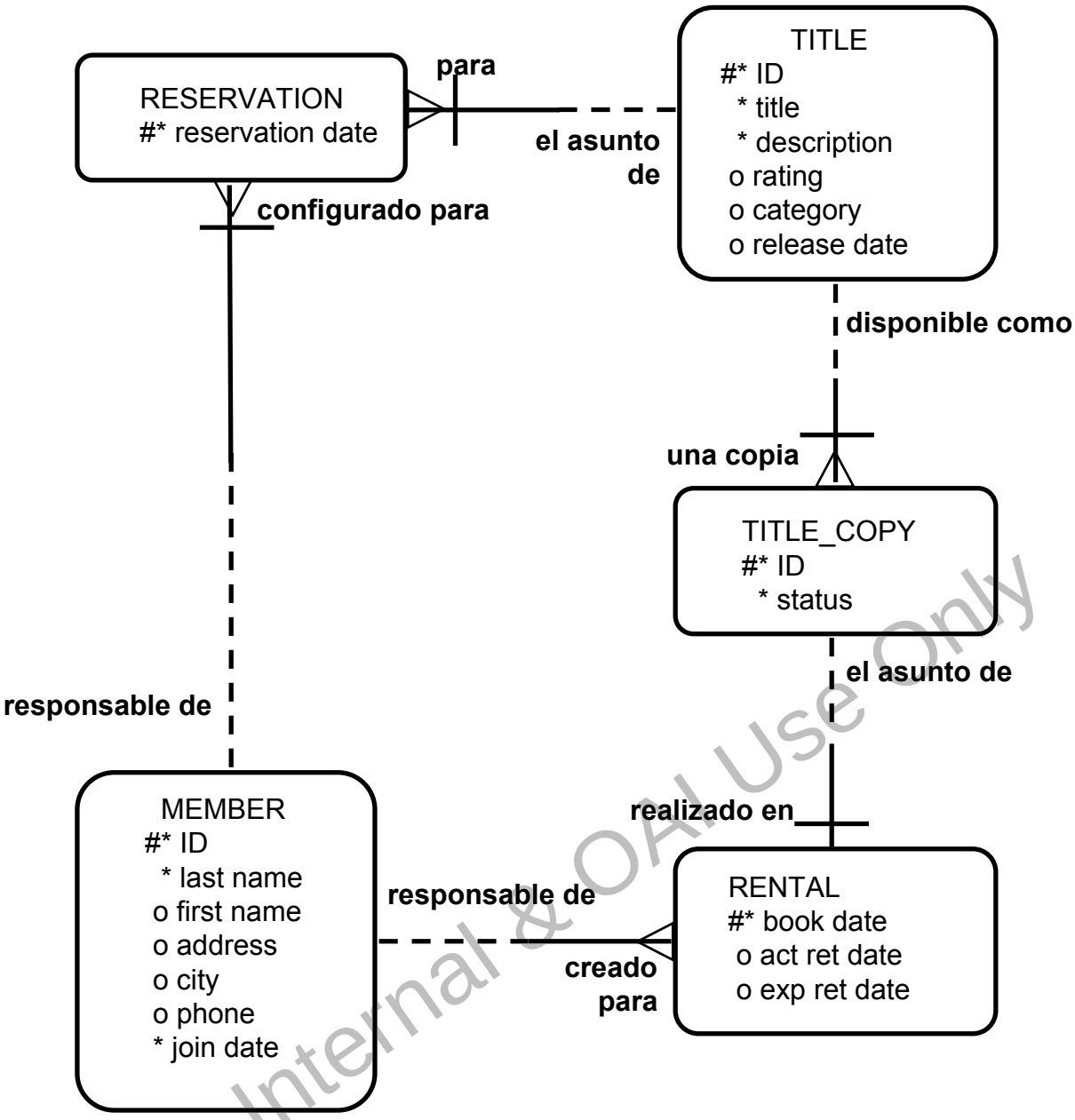
8. En este ejercicio, cree un disparador para asegurarse de que los salarios mínimo y máximo de un trabajo nunca se modifiquen de forma que el salario de un empleado existente con ese identificador de trabajo esté fuera del nuevo rango especificado para el trabajo.

   a. Cree un disparador denominado `CHECK_SAL_RANGE` que se dispare antes de cada fila que se
   actualice en las columnas `MIN_SALARY` y `MAX_SALARY` de la tabla `JOBS`. Para cada valor de salario mínimo o máximo que se cambie, compruebe si el salario de cualquier empleado existente con ese identificador de trabajo de la tabla `EMPLOYEES` entra dentro del nuevo rango de salarios especificado para este identificador de trabajo. Incluya el manejo de excepciones para cubrir un rango de salarios que afecte al registro de cualquier empleado existente.

   b. Pruebe el disparador utilizando el trabajo `SY_ANAL` y defina el nuevo salario mínimo en `5000` y el nuevo salario máximo en `7000`. Antes de realizar los cambios, escriba una consulta para mostrar el rango de salarios actual para el identificador de trabajo `SY_ANAL` y otra consulta para mostrar el identificador, apellido y salario del empleado para el mismo identificador de trabajo. Después de la actualización, consulte los cambios (si existen) realizados en la tabla `JOBS` para el identificador de trabajo especificado.

   c. Mediante el trabajo `SY_ANAL`, defina el nuevo salario mínimo en `7000` y el nuevo salario máximo en `18000`. Explique los resultados.

Parte B: Diagrama de Entidad/Relación



Parte B (continuación)

En este caso práctico, creará un paquete denominado `VIDEO_PKG` que contenga los procedimientos y funciones de una aplicación de un videoclub. Esta aplicación permite a los clientes convertirse en miembros del videoclub. Cualquier miembro podrá alquilar películas, devolver películas alquiladas y reservar películas. Además, creará un disparador para asegurarse de que los datos de las tablas de vídeos se modifican sólo durante las horas laborables.

Cree el paquete con *i*SQL*Plus y utilice el paquete `DBMS_OUTPUT` proporcionado por Oracle para mostrar los mensajes.

La base de datos del videoclub contiene las siguientes tablas: `TITLE`, `TITLE_COPY`, `RENTAL`, `RESERVATION` y `MEMBER`. El diagrama de entidad/relación aparece en la página anterior.

Parte B (continuación)

1. Cargue y ejecute el archivo de comandos `E:\labs\PLPU\labs\buildvid1.sql` para crear todas las tablas y secuencias necesarias para este ejercicio.
2. Cargue y ejecute el archivo de comandos `E:\labs\PLPU\labs\buildvid2.sql` para rellenar todas las tablas creadas por el archivo de comandos `buildvid1.sql`.
3. Cree un paquete denominado `VIDEO_PKG` con los siguientes procedimientos y funciones:
   a. **`NEW_MEMBER:`** Procedimiento público que agrega un nuevo miembro a la tabla `MEMBER`. Para el número de identificador de miembro, utilice la secuencia `MEMBER_ID_SEQ`; para la fecha de unión, utilice `SYSDATE`. Transfiera los demás valores que desee insertar en una nueva fila como parámetros.
   b. **`NEW_RENTAL:`** Función pública sobrecargada para registrar un nuevo alquiler. Transfiera el número de identificador de título del vídeo que el cliente desea alquilar junto con el apellido del cliente o el número de identificador de miembro a la función. La función deberá devolver la fecha de vencimiento del vídeo. Las fechas de vencimiento son tres días desde la fecha de alquiler del vídeo. Si el estado de una película solicitada aparece como `AVAILABLE` en la tabla `TITLE_COPY` para una copia de este título, actualice la tabla `TITLE_COPY` y defina el estado en `RENTED`. Si no hay ninguna copia disponible, la función debe devolver el valor `NULL`. A continuación, inserte un nuevo registro en la tabla `RENTAL` que identifique la fecha de reserva como la fecha de hoy, el número de identificador de copia, el número de identificador de miembro, el número de identificador de título y la fecha de devolución esperada. Tenga en cuenta que puede que existan varios clientes con el mismo apellido. En este caso, haga que la función devuelva `NULL` y muestre una lista de todos los nombres de clientes que coincidan y sus números de identificador.
   c. **`RETURN_MOVIE:`** Procedimiento público que actualiza el estado de un vídeo (disponible, alquilado o dañado) y define la fecha de devolución. Transfiera el identificador de título, el identificador de copia y el estado a este procedimiento. Compruebe si existen reservas para ese título y muestre un mensaje, si está reservado. Actualice la tabla `RENTAL` y defina la fecha de devolución real en la fecha de hoy. Actualice el estado en la tabla `TITLE_COPY` según el parámetro de estado transferido al procedimiento.
   d. **`RESERVE_MOVIE:`** Procedimiento privado que se ejecuta sólo si todas las copias de vídeos solicitadas en el procedimiento `NEW_RENTAL` tienen el estado `RENTED`. Transfiera el número de identificador de miembro y el número de identificador de título a este procedimiento. Inserte un nuevo registro en la tabla `RESERVATION` y registre la fecha de reserva, el número de identificador de miembro y el número de identificador de título. Imprima un mensaje que indique que una película está reservada y la fecha de devolución esperada.

Parte B (continuación)

e. **`EXCEPTION_HANDLER:`** Procedimiento privado que se llama desde el manejador de excepciones de los programas públicos. Transfiera el número `SQLCODE` a este procedimiento y el nombre del programa (como cadena de texto) en el que se ha producido el error. Utilice `RAISE_APPLICATION_ERROR` para emitir un error personalizado. Empiece por una violación de clave única (`-1`) y una violación de clave ajena (`-2292`). Permita que el manejador de excepciones emita un error genérico para cualquier otro error.

4. Utilice los siguientes archivos de comandos ubicados en el directorio `E:\labs\PLPU\soln` para probar las rutinas:
   a. Agregue dos miembros con `sol_apb_04_a_new_members.sql`.
   b. Agregue nuevos alquileres de vídeos con `sol_apb_04_b_new_rentals.sql`.
   c. Devuelva películas con el archivo de comandos `sol_apb_04_c_return_movie.sql`.
5. Las horas laborables para el videoclub son de 8:00 a.m. a 10:00 p.m. (de domingo a viernes) y de 8:00 a.m. a 12:00 a.m. (sábados). Para asegurarse de que las tablas sólo se puedan modificar durantes estas horas, cree un procedimiento almacenado al que se llame con los disparadores de las tablas.
   a. Cree un procedimiento almacenado denominado `TIME_CHECK` que compruebe la hora actual con respecto a las horas laborables. Si la hora actual no está dentro de las horas laborables, utilice el procedimiento `RAISE_APPLICATION_ERROR` para emitir el mensaje adecuado.
   b. Cree un disparador en cada una de las cinco tablas. Arranque el disparador antes de insertar, actualizar y suprimir los datos de las tablas. Llame al procedimiento `TIME_CHECK` desde cada disparador.

Prácticas Adicionales: Soluciones

Parte A: Soluciones a la Práctica Adicional 1

1. En este ejercicio, cree un programa para agregar un nuevo trabajo a la tabla JOBS.

   a. Cree un procedimiento almacenado denominado NEW_JOB para introducir un nuevo orden en la tabla JOBS. El procedimiento debe aceptar tres parámetros. El primero y el segundo proporcionan un identificador y un título de trabajo. El tercero proporciona el salario mínimo. Utilice el salario máximo del nuevo trabajo como el doble del salario mínimo proporcionado para el identificador de trabajo.

```
CREATE OR REPLACE PROCEDURE new_job(
  jobid  IN jobs.job_id%TYPE,
  title  IN jobs.job_title%TYPE,
  minsal IN jobs.min_salary%TYPE) IS
  maxsal  jobs.max_salary%TYPE := 2 * minsal;
BEGIN
  INSERT INTO jobs(job_id, job_title, min_salary, max_salary)
  VALUES (jobid, title, minsal, maxsal);
  DBMS_OUTPUT.PUT_LINE ('New row added to JOBS table:');
  DBMS_OUTPUT.PUT_LINE (jobid || '  ' || title ||' '||
                        minsal || '  ' || maxsal);
END new_job;
/
SHOW ERRORS

Procedure created.

No errors.
```

   b. Llame al procedimiento para agregar un nuevo trabajo con el identificador 'SY_ANAL', el título 'System Analyst' y un salario mínimo de 6,000.

```
SET SERVEROUTPUT ON
EXECUTE new_job ('SY_ANAL', 'System Analyst', 6000)

New row added to JOBS table:
SY_ANAL System Analyst 6000 12000
PL/SQL procedure successfully completed.
```

   c. Verifique si se ha agregado una fila y anote el identificador del nuevo trabajo para utilizarlo en el siguiente ejercicio. Confirme los cambios.

```
SELECT *
FROM   jobs
WHERE  job_id = 'SY_ANAL';
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
SY_ANAL	System Analyst	6000	12000

```
COMMIT;

Commit complete.
```

Parte A: Soluciones a la Práctica Adicional 2

2. En este ejercicio, cree un programa para agregar una nueva fila a la tabla JOB_HISTORY para un empleado existente.

   a. Cree un procedimiento almacenado denominado ADD_JOB_HIST para agregar una nueva fila a la tabla JOB_HISTORY para un empleado que está cambiando su trabajo al nuevo identificador de trabajo ('SY_ANAL') creado en el ejercicio 1b.

      El procedimiento debe proporcionar dos parámetros: uno para el identificador del empleado que está cambiando el trabajo y el otro para el identificador del nuevo trabajo. Consulte el identificador del empleado en la tabla EMPLOYEES e introdúzcalo en la tabla JOB_HISTORY. Indique la fecha de contratación de este empleado como fecha de inicio y la fecha de hoy como fecha final para esta fila en la tabla JOB_HISTORY.

      Cambie la fecha de contratación de este empleado en la tabla EMPLOYEES a la fecha de hoy. Actualice el identificador de trabajo de este empleado según el identificador de trabajo transferido como parámetro (utilice el identificador de trabajo 'SY_ANAL') y el salario igual al salario mínimo para ese identificador de trabajo + 500.
      Nota: Incluya el manejo de excepciones para manejar el intento de insertar un empleado no existente.

```
CREATE OR REPLACE PROCEDURE add_job_hist(
  emp_id    IN employees.employee_id%TYPE,
  new_jobid IN jobs.job_id%TYPE) IS
BEGIN
  INSERT INTO job_history
    SELECT employee_id, hire_date, SYSDATE, job_id, department_id
    FROM employees
    WHERE  employee_id = emp_id;
  UPDATE employees
    SET  hire_date = SYSDATE,
         job_id = new_jobid,
         salary = (SELECT min_salary + 500
                   FROM   jobs
                   WHERE  job_id = new_jobid)
   WHERE employee_id = emp_id;
  DBMS_OUTPUT.PUT_LINE ('Added employee ' || emp_id ||
                        ' details to the JOB_HISTORY table');
  DBMS_OUTPUT.PUT_LINE ('Updated current job of employee ' ||
                        emp_id|| ' to '|| new_jobid);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR (-20001, 'Employee does not exist!');
END add_job_hist;
/
SHOW ERRORS

Procedure created.

No errors.
```

Parte A: Soluciones a la Práctica Adicional 2 (continuación)

b. Desactive todos los disparadores de las tablas EMPLOYEES, JOBS y JOB_HISTORY antes de llamar al procedimiento ADD_JOB_HIST.

```
ALTER TABLE employees DISABLE ALL TRIGGERS;
ALTER TABLE jobs DISABLE ALL TRIGGERS;
ALTER TABLE job_history DISABLE ALL TRIGGERS;

Table altered.

Table altered.

Table altered.
```

c. Ejecute el procedimiento con el identificador de empleado 106 y el identificador de trabajo 'SY_ANAL' como parámetros.

```
EXECUTE add_job_hist(106, 'SY_ANAL')

Added employee 106 details to the JOB_HISTORY table
Updated current job of employee 106 to SY_ANAL
PL/SQL procedure successfully completed.
```

d. Consulte las tablas JOB_HISTORY y EMPLOYEES para ver los cambios del empleado 106 y, a continuación, confirme los cambios.

```
SELECT *   FROM   job_history
WHERE  employee_id = 106;

SELECT job_id, salary   FROM   employees
WHERE  employee_id = 106;

COMMIT;
```

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
106	05-FEB-98	22-FEB-04	IT_PROG	60

JOB_ID	SALARY
SY_ANAL	6500

```
Commit complete.
```

e. Vuelva a activar los disparadores de las tablas EMPLOYEES, JOBS y JOB_HISTORY.

```
ALTER TABLE employees ENABLE ALL TRIGGERS;
ALTER TABLE jobs ENABLE ALL TRIGGERS;
ALTER TABLE job_history ENABLE ALL TRIGGERS;

Table altered.

Table altered.

Table altered.
```

Parte A: Soluciones a la Práctica Adicional 3

3. En este ejercicio, cree un programa para actualizar los salarios mínimo y máximo de un trabajo de la tabla JOBS.

   a. Cree un procedimiento almacenado denominado UPD_JOBSAL para actualizar los salarios mínimo y máximo de un identificador de trabajo concreto de la tabla JOBS. El procedimiento debe proporcionar tres parámetros: el identificador de trabajo, un nuevo salario mínimo y un nuevo salario máximo. Agregue el manejo de excepciones para justificar un identificador de trabajo no válido en la tabla JOBS. Emita una excepción si el salario máximo proporcionado es menor que el salario mínimo. Proporcione un mensaje que se mostrará si la fila de la tabla JOBS está bloqueada.
   Indicación: El número de error de recurso bloqueado/ocupado es −54.

```
CREATE OR REPLACE PROCEDURE upd_jobsal(
  jobid       IN jobs.job_id%type,
  new_minsal  IN jobs.min_salary%type,
  new_maxsal  IN jobs.max_salary%type) IS
  dummy            PLS_INTEGER;
  e_resource_busy  EXCEPTION;
  sal_error        EXCEPTION;
  PRAGMA           EXCEPTION_INIT (e_resource_busy , -54);
BEGIN
  IF (new_maxsal < new_minsal) THEN
    RAISE sal_error;
  END IF;
  SELECT 1 INTO dummy
    FROM jobs
    WHERE job_id = jobid
    FOR UPDATE OF min_salary NOWAIT;
  UPDATE jobs
    SET min_salary =  new_minsal,
        max_salary =  new_maxsal
    WHERE job_id  = jobid;
EXCEPTION
  WHEN e_resource_busy THEN
    RAISE_APPLICATION_ERROR (-20001,
      'Job information is currently locked, try later.');
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20001, 'This job ID does not exist');
  WHEN sal_error THEN
    RAISE_APPLICATION_ERROR(-20001,
     'Data error: Max salary should be more than min salary');
END upd_jobsal;
/
SHOW ERRORS

Procedure created.

No errors.
```

b. Ejecute el procedimiento `UPD_JOBSAL` utilizando un identificador de trabajo `'SY_ANAL'`, un salario mínimo de `7000` y un salario máximo de `140`.
Nota: Esto debe generar un mensaje de excepción.

```
EXECUTE upd_jobsal('SY_ANAL', 7000, 140)

BEGIN upd_jobsal('SY_ANAL', 7000, 140); END;

*

ERROR at line 1:
ORA-20001: Data error: Max salary should be more than min salary
ORA-06512: at "ORA1.UPD_JOBSAL", line 28
ORA-06512: at line 1
```

c. Desactive los disparadores de las tablas `EMPLOYEES` y `JOBS`.

```
ALTER TABLE employees DISABLE ALL TRIGGERS;
ALTER TABLE jobs DISABLE ALL TRIGGERS;

Table altered.

Table altered.
```

d. Ejecute el procedimiento `UPD_JOBSAL` utilizando un identificador de trabajo `'SY_ANAL'`, un salario mínimo de `7000` y un salario máximo de `14000`.

```
EXECUTE upd_jobsal('SY_ANAL', 7000, 14000)

PL/SQL procedure successfully completed.
```

e. Consulte la tabla `JOBS` para ver los cambios y, a continuación, confírmelos.

```
SELECT *
FROM  jobs
WHERE job_id = 'SY_ANAL';
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
SY_ANAL	System Analyst	7000	14000

f. Active los disparadores de las tablas `EMPLOYEES` y `JOBS`.

```
ALTER TABLE employees ENABLE ALL TRIGGERS;
ALTER TABLE jobs ENABLE ALL TRIGGERS;

Table altered.

Table altered.
```

Parte A: Soluciones a la Práctica Adicional 4

4. En este ejercicio, cree un procedimiento para controlar si los salarios de los empleados han excedido la media para el tipo de trabajo.

   a. Desactive el disparador SECURE_EMPLOYEES.

```
ALTER TRIGGER secure_employees DISABLE;

Trigger altered.
```

   b. En la tabla EMPLOYEES, agregue una columna EXCEED_AVGSAL para almacenar hasta tres caracteres y un valor por defecto 'NO'. Utilice una restricción de control para permitir los valores 'YES' o 'NO'.

```
ALTER TABLE employees (
  ADD (exceed_avgsal VARCHAR2(3) DEFAULT 'NO'
    CONSTRAINT employees_exceed_avgsal_ck
    CHECK (exceed_avgsal IN ('YES', 'NO')));

Table altered.
```

   c. Escriba un procedimiento almacenado denominado CHECK_AVGSAL que comprueba si el salario de cada empleado excede el salario medio para JOB_ID. El salario medio de un trabajo se calcula a partir de la información de la tabla JOBS. Si el salario del empleado excede la media para el trabajo, actualice la columna EXCEED_AVGSAL de la tabla EMPLOYEES con el valor YES; de lo contrario, defina el valor en NO. Utilice un cursor para seleccionar las filas de los empleados con la opción FOR UPDATE de la consulta. Agregue el manejo de excepciones para justificar el bloqueo de un registro.
   Indicación: El número de error de recurso bloqueado/ocupado es –54. Escriba y utilice una función local denominada GET_JOB_AVGSAL para determinar el salario medio de un identificador de trabajo especificado como parámetro.

```
CREATE OR REPLACE PROCEDURE check_avgsal IS
  avgsal_exceeded employees.exceed_avgsal%type;
  CURSOR  emp_csr IS
    SELECT  employee_id, job_id, salary
    FROM employees
    FOR UPDATE;
  e_resource_busy  EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_resource_busy, -54);
```

```
  FUNCTION get_job_avgsal (jobid VARCHAR2) RETURN NUMBER IS
    avg_sal employees.salary%type;
  BEGIN
    SELECT (max_salary + min_salary)/2 INTO avg_sal
    FROM jobs
    WHERE job_id = jobid;
    RETURN avg_sal;
  END;

BEGIN
  FOR emprec IN emp_csr
  LOOP
    avgsal_exceeded := 'NO';
    IF emprec.salary >= get_job_avgsal(emprec.job_id) THEN
      avgsal_exceeded := 'YES';
    END IF;
    UPDATE employees
      SET exceed_avgsal = avgsal_exceeded
      WHERE CURRENT OF emp_csr;
  END LOOP;
EXCEPTION
  WHEN e_resource_busy THEN
    ROLLBACK;
    RAISE_APPLICATION_ERROR (-20001, 'Record is busy, try later.');
END check_avgsal;
/
SHOW ERRORS

Procedure created.

No errors.
```

d. Ejecute el procedimiento `CHECK_AVGSAL`. A continuación, para ver los resultados de las modificaciones, escriba una consulta para mostrar el identificador del empleado, el trabajo, el salario medio del trabajo, el salario del empleado y la columna del indicador `exceed_avgsal` para empleados cuyos salarios exceden la media para el trabajo. Por último, confirme los cambios.

```
EXECUTE check_avgsal

SELECT e.employee_id, e.job_id, (j.max_salary-j.min_salary/2) job_avgsal,
       e.salary, e.exceed_avgsal avg_exceeded
FROM   employees e, jobs j
WHERE  e.job_id = j.job_id
and e.exceed_avgsal = 'YES';

COMMIT;
```

```
PL/SQL procedure successfully completed.
```

EMPLOYEE_ID	JOB_ID	JOB_AVGSAL	SALARY	AVG_EXCEE
103	IT_PROG	8000	9000	YES
109	FI_ACCOUNT	6900	9000	YES
110	FI_ACCOUNT	6900	8200	YES
111	FI_ACCOUNT	6900	7700	YES
112	FI_ACCOUNT	6900	7800	YES
113	FI_ACCOUNT	6900	6900	YES
:	:	:	:	:
226	IT_PROG	8000	9000	YES
201	MK_MAN	10500	13000	YES
203	HR_REP	7000	6500	YES
204	PR_REP	8250	10000	YES
206	AC_ACCOUNT	6900	8300	YES

```
31 rows selected.

Commit complete.
```

Parte A: Soluciones a la Práctica Adicional 5

5. Cree un subprograma para recuperar el número de años de servicio de un empleado concreto.

   a. Cree una función almacenada denominada GET_YEARS_SERVICE para recuperar el número total de años de servicio de un empleado concreto. La función debe aceptar el identificador de empleado como parámetro y devolver el número de años de servicio. Agregue la gestión de errores para justificar un identificador de empleado no válido.

```
CREATE OR REPLACE FUNCTION get_years_service(
  emp_id IN employees.employee_id%TYPE) RETURN NUMBER IS
  CURSOR jobh_csr IS
    SELECT MONTHS_BETWEEN(end_date, start_date)/12) years_in_job
    FROM   job_history
    WHERE  employee_id = emp_id;
    years_service NUMBER(2) := 0;
    years_in_job  NUMBER(2) := 0;
BEGIN
  FOR jobh_rec IN jobh_csr
  LOOP
    EXIT WHEN jobh_csr%NOTFOUND;
    years_service := years_service + job_rec.years_in_job;
  END LOOP;
  SELECT MONTHS_BETWEEN(SYSDATE, hire_date)/12 INTO years_in_job
  FROM   employees
  WHERE  employee_id = emp_id;
  years_service := years_service + years_in_job;
  RETURN ROUND(years_service);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20348,
      'Employee with ID '|| emp_id ||' does not exist.');
END get_years_service;
/
SHOW ERRORS

Function created.

No errors.
```

   b. Llame a la función GET_YEARS_SERVICE en una llamada a DBMS_OUTPUT.PUT_LINE para un empleado que tenga el identificador 999.

```
EXECUTE DBMS_OUTPUT.PUT_LINE(get_years_service (999))

BEGIN DBMS_OUTPUT.PUT_LINE(get_years_service (999)); END;

*

ERROR at line 1:
ORA-20348: Employee with ID 999 does not exist.
ORA-06512: at "ORA1.GET_YEARS_SERVICE", line 22
ORA-06512: at line 1
```

c. Muestre el número de años de servicio para el empleado 106 mediante la llamada de `DBMS_OUTPUT.PUT_LINE` a la función `GET_YEARS_SERVICE`.

```
BEGIN
  DBMS_OUTPUT.PUT_LINE (
    'Employee 106 has worked ' || get_years_service(106) || ' years');
END;
/

Employee 106 has worked 6 years
PL/SQL procedure successfully completed.
```

d. Consulte las tablas `JOB_HISTORY` y `EMPLOYEES` del empleado especificado para verificar que las modificaciones son precisas.
Nota: Los valores representados en los resultados de esta página pueden variar de los valores obtenidos al ejecutar las consultas.

```
SELECT employee_id, job_id,
       MONTHS_BETWEEN(end_date, start_date)/12 duration
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID	DURATION
102	IT_PROG	5.52956989
101	AC_ACCOUNT	4.09946237
101	AC_MGR	3.38172043
201	MK_REP	3.83870968
114	ST_CLERK	1.7688172
122	ST_CLERK	.997311828
200	AD_ASST	5.75
176	SA_REP	.768817204
176	SA_MAN	.997311828
200	AC_ACCOUNT	4.49731183
106	IT_PROG	6.04765846

```
11 rows selected.

SELECT job_id, MONTHS_BETWEEN(SYSDATE, hire_date)/12 duration
FROM   employees
WHERE  employee_id = 106;
```

JOB_ID	DURATION
SY_ANAL	0

Parte A: Soluciones a la Práctica Adicional 6

6. En este ejercicio, cree un programa para recuperar el número de trabajos diferentes que ha tenido un empleado durante su servicio.

   a. Cree una función almacenada denominada `GET_JOB_COUNT` para recuperar el número total de trabajos diferentes que ha tenido un empleado.

      La función debe aceptar el identificador de empleado en un parámetro y devolver el número de trabajos diferentes que ha tenido un empleado hasta la fecha, incluido el presente. Agregue el manejo de excepciones para justificar un identificador de empleado no válido.
      Indicación: Utilice los distintos identificadores de trabajo de la tabla `JOB_HISTORY` y excluya el identificador de trabajo actual, si se trata de uno de los identificadores de trabajo en los que ya ha trabajado el empleado. Una dos consultas con `UNION` y cuente las filas recuperadas en una tabla PL/SQL. Utilice `FETCH` con `BULK COLLECT INTO` para obtener los trabajos únicos del empleado.

```
CREATE OR REPLACE FUNCTION get_job_count(
  emp_id IN employees.employee_id%TYPE) RETURN NUMBER IS
  TYPE jobs_tabtype IS TABLE OF jobs.job_id%type;
  jobtab jobs_tabtype;
  CURSOR empjob_csr IS
   SELECT job_id
   FROM job_history
   WHERE employee_id = emp_id
   UNION
   SELECT job_id
   FROM employees
   WHERE employee_id = emp_id;
BEGIN
  OPEN empjob_csr;
  FETCH empjob_csr BULK COLLECT INTO jobtab;
  CLOSE empjob_csr;
  RETURN jobtab.count;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20348,
      'Employee with ID '|| emp_id ||' does not exist!');
END get_job_count;
/
SHOW ERRORS

Function created.

No errors.
```

b. Llame a la función para el empleado con el identificador 176.

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('Employee 176 worked on ' ||
     get_job_count(176) || ' different jobs.');
END;
/

Employee 176 worked on 2 different jobs.
PL/SQL procedure successfully completed.
```

Parte A: Soluciones a la Práctica Adicional 7

7. Cree un paquete denominado EMPJOB_PKG que contenga los procedimientos NEW_JOB, ADD_JOB_HIST y UPD_JOBSAL, así como las funciones GET_YEARS_SERVICE y GET_JOB_COUNT.

   a. Cree la especificación del paquete con toda la construcción del subprograma pública. Mueva cualquier tipo definido localmente del subprograma a la especificación del paquete.

```
CREATE OR REPLACE PACKAGE empjob_pkg IS
  TYPE jobs_tabtype IS TABLE OF jobs.job_id%type;

  PROCEDURE add_job_hist(
    emp_id IN employees.employee_id%TYPE,
    new_jobid IN jobs.job_id%TYPE);
  FUNCTION get_job_count(
   emp_id IN employees.employee_id%TYPE) RETURN NUMBER;
   FUNCTION get_years_service(
   emp_id IN employees.employee_id%TYPE) RETURN NUMBER;
  PROCEDURE new_job(
   jobid IN jobs.job_id%TYPE,
    title IN jobs.job_title%TYPE,
    minsal IN jobs.min_salary%TYPE);
  PROCEDURE upd_jobsal(
    jobid IN jobs.job_id%type,
    new_minsal IN jobs.min_salary%type,
    new_maxsal IN jobs.max_salary%type);
END empjob_pkg;
/
SHOW ERRORS

Package created.

No errors.
```

Parte A: Soluciones a la Práctica Adicional 7 (continuación)

b. Cree el cuerpo del paquete con la implementación del subprograma; recuerde eliminar de las implementaciones del subprograma cualquier tipo que haya movido a la especificación del paquete.

```
CREATE OR REPLACE PACKAGE BODY empjob_pkg IS
  PROCEDURE add_job_hist(
    emp_id IN employees.employee_id%TYPE,
    new_jobid IN jobs.job_id%TYPE) IS
  BEGIN
    INSERT INTO job_history
      SELECT employee_id, hire_date, SYSDATE, job_id, department_id
      FROM employees
      WHERE employee_id = emp_id;
    UPDATE employees
      SET hire_date = SYSDATE,
          job_id = new_jobid,
          salary = (SELECT min_salary + 500
                    FROM jobs
          WHERE job_id = new_jobid)
      WHERE employee_id = emp_id;
    DBMS_OUTPUT.PUT_LINE ('Added employee ' || emp_id ||
       ' details to the JOB_HISTORY table');
    DBMS_OUTPUT.PUT_LINE ('Updated current job of employee ' ||
       emp_id|| ' to '|| new_jobid);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR (-20001, 'Employee does not exist!');
  END add_job_hist;

  FUNCTION get_job_count(
    emp_id IN employees.employee_id%TYPE) RETURN NUMBER IS
    jobtab jobs_tabtype;
    CURSOR empjob_csr IS
      SELECT job_id
      FROM job_history
      WHERE employee_id = emp_id
      UNION
      SELECT job_id
      FROM employees
      WHERE employee_id = emp_id;
  BEGIN
    OPEN empjob_csr;
    FETCH empjob_csr BULK COLLECT INTO jobtab;
    CLOSE empjob_csr;
    RETURN jobtab.count;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR(-20348,
        'Employee with ID '|| emp_id ||' does not exist!');
  END get_job_count;
```

Parte A: Soluciones a la Práctica Adicional 7 (continuación)

```
FUNCTION get_years_service(
  emp_id IN employees.employee_id%TYPE) RETURN NUMBER IS
  CURSOR jobh_csr IS
    SELECT MONTHS_BETWEEN(end_date, start_date)/12 years_in_job
    FROM job_history
    WHERE employee_id = emp_id;
  years_service NUMBER(2) := 0;
  years_in_job NUMBER(2) := 0;
BEGIN
  FOR jobh_rec IN jobh_csr
  LOOP
    EXIT WHEN jobh_csr%NOTFOUND;
    years_service := years_service + jobh_rec.years_in_job;
  END LOOP;
  SELECT MONTHS_BETWEEN(SYSDATE, hire_date)/12 INTO years_in_job
  FROM employees
  WHERE  employee_id = emp_id;
  years_service := years_service + years_in_job;
  RETURN ROUND(years_service);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20348,
      'Employee with ID '|| emp_id ||' does not exist.');
END get_years_service;

PROCEDURE new_job(
  jobid  IN jobs.job_id%TYPE,
  title  IN jobs.job_title%TYPE,
  minsal IN jobs.min_salary%TYPE) IS
  maxsal  jobs.max_salary%TYPE := 2 * minsal;
BEGIN
  INSERT INTO jobs(job_id, job_title, min_salary, max_salary)
  VALUES (jobid, title, minsal, maxsal);
  DBMS_OUTPUT.PUT_LINE ('New row added to JOBS table:');
  DBMS_OUTPUT.PUT_LINE (jobid ||' ' || title ||' '||
                        minsal ||'  ' || maxsal);
END new_job;
```

```
  PROCEDURE upd_jobsal(
    jobid IN jobs.job_id%type,
    new_minsal  IN jobs.min_salary%type,
    new_maxsal  IN jobs.max_salary%type) IS
    dummy PLS_INTEGER;
    e_resource_busy  EXCEPTION;
    sal_error EXCEPTION;
    PRAGMA EXCEPTION_INIT (e_resource_busy , -54);
  BEGIN
    IF (new_maxsal < new_minsal) THEN
      RAISE sal_error;
    END IF;
    SELECT 1 INTO dummy
    FROM jobs
    WHERE job_id = jobid
    FOR UPDATE OF min_salary NOWAIT;
    UPDATE jobs
      SET min_salary =  new_minsal,
          max_salary =  new_maxsal
    WHERE job_id = jobid;
  EXCEPTION
    WHEN e_resource_busy THEN
      RAISE_APPLICATION_ERROR (-20001,
        'Job information is currently locked, try later.');
    WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR(-20001, 'This job ID does not exist');
    WHEN sal_error THEN
      RAISE_APPLICATION_ERROR(-20001,
        'Data error: Max salary should be more than min salary');
  END upd_jobsal;
END empjob_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

c. Llame al procedimiento `EMPJOB_PKG.NEW_JOB` para crear un nuevo trabajo con el identificador `PR_MAN`, el título `Public Relations Manager` y el salario `6,250`.

```
EXECUTE empjob_pkg.new_job('PR_MAN', 'Public Relations Manager', 6250)

New row added to JOBS table:
PR_MAN Public Relations Manager 6250 12500
PL/SQL procedure successfully completed.
```

Parte A: Soluciones a la Práctica Adicional 7 (continuación)

d. Llame al procedimiento `EMPJOB_PKG.ADD_JOB_HIST` para modificar el trabajo del empleado con identificador `110` por el identificador de trabajo `PR_MAN`.
Nota: Debe desactivar el disparador `UPDATE_JOB_HISTORY` antes de ejecutar el procedimiento `ADD_JOB_HIST` y volver a activar el disparador después de ejecutar el procedimiento.

```
ALTER TRIGGER update_job_history DISABLE;
EXECUTE empjob_pkg.add_job_hist(110, 'PR_MAN')
ALTER TRIGGER update_job_history ENABLE;

Trigger altered.

Added employee 110 details to the JOB_HISTORY table
Updated current job of employee 110 to PR_MAN
PL/SQL procedure successfully completed.

Trigger altered.
```

e. Consulte las tablas `JOBS`, `JOB_HISTORY` y `EMPLOYEES` para verificar los resultados.

```
SELECT * FROM jobs WHERE job_id = 'PR_MAN';
SELECT * FROM hr.employees WHERE employee_id = 110;
SELECT job_id, salary FROM employees WHERE employee_id = 110;
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
PR_MAN	Public Relations Manager	6250	12500

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
110	28-SEP-97	22-FEB-04	FI_ACCOUNT	100

JOB_ID	SALARY
PR_MAN	6750

Parte A: Soluciones a la Práctica Adicional 8

8. En este ejercicio, cree un disparador para asegurarse de que los salarios mínimo y máximo de un trabajo nunca se modifiquen de forma que el salario de un empleado existente con ese identificador de trabajo esté fuera del nuevo rango especificado para el trabajo.

   a. Cree un disparador denominado `CHECK_SAL_RANGE` que se dispare antes de cada fila que se actualice en las columnas `MIN_SALARY` y `MAX_SALARY` de la tabla `JOBS`. Para cada valor de salario mínimo o máximo que se cambie, compruebe si el salario de cualquier empleado existente con ese identificador de trabajo de la tabla `EMPLOYEES` entra dentro del nuevo rango de salarios especificado para este identificador de trabajo. Incluya el manejo de excepciones para cubrir un rango de salarios que afecte al registro de cualquier empleado existente.

```
CREATE OR REPLACE TRIGGER check_sal_range
BEFORE UPDATE OF min_salary, max_salary ON jobs
FOR EACH ROW
DECLARE
  minsal employees.salary%TYPE;
  maxsal employees.salary%TYPE;
  e_invalid_salrange EXCEPTION;
BEGIN
  SELECT MIN(salary), MAX(salary) INTO minsal, maxsal
  FROM employees
  WHERE job_id = :NEW.job_id;
  IF (minsal < :NEW.min_salary) OR (maxsal > :NEW.max_salary) THEN
    RAISE e_invalid_salrange;
  END IF;
EXCEPTION
  WHEN e_invalid_salrange THEN
    RAISE_APPLICATION_ERROR(-20550,
     'Employees exist whose salary is out of the specified range. '||
     'Therefore the specified salary range cannot be updated.');
END check_sal_range;
/
SHOW ERRORS

Trigger created.

No errors.
```

Parte A: Soluciones a la Práctica Adicional 8 (continuación)

b. Pruebe el disparador utilizando el trabajo SY_ANAL y defina el nuevo salario mínimo en 5000 y el nuevo salario máximo en 7000. Antes de realizar los cambios, escriba una consulta para mostrar el rango de salarios actual para el identificador de trabajo SY_ANAL y otra consulta para mostrar el identificador, apellido y salario del empleado para el mismo identificador de trabajo. Después de la actualización, consulte los cambios (si existen) realizados en la tabla JOBS para el identificador de trabajo especificado.

```
SELECT * FROM jobs
WHERE job_id = 'SY_ANAL';
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
SY_ANAL	System Analyst	6000	12000

```
SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'SY_ANAL';
```

EMPLOYEE_ID	LAST_NAME	SALARY
106	Pataballa	6500

```
UPDATE jobs
  SET min_salary = 5000, max_salary = 7000
  WHERE job_id = 'SY_ANAL';

1 row updated.

SELECT * FROM jobs
WHERE job_id = 'SY_ANAL';
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
SY_ANAL	System Analyst	5000	7000

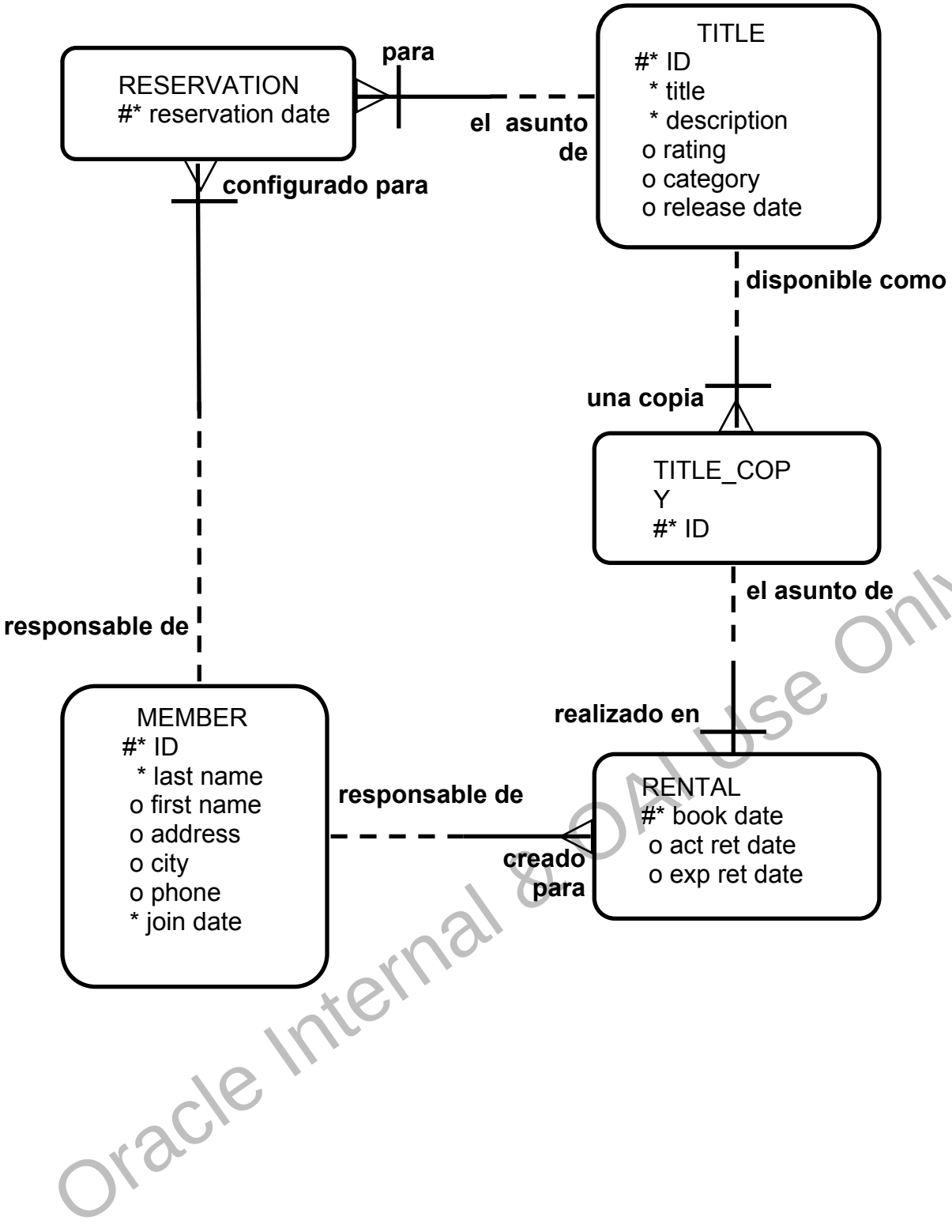
c. Mediante el trabajo SY_ANAL, defina el nuevo salario mínimo en 7000 y el nuevo salario máximo en 18000. Explique los resultados.

```
UPDATE jobs
  SET min_salary = 7000, max_salary = 18000
  WHERE  job_id = 'SY_ANAL';

UPDATE jobs
       *

ERROR at line 1:
ORA-20550: Employees exist whose salary is out of the specified range.
Therefore the specified salary range cannot be updated.
ORA-06512: at "ORA1.CHECK_SAL_RANGE", line 14
ORA-04088: error during execution of trigger 'ORA1.CHECK_SAL_RANGE'
```

Parte A: Soluciones a la Práctica Adicional 8 (continuación)

La actualización no consigue cambiar el rango de salarios debido a la funcionalidad que proporciona el el disparador `CHECK_SAL_RANGE`, porque el empleado 106 con identificador de trabajo `SY_ANAL` tiene un salario de 6500, cifra inferior al salario mínimo para el nuevo rango de salarios especificado en la sentencia `UPDATE`.

Parte B: Diagrama de Entidad Relacional



Parte B (continuación)

En este caso práctico, creará un paquete denominado `VIDEO_PKG` que contenga los procedimientos y funciones de una aplicación de un videoclub. Esta aplicación permite a los clientes convertirse en miembros del videoclub. Cualquier miembro podrá alquilar películas, devolver películas alquiladas y reservar películas. Además, creará un disparador para asegurarse de que los datos de las tablas de vídeos se modifican sólo durante las horas laborables.
Cree el paquete con *i*SQL*Plus y utilice el paquete `DBMS_OUTPUT` proporcionado por Oracle para mostrar los mensajes.
La base de datos del videoclub contiene las siguientes tablas: `TITLE`, `TITLE_COPY`, `RENTAL`, `RESERVATION` y `MEMBER`. El diagrama de entidad/relación aparece en la página anterior.

Parte B: Soluciones a la Práctica Adicional 1

1. Cargue y ejecute el archivo de comandos `E:\labs\PLPU\labs\buildvid1.sql` para crear todas las tablas y secuencias necesarias para este ejercicio.

```
SET ECHO OFF
/* Script to build the Video Application (Part 1 - buildvid1.sql)
   for the Oracle Introduction to Oracle with Procedure Builder course.
   Created by: Debby Kramer Creation date: 12/10/95
   Last upated: 2/13/96
   Modified by Nagavalli Pataballa on 26-APR-2001
    For the course Introduction to Oracle9i: PL/SQL
    This part of the script creates tables and sequences that are used
    by Part B of the Additional Practices of the course.
*/

DROP TABLE rental CASCADE CONSTRAINTS;
DROP TABLE reservation CASCADE CONSTRAINTS;
DROP TABLE title_copy CASCADE CONSTRAINTS;
DROP TABLE title CASCADE CONSTRAINTS;
DROP TABLE member CASCADE CONSTRAINTS;

PROMPT Please wait while tables are created....

CREATE TABLE MEMBER
  (member_id  NUMBER (10)        CONSTRAINT member_id_pk PRIMARY KEY
 , last_name  VARCHAR2(25)
    CONSTRAINT member_last_nn NOT NULL
 , first_name VARCHAR2(25)
 , address    VARCHAR2(100)
 , city       VARCHAR2(30)
 , phone      VARCHAR2(25)
 , join_date  DATE DEFAULT SYSDATE
    CONSTRAINT join_date_nn NOT NULL)
/

CREATE TABLE TITLE
  (title_id   NUMBER(10)
     CONSTRAINT title_id_pk PRIMARY KEY
 , title      VARCHAR2(60)
     CONSTRAINT title_nn NOT NULL
 , description VARCHAR2(400)
     CONSTRAINT title_desc_nn NOT NULL
 , rating      VARCHAR2(4)
     CONSTRAINT title_rating_ck CHECK (rating IN
('G','PG','R','NC17','NR'))
 , category     VARCHAR2(20) DEFAULT 'DRAMA'
     CONSTRAINT title_categ_ck CHECK (category IN
('DRAMA','COMEDY','ACTION', 'CHILD','SCIFI','DOCUMENTARY'))
 , release_date DATE)
/
```

Parte B: Soluciones a la Práctica Adicional 1 (continuación)

```
CREATE TABLE TITLE_COPY
  (copy_id   NUMBER(10)
 , title_id  NUMBER(10)
    CONSTRAINT copy_title_id_fk
       REFERENCES title(title_id)
 , status    VARCHAR2(15)
     CONSTRAINT copy_status_nn NOT NULL
     CONSTRAINT copy_status_ck CHECK (status IN ('AVAILABLE',
'DESTROYED',
                                    'RENTED', 'RESERVED'))
 , CONSTRAINT copy_title_id_pk  PRIMARY KEY(copy_id, title_id))
/
CREATE TABLE RENTAL
  (book_date DATE DEFAULT SYSDATE
 , copy_id   NUMBER(10)
 , member_id NUMBER(10)
    CONSTRAINT rental_mbr_id_fk REFERENCES member(member_id)
 , title_id  NUMBER(10)
 , act_ret_date DATE
 , exp_ret_date DATE DEFAULT SYSDATE+2
 , CONSTRAINT rental_copy_title_id_fk FOREIGN KEY (copy_id, title_id)
              REFERENCES title_copy(copy_id,title_id)
 , CONSTRAINT rental_id_pk PRIMARY KEY(book_date, copy_id, title_id,
member_id))
/
CREATE TABLE RESERVATION
  (res_date  DATE
 , member_id NUMBER(10)
 , title_id  NUMBER(10)
 , CONSTRAINT res_id_pk PRIMARY KEY(res_date, member_id, title_id))
/

PROMPT Tables created.
DROP SEQUENCE title_id_seq;
DROP SEQUENCE member_id_seq;

PROMPT Creating Sequences...
CREATE SEQUENCE member_id_seq
  START WITH 101
  NOCACHE

CREATE SEQUENCE title_id_seq
  START WITH 92
  NOCACHE
/

PROMPT Sequences created.

PROMPT Run buildvid2.sql now to populate the above tables.
```

Parte B: Soluciones a la Práctica Adicional 2

2. Cargue y ejecute el archivo de comandos `E:\labs\PLPU\labs\buildvid2.sql` para rellenar todas las tablas creadas por el archivo de comandos `buildvid1.sql`.

```
/* Script to build the Video Application (Part 2 - buildvid2.sql)
   This part of the script populates the tables that are created using
   buildvid1.sql
   These are used by Part B of the Additional Practices of the course.
   You should run the script buildvid1.sql before running this script to
   create the above tables.
*/

INSERT INTO member
  VALUES  (member_id_seq.NEXTVAL, 'Velasquez', 'Carmen',
    '283 King Street', 'Seattle', '587-99-6666', '03-MAR-90');
INSERT INTO member
  VALUES   (member_id_seq.NEXTVAL, 'Ngao', 'LaDoris',
    '5 Modrany',  'Bratislava', '586-355-8882', '08-MAR-90');
INSERT INTO member
  VALUES  (member_id_seq.NEXTVAL,'Nagayama', 'Midori',
    '68 Via Centrale', 'Sao Paolo', '254-852-5764', '17-JUN-91');
INSERT INTO member
  VALUES  (member_id_seq.NEXTVAL,'Quick-To-See','Mark',
    '6921 King Way', 'Lagos', '63-559-777', '07-APR-90');
INSERT INTO member
   VALUES  (member_id_seq.NEXTVAL, 'Ropeburn', 'Audry',
    '86 Chu Street',  'Hong Kong', '41-559-87', '04-MAR-90');
INSERT INTO member
  VALUES (member_id_seq.NEXTVAL, 'Urguhart', 'Molly',
    '3035 Laurier Blvd.',  'Quebec', '418-542-9988', '18-JAN-91');
INSERT INTO member
  VALUES (member_id_seq.NEXTVAL, 'Menchu', 'Roberta',
    'Boulevard de Waterloo 41', 'Brussels', '322-504-2228', '14-MAY-90');
INSERT INTO member
  VALUES (member_id_seq.NEXTVAL, 'Biri', 'Ben',
    '398 High St.', 'Columbus', '614-455-9863', '07-APR-90');
INSERT INTO member
  VALUES (member_id_seq.NEXTVAL, 'Catchpole', 'Antoinette',
    '88 Alfred St.', 'Brisbane', '616-399-1411', '09-FEB-92');

COMMIT;
```

Parte B: Soluciones a la Práctica Adicional 2 (continuación)

```
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 made a Christmas list for Santa, but Willie
has yet to create his own wish list.', 'G', 'CHILD', '05-OCT-95');
INSERT INTO TITLE (title_id, title, description, rating, category,
release_date)
  VALUES (TITLE_ID_SEQ.NEXTVAL, 'Alien Again', 'Another installment of
science fiction history. Can the heroine save the planet from the alien
life form?', 'R', 'SCIFI',             '19-MAY-95');
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 carivorous goo in this classic.', 'NR',
'SCIFI', '12-AUG-95');
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 of ingenuity, a teenager skips school for a day in New York.', 'PG',
'COMEDY', '12-JUL-95');
INSERT INTO TITLE (title_id, title, description, rating, category,
release_date)
  VALUES (TITLE_ID_SEQ.NEXTVAL, 'Miracles on Ice', 'A six-year-old has
doubts about Santa Claus. But she discovers that miracles really do
exist.', 'PG', 'DRAMA', '12-SEP-95');
INSERT INTO TITLE (title_id, title, description, rating, category,
release_date)
  VALUES (TITLE_ID_SEQ.NEXTVAL,  'Soda Gang', 'After discovering a cached
of drugs, a young couple find themselves pitted against a vicious gang.',
'NR', 'ACTION', '01-JUN-95');
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');

COMMIT;

INSERT INTO title_copy VALUES (1,92, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,93, 'AVAILABLE');
INSERT INTO title_copy VALUES (2,93, 'RENTED');
INSERT INTO title_copy VALUES (1,94, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,95, 'AVAILABLE');
INSERT INTO title_copy VALUES (2,95, 'AVAILABLE');
INSERT INTO title_copy VALUES (3,95, 'RENTED');
INSERT INTO title_copy VALUES (1,96, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,97, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,98, 'RENTED');
INSERT INTO title_copy VALUES (2,98, 'AVAILABLE');

COMMIT;
```

Parte B: Soluciones a la Práctica Adicional 2 (continuación)

```
INSERT INTO reservation VALUES (sysdate-1, 101, 93);
INSERT INTO reservation VALUES (sysdate-2, 106, 102);

COMMIT;

INSERT INTO rental VALUES (sysdate-1, 2, 101, 93, null, sysdate+1);
INSERT INTO rental VALUES (sysdate-2, 3, 102, 95, null, sysdate);
INSERT INTO rental VALUES (sysdate-3, 1, 101, 98, null, sysdate-1);
INSERT INTO rental VALUES (sysdate-4, 1, 106, 97, sysdate-2, sysdate-2);
INSERT INTO rental VALUES (sysdate-3, 1, 101, 92, sysdate-2, sysdate-1);

COMMIT;

PROMPT ** Tables built and data loaded **
```

Parte B: Soluciones a la Práctica Adicional 3

3. Cree un paquete denominado VIDEO_PKG con los siguientes procedimientos y funciones:

   a. NEW_MEMBER: Procedimiento público que agrega un nuevo miembro a la tabla MEMBER. Para el número de identificador de miembro, utilice la secuencia MEMBER_ID_SEQ; para la fecha de unión, utilice SYSDATE. Transfiera los demás valores que desee insertar en una nueva fila como parámetros.

   b. NEW_RENTAL: Función pública sobrecargada para registrar un nuevo alquiler. Transfiera el número del identificador de título del vídeo que el cliente desea alquilar junto con el apellido del cliente o el número de identificador de miembro a la función. La función deberá devolver la fecha de vencimiento del vídeo. Las fechas de vencimiento son tres días desde la fecha de alquiler del vídeo. Si el estado de una película solicitada aparece como AVAILABLE en la tabla TITLE_COPY para una copia de este título, actualice la tabla TITLE_COPY y defina el estado en RENTED. Si no hay ninguna copia disponible, la función debe devolver el valor NULL. A continuación, inserte un nuevo registro en la tabla RENTAL que identifique la fecha de reserva como la fecha de hoy, el número de identificador de copia, el número de identificador de miembro, el número de identificador de título y la fecha de devolución esperada. Tenga en cuenta que puede que existan varios clientes con el mismo apellido. En este caso, haga que la función devuelva NULL y muestre una lista de todos los nombres de clientes que coincidan y sus números de identificador.

   c. RETURN_MOVIE: Procedimiento público que actualiza el estado de un vídeo (disponible, alquilado o dañado) y define la fecha de devolución. Transfiera el identificador de título, el identificador de copia y el estado a este procedimiento. Compruebe si existen reservas para ese título y muestre un mensaje, si está reservado. Actualice la tabla RENTAL y defina la fecha de devolución real en la fecha de hoy. Actualice el estado en la tabla TITLE_COPY según el parámetro de estado transferido al procedimiento.

   d. RESERVE_MOVIE: Procedimiento privado que se ejecuta sólo si todas las copias de vídeos solicitadas en el procedimiento NEW_RENTAL tienen el estado RENTED. Transfiera el número de identificador de miembro y el número de identificador de título a este procedimiento. Inserte un nuevo registro en la tabla RESERVATION y registre la fecha de reserva, el número de identificador de miembro y el número de identificador de título. Imprima un mensaje que indique que una película está reservada y la fecha de devolución esperada.

   e. EXCEPTION_HANDLER: Procedimiento privado que se llama desde el manejador de excepciones de los programas públicos. Transfiera el número SQLCODE a este procedimiento y el nombre del programa (como cadena de texto) en el que se ha producido el error. Utilice RAISE_APPLICATION_ERROR para emitir un error personalizado. Empiece por una violación de clave única (-1) y una violación de clave ajena (-2292). Permita que el manejador de excepciones emita un error genérico para cualquier otro error.

Parte B: Soluciones a la Práctica Adicional 3 (continuación)

Especificación del Paquete `VIDEO_PKG`

```
CREATE OR REPLACE PACKAGE video_pkg IS
  PROCEDURE new_member
    (lname       IN member.last_name%TYPE,
     fname       IN member.first_name%TYPE   DEFAULT NULL,
     address     IN member.address%TYPE      DEFAULT NULL,
     city        IN member.city%TYPE         DEFAULT NULL,
     phone       IN member.phone%TYPE        DEFAULT NULL);

  FUNCTION new_rental
    (memberid   IN rental.member_id%TYPE,
     titleid    IN rental.title_id%TYPE)
    RETURN DATE;

  FUNCTION new_rental
    (membername IN member.last_name%TYPE,
     titleid    IN rental.title_id%TYPE)
    RETURN DATE;

  PROCEDURE return_movie
    (titleid    IN rental.title_id%TYPE,
     copyid     IN rental.copy_id%TYPE,
     sts        IN title_copy.status%TYPE);
END video_pkg;
/
SHOW ERRORS

Package created.

No errors.
```

Cuerpo del Paquete `VIDEO_PKG`

```
CREATE OR REPLACE PACKAGE BODY video_pkg IS
  PROCEDURE exception_handler(errcode IN  NUMBER, context IN VARCHAR2) IS
  BEGIN
    IF errcode = -1 THEN
      RAISE_APPLICATION_ERROR(-20001,
        'The number is assigned to this member is already in use, '||
        'try again.');
    ELSIF errcode = -2291 THEN
      RAISE_APPLICATION_ERROR(-20002, context ||
        ' has attempted to use a foreign key value that is invalid');
    ELSE
      RAISE_APPLICATION_ERROR(-20999, 'Unhandled error in ' ||
        context || '. Please contact your application '||
        'administrator with the following information: '
        || CHR(13) || SQLERRM);
    END IF;
  END exception_handler;
```

Parte B: Soluciones a la Práctica Adicional 3 (continuación)

```
  PROCEDURE reserve_movie
    (memberid  IN  reservation.member_id%TYPE,
     titleid   IN  reservation.title_id%TYPE) IS
    CURSOR rented_csr IS
      SELECT exp_ret_date
        FROM rental
        WHERE title_id = titleid
        AND act_ret_date IS NULL;
  BEGIN
    INSERT INTO reservation (res_date, member_id, title_id)
    VALUES (SYSDATE, memberid, titleid);
    COMMIT;
    FOR rented_rec IN rented_csr LOOP
      DBMS_OUTPUT.PUT_LINE('Movie reserved. Expected back on: '
        || rented_rec.exp_ret_date);
      EXIT WHEN rented_csr%found;
    END LOOP;
  EXCEPTION
    WHEN OTHERS THEN
      exception_handler(SQLCODE, 'RESERVE_MOVIE');
  END reserve_movie;

 PROCEDURE return_movie(
   titleid IN rental.title_id%TYPE,
   copyid IN rental.copy_id%TYPE,
   sts IN title_copy.status%TYPE) IS
    v_dummy VARCHAR2(1);
    CURSOR res_csr IS
      SELECT *
      FROM reservation
      WHERE title_id = titleid;
  BEGIN
    SELECT '' INTO v_dummy
      FROM title
      WHERE title_id = titleid;
    UPDATE rental
      SET act_ret_date = SYSDATE
      WHERE title_id = titleid
      AND copy_id = copyid AND act_ret_date IS NULL;
    UPDATE title_copy
      SET status = UPPER(sts)
      WHERE title_id = titleid AND copy_id = copyid;
    FOR res_rec IN res_csr LOOP
      IF res_csr%FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Put this movie on hold -- '||
        'reserved by member #' || res_rec.member_id);
      END IF;
    END LOOP;
  EXCEPTION
    WHEN OTHERS THEN
      exception_handler(SQLCODE, 'RETURN_MOVIE');
  END return_movie;
```

Parte B: Soluciones a la Práctica Adicional 3 (continuación)

```
FUNCTION new_rental(
  memberid  IN  rental.member_id%TYPE,
  titleid   IN  rental.title_id%TYPE) RETURN DATE IS
  CURSOR copy_csr IS
    SELECT * FROM title_copy
    WHERE title_id = titleid
    FOR UPDATE;
  flag   BOOLEAN  := FALSE;
BEGIN

  FOR copy_rec IN copy_csr LOOP
    IF copy_rec.status = 'AVAILABLE' THEN
      UPDATE title_copy
        SET status = 'RENTED'
        WHERE CURRENT OF copy_csr;
      INSERT INTO rental(book_date, copy_id, member_id,
                         title_id, exp_ret_date)
      VALUES (SYSDATE, copy_rec.copy_id, memberid,
                         titleid, SYSDATE + 3);
      flag := TRUE;
      EXIT;
    END IF;
  END LOOP;
  COMMIT;
  IF flag THEN
    RETURN (SYSDATE + 3);
  ELSE
    reserve_movie(memberid, titleid);
    RETURN NULL;
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    exception_handler(SQLCODE, 'NEW_RENTAL');
END new_rental;

FUNCTION new_rental(
  membername IN member.last_name%TYPE,
  titleid   IN  rental.title_id%TYPE) RETURN DATE IS
  CURSOR copy_csr IS
    SELECT * FROM title_copy
      WHERE title_id = titleid
      FOR UPDATE;
  flag   BOOLEAN  := FALSE;
  memberid  member.member_id%TYPE;
  CURSOR member_csr IS
    SELECT member_id, last_name, first_name
      FROM member
      WHERE LOWER(last_name) = LOWER(membername)
      ORDER BY last_name, first_name;
```

Parte B: Soluciones a la Práctica Adicional 3 (continuación)

```
  BEGIN
    SELECT member_id INTO memberid
      FROM member
      WHERE lower(last_name) = lower(membername);
    FOR copy_rec IN copy_csr LOOP
      IF copy_rec.status = 'AVAILABLE' THEN
        UPDATE title_copy
          SET status = 'RENTED'
          WHERE CURRENT OF copy_csr;
        INSERT INTO rental (book_date, copy_id, member_id,
                            title_id, exp_ret_date)
          VALUES (SYSDATE, copy_rec.copy_id, memberid,
                            titleid, SYSDATE + 3);
        flag := TRUE;
        EXIT;
      END IF;
    END LOOP;
    COMMIT;
    IF flag THEN
      RETURN(SYSDATE + 3);
    ELSE
      reserve_movie(memberid, titleid);
      RETURN NULL;
    END IF;
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE(
       'Warning! More than one member by this name.');
      FOR member_rec IN member_csr LOOP
        DBMS_OUTPUT.PUT_LINE(member_rec.member_id || CHR(9) ||
          member_rec.last_name || ', ' || member_rec.first_name);
      END LOOP;
      RETURN NULL;
    WHEN OTHERS THEN
      exception_handler(SQLCODE, 'NEW_RENTAL');
  END new_rental;

  PROCEDURE new_member(
    lname       IN member.last_name%TYPE,
    fname       IN member.first_name%TYPE   DEFAULT NULL,
    address     IN member.address%TYPE      DEFAULT NULL,
    city        IN member.city%TYPE         DEFAULT NULL,
    phone       IN member.phone%TYPE        DEFAULT NULL) IS
  BEGIN
    INSERT INTO member(member_id, last_name, first_name,
                       address, city, phone, join_date)
      VALUES(member_id_seq.NEXTVAL, lname, fname,
             address, city, phone, SYSDATE);
    COMMIT;
```

Parte B: Soluciones a la Práctica Adicional 3 (continuación)

```
  EXCEPTION
    WHEN OTHERS THEN
      exception_handler(SQLCODE, 'NEW_MEMBER');
  END new_member;
END video_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

Parte B: Soluciones a la Práctica Adicional 4

4. Utilice los siguientes archivos de comandos ubicados en el directorio `E:\labs\PLPU\soln` para probar las rutinas:

   a. Agregue dos miembros con `sol_apb_04_a.sql`.

```
SET SERVEROUTPUT ON
EXECUTE video_pkg.new_member('Haas', 'James', 'Chestnut Street',
'Boston', '617-123-4567')
EXECUTE  video_pkg.new_member('Biri', 'Allan',  'Hiawatha Drive', 'New
York', '516-123-4567')

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.
```

   b. Agregue nuevos alquileres de vídeos con `sol_apb_04_b.sql`.

```
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(110, 98))

26-FEB-04
PL/SQL procedure successfully completed.

EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(109, 93))

26-FEB-04
PL/SQL procedure successfully completed.

EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(110, 98))

Movie reserved. Expected back on: 21-FEB-04
PL/SQL procedure successfully completed.

EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental('Biri', 97))

Warning! More than one member by this name.
112 Biri, Allan
108 Biri, Ben
PL/SQL procedure successfully completed.

EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(97, 97))

BEGIN DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(97, 97)); END;

*

ERROR at line 1:
ORA-20002: NEW_RENTAL has attempted to use a foreign key value that is
invalid
ORA-06512: at "ORA1.VIDEO_PKG", line 9
ORA-06512: at "ORA1.VIDEO_PKG", line 103
ORA-06512: at line 1
```

Parte B: Soluciones a la Práctica Adicional 4 (continuación)

c. Devuelva películas con el archivo de comandos `sol_apb_04_c.sql`.

```
EXECUTE video_pkg.return_movie(98, 1, 'AVAILABLE')

Put this movie on hold -- reserved by member #107
PL/SQL procedure successfully completed.

EXECUTE video_pkg.return_movie(95, 3, 'AVAILABLE')

PL/SQL procedure successfully completed.

EXECUTE video_pkg.return_movie(111, 1, 'RENTED')

BEGIN video_pkg.return_movie(111, 1, 'RENTED'); END;

*

ERROR at line 1:
ORA-20999: Unhandled error in RETURN_MOVIE. Please contact your
application administrator with the following information: ORA-01403: no
data found
ORA-06512: at "ORA1.VIDEO_PKG", line 12
ORA-06512: at "ORA1.VIDEO_PKG", line 69
ORA-06512: at line 1
```

Parte B: Soluciones a la Práctica Adicional 5

5. Las horas laborables para el videoclub son de 8:00 a.m. a 10:00 p.m. (de domingo a viernes) y de 8:00 a.m. a 12:00 a.m. (sábados). Para asegurarse de que las tablas sólo se pueden modificar durante estas horas, cree un procedimiento almacenado al que llamen los disparadores en las tablas.

   a. Cree un procedimiento almacenado denominado `TIME_CHECK` que compruebe la hora actual con respecto a las horas laborables. Si la hora actual no está dentro de las horas laborables, utilice el procedimiento `RAISE_APPLICATION_ERROR` para emitir el mensaje adecuado.

```
CREATE OR REPLACE PROCEDURE time_check IS
BEGIN
  IF ((TO_CHAR(SYSDATE,'D') BETWEEN 1 AND 6) AND
      (TO_DATE(TO_CHAR(SYSDATE, 'hh24:mi'), 'hh24:mi') NOT BETWEEN
       TO_DATE('08:00', 'hh24:mi') AND TO_DATE('22:00', 'hh24:mi')))
       OR ((TO_CHAR(SYSDATE, 'D') = 7)
       AND  (TO_DATE(TO_CHAR(SYSDATE, 'hh24:mi'), 'hh24:mi') NOT BETWEEN
       TO_DATE('08:00', 'hh24:mi') AND TO_DATE('24:00', 'hh24:mi'))) THEN
    RAISE_APPLICATION_ERROR(-20999,
       'Data changes restricted to office hours.');
  END IF;
END time_check;
/
SHOW ERRORS

Procedure created.

No errors.
```

   b. Cree un disparador en cada una de las cinco tablas. Arranque el disparador antes de insertar, actualizar y suprimir los datos de las tablas. Llame al procedimiento `TIME_CHECK` desde cada disparador.

```
CREATE OR REPLACE TRIGGER member_trig
  BEFORE INSERT OR UPDATE OR DELETE ON member
CALL time_check
/

CREATE OR REPLACE TRIGGER rental_trig
  BEFORE INSERT OR UPDATE OR DELETE ON rental
CALL time_check
/

CREATE OR REPLACE TRIGGER title_copy_trig
  BEFORE INSERT OR UPDATE OR DELETE ON title_copy
CALL time_check
/

CREATE OR REPLACE TRIGGER title_trig
  BEFORE INSERT OR UPDATE OR DELETE ON title
CALL time_check
/
```

```
CREATE OR REPLACE TRIGGER reservation_trig
  BEFORE INSERT OR UPDATE OR DELETE ON reservation
CALL time_check
/

Trigger created.

Trigger created.

Trigger created.

Trigger created.

Trigger created.
```

c. Pruebe los disparadores.

Nota: Para que el disparador falle, es posible que tenga que cambiar la hora para que esté fuera del rango de la hora actual de la clase. Por ejemplo, mientras realiza la prueba, puede interesarle que las horas válidas de vídeo del disparador sean desde las 6:00 p.m. a las 8:00 a.m.

```
-- First determine current timezone and time
SELECT SESSIONTIMEZONE,
       TO_CHAR(CURRENT_DATE, 'DD-MON-YYYY HH24:MI') CURR_DATE
FROM DUAL;
```

SESSIONTIMEZONE	CURR_DATE
+00:00	23-FEB-2004 11:39

```
-- Change your time zone usinge [+|-]HH:MI format such that the current
-- time returns a time between 6pm and 8am
ALTER SESSION SET TIME_ZONE='-07:00';

Session altered.

SELECT SESSIONTIMEZONE,
       TO_CHAR(CURRENT_DATE, 'DD-MON-YYYY HH24:MI') CURR_DATE
FROM DUAL;
```

SESSIONTIMEZONE	CURR_DATE
-07:00	23-FEB-2004 04:39

Parte B: Soluciones a la Práctica Adicional 5 (continuación)

```
-- Add a new member (for a sample test)
EXECUTE  video_pkg.new_member('Elias', 'Elliane',  'Vine Street',
'California', '789-123-4567')

BEGIN video_pkg.new_member('Elias', 'Elliane',  'Vine Street',
'California', '789-123-4567'); END;

*

ERROR at line 1:
ORA-20999: Unhandled error in NEW_MEMBER. Please contact your application
administrator with the following information: ORA-20999: Data changes
restricted to office hours.
ORA-06512: at "ORA1.TIME_CHECK", line 9
ORA-06512: at "ORA1.MEMBER_TRIG", line 1
ORA-04088: error during execution of trigger 'ORA1.MEMBER_TRIG'
ORA-06512: at "ORA1.VIDEO_PKG", line 12
ORA-06512: at "ORA1.VIDEO_PKG", line 171
ORA-06512: at line 1

-- Restore the original time zone for your session.
ALTER SESSION SET TIME_ZONE='-07:00';

Session altered.
```

Prácticas Adicionales: Descripciones de las Tablas y Datos

Parte A:

Las tablas y los datos utilizados en la parte A son los mismos que los del Apéndice B, “Descripciones de las Tablas y Datos”.

Parte B: Tablas Usadas

TNAME	TABTYPE	CLUSTERID
MEMBER	TABLE	
RENTAL	TABLE	
RESERVATION	TABLE	
TITLE	TABLE	
TITLE_COPY	TABLE	

Parte B: Tabla MEMBER

```
DESCRIBE member
```

Name	Null?	Type
MEMBER_ID	NOT NULL	NUMBER(10)
LAST_NAME	NOT NULL	VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
ADDRESS		VARCHAR2(100)
CITY		VARCHAR2(30)
PHONE		VARCHAR2(25)
JOIN_DATE	NOT NULL	DATE

```
SELECT * FROM member;
```

MEMBER_ID	LAST_NAME	FIRST_NAME	ADDRESS	CITY	PHONE	JOIN_DATE
101	Velasquez	Carmen	283 King Street	Seattle	587-99-6666	03-MAR-90
102	Ngao	LaDoris	5 Modrany	Bratislava	586-355-8882	08-MAR-90
103	Nagayama	Midori	68 Via Centrale	Sao Paolo	254-852-5764	17-JUN-91
104	Quick-To-See	Mark	6921 King Way	Lagos	63-559-777	07-APR-90
105	Ropeburn	Audry	86 Chu Street	Hong Kong	41-559-87	04-MAR-90
106	Urguhart	Molly	3035 Laurier Blvd.	Quebec	418-542-9988	18-JAN-91
107	Menchu	Roberta	Boulevard de Waterloo 41	Brussels	322-504-2228	14-MAY-90
108	Biri	Ben	398 High St.	Columbus	614-455-9863	07-APR-90
109	Catchpole	Antoinette	88 Alfred St.	Brisbane	616-399-1411	09-FEB-92

9 rows selected.

Parte B: Tabla `RENTAL`

```
DESCRIBE rental
```

Name	Null?	Type
BOOK_DATE	NOT NULL	DATE
COPY_ID	NOT NULL	NUMBER(10)
MEMBER_ID	NOT NULL	NUMBER(10)
TITLE_ID	NOT NULL	NUMBER(10)
ACT_RET_DATE		DATE
EXP_RET_DATE		DATE

```
SELECT * FROM rental;
```

BOOK_DATE	COPY_ID	MEMBER_ID	TITLE_ID	ACT_RET_D	EXP_RET_D
02-OCT-01	2	101	93		04-OCT-01
01-OCT-01	3	102	95		03-OCT-01
30-SEP-01	1	101	98		02-OCT-01
29-SEP-01	1	106	97	01-OCT-01	01-OCT-01
30-SEP-01	1	101	92	01-OCT-01	02-OCT-01

Parte B: Tabla `RESERVATION`

```
DESCRIBE reservation
```

Name	Null?	Type
RES_DATE	NOT NULL	DATE
MEMBER_ID	NOT NULL	NUMBER(10)
TITLE_ID	NOT NULL	NUMBER(10)

```
SELECT * FROM reservation;
```

RES_DATE	MEMBER_ID	TITLE_ID
02-OCT-01	101	93
01-OCT-01	106	102

Parte B: Tabla `TITLE`

```
DESCRIBE title
```

Name	Null?	Type
TITLE_ID	NOT NULL	NUMBER(10)
TITLE	NOT NULL	VARCHAR2(60)
DESCRIPTION	NOT NULL	VARCHAR2(400)
RATING		VARCHAR2(4)
CATEGORY		VARCHAR2(20)
RELEASE_DATE		DATE

```
SELECT * FROM title;
```

TITLE_ID	TITLE	DESCRIPTION	RATI	CATEGORY	RELEASE_D
92	Willie and Christmas Too	All of Willie's friends made a Christmas list for Santa, but Willie has yet to create his own wish list.	G	CHILD	05-OCT-95
93	Alien Again	Another installment of science fiction history. Can the heroine save the planet from the alien life form?	R	SCIFI	19-MAY-95
94	The Glob	A meteor crashes near a small American town and unleashes carivorous goo in this classic.	NR	SCIFI	12-AUG-95
95	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-95
96	Miracles on Ice	A six-year-old has doubts about Santa Claus. But she discovers that miracles really do exist.	PG	DRAMA	12-SEP-95
97	Soda Gang	After discovering a cached of drugs, a young couple find themselves pitted against a vicious gang.	NR	ACTION	01-JUN-95
98	Interstellar Wars	Futuristic interstellar action movie. Can the rebels save the humans from the evil Empire?	PG	SCIFI	07-JUL-77

7 rows selected.

Parte B: Tabla `TITLE_COPY`

```
DESCRIBE title_copy
```

Name	Null?	Type
COPY_ID	NOT NULL	NUMBER(10)
TITLE_ID	NOT NULL	NUMBER(10)
STATUS	NOT NULL	VARCHAR2(15)

```
SELECT * FROM title_copy;
```

COPY_ID	TITLE_ID	STATUS
1	92	AVAILABLE
1	93	AVAILABLE
2	93	RENTED
1	94	AVAILABLE
1	95	AVAILABLE
2	95	AVAILABLE
3	95	RENTED
1	96	AVAILABLE
1	97	AVAILABLE
1	98	RENTED
2	98	AVAILABLE

11 rows selected.