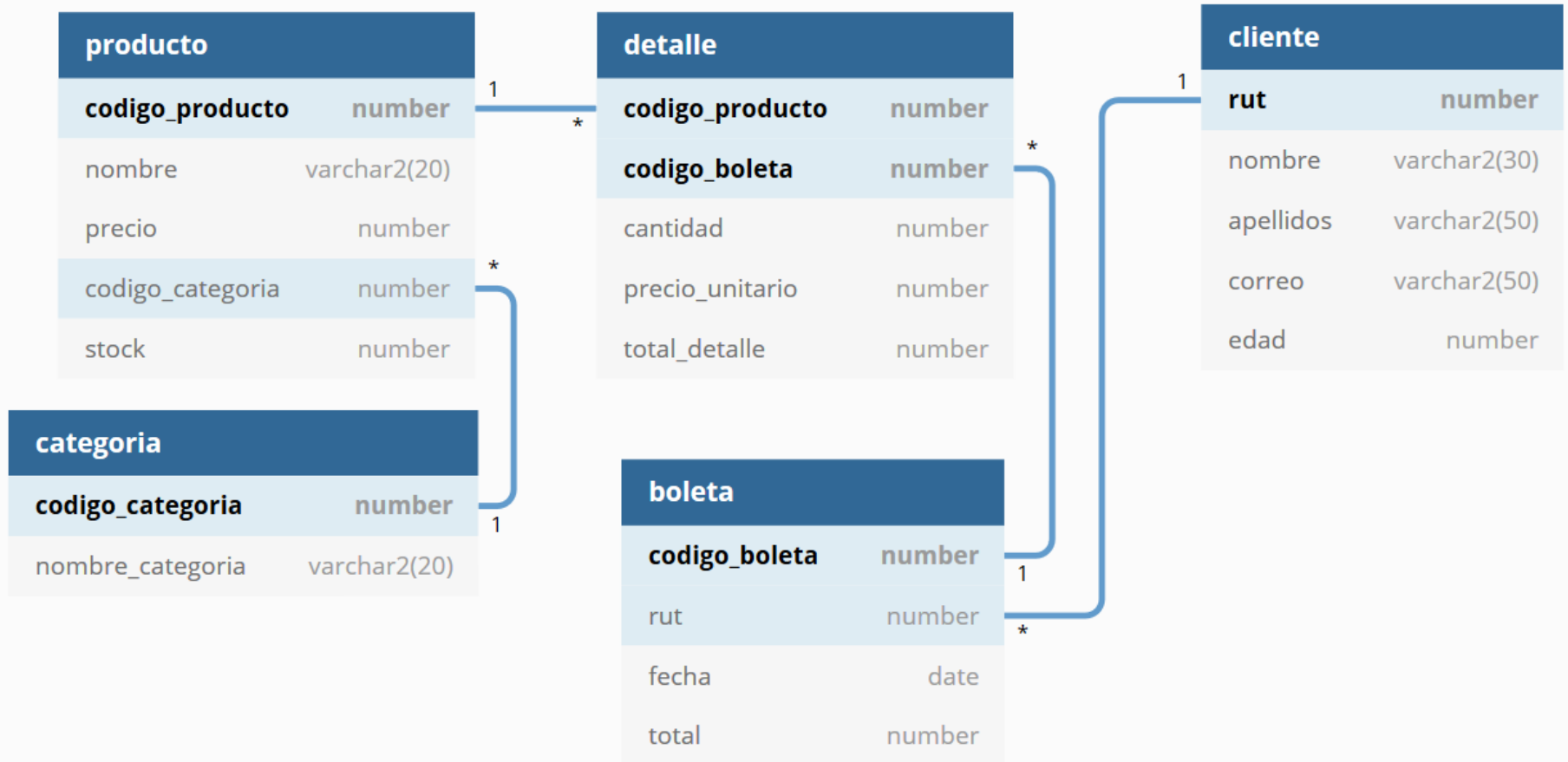


# Procedimientos

# Programa de hoy

- Procedimientos (Programación por bloques)
- Transacciones
- Concepto de bloqueo de tablas
- Concepto de Commit
- Concepto de Rollback
- Control de errores (Excepciones)

# Modelo de trabajo



# Programación por bloques

Hasta el momento, solo hemos utilizado el lenguaje de manipulación y definición de datos.

# Programación por bloques

Hasta el momento, solo hemos utilizado el lenguaje de manipulación y definición de datos. Ahora explotaremos otro aspecto del lenguaje que es conocido como PL/SQL.

# Programación por bloques

Hasta el momento, solo hemos utilizado el lenguaje de manipulación y definición de datos. Ahora explotaremos otro aspecto del lenguaje que es conocido como PL/SQL.

Este aspecto sirve para que los programadores puedan construir bloques de código, los cuales pueden utilizarse como procedimientos o funciones.

# Programación por bloques

EL PL/SQL incluye nuevas características:

- Manejo de variables.
- Estructuras modulares.
- Estructuras de control de flujo.
- Control de excepciones.

# Programación por bloques

Dentro de la programación por bloques tenemos los siguientes tipos:

## Anonymous

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

## Procedure

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

## Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```



# Bloques anónimos

La clase anterior, vimos bloques anónimos.

# Bloques anónimos

La clase anterior, vimos bloques anónimos.

Los bloques anónimos son segmentos de código que pueden ejecutarse para que realicen un cálculo o resuelvan un problema.

# Bloques anónimos

La clase anterior, vimos bloques anónimos.

Los bloques anónimos son segmentos de código que pueden ejecutarse para que realicen un cálculo o resuelvan un problema.

Estos bloques no se guardan en la base de datos.

# Bloques anónimos

Los bloques anónimos poseen la siguiente sintaxis:

```
DECLARE  
    DECLARACION_DE_VARIABLES  
BEGIN  
    [OPERACIONES] [CONSULTAS] [ETC]  
END;
```

# Procedimientos

Los procedimientos son segmentos de código que pueden ejecutarse para que realicen un cálculo y/o resuelvan un problema.

# Procedimientos

Los procedimientos son segmentos de código que pueden ejecutarse para que realicen un cálculo y/o resuelvan un problema.

Este tipo de bloque sí se almacena en la base de datos.

# Procedimientos

Los procedimientos son segmentos de código que pueden ejecutarse para que realicen un cálculo y/o resuelvan un problema.

Este tipo de bloque sí se almacena en la base de datos.

Una vez creados, pueden llamarse para reutilizarse.

# Procedimiento (Sintaxis)

## Creación del Procedimiento

```
CREATE OR REPLACE PROCEDURE NOMBRE_PROCEDIMIENTO (  
    DATOS DE ENTRADA DEL PROCEDIMIENTO)  
IS  
    [VARIABLES]  
BEGIN  
    [OPERACIONES] [CONSULTAS] [LLAMADA PROCEDIMIENTO] [LLAMADA FUNCIONES]  
END;
```

## Borrado del Procedimiento

```
DROP PROCEDURE NOMBRE_PROCEDIMIENTO;
```



# Procedimientos

Los procedimientos no retornan valores como las funciones, pero, pueden pasar parámetros por referencia.

# Procedimientos

Los procedimientos no retornan valores como las funciones, pero, pueden pasar parámetros por referencia.

El procedimiento trabaja con 3 tipos de parámetros:

# Procedimientos

Los procedimientos no retornan valores como las funciones, pero, pueden pasar parámetros por referencia.

El procedimiento trabaja con 3 tipos de parámetros:

- IN: Quiere decir que el dato será utilizado o modificado únicamente al interior del procedimiento.
- OUT: Quiere decir que el dato será utilizado o modificado dentro del procedimiento y podrá salir de este para ser utilizado. (Entiéndase valor por referencia).
- IN OUT: Es una mezcla de los 2 anteriores.

# Procedimiento (Parámetro IN)

Para el ejemplo, crearemos un procedimiento que realiza el ingreso de un nuevo cliente.

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE (  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS  
    --DECLARACIÓN DE VARIABLES (SI ES QUE SE NECESITAN)  
BEGIN  
    INSERT INTO CLIENTE VALUES (RUTC, NOMBREC, APELLIDOSC, CORREOC, EDADC);  
    DBMS_OUTPUT.PUT_LINE('CLIENTE INGRESADO CON ÉXITO');  
END;
```

# Procedimiento (Parámetro IN)

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE(  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS  
    --DECLARACIÓN DE VARIABLES (SI ES QUE SE NECESITAN)  
BEGIN  
    INSERT INTO CLIENTE VALUES (RUTC, NOMBREC, APELLIDOSC, CORREOC, EDADC);  
    DBMS_OUTPUT.PUT_LINE('CLIENTE INGRESADO CON ÉXITO');  
END;
```

Este procedimiento recibe 5 parámetros de entrada (IN).

Una vez creado el procedimiento, podemos verlo almacenado en nuestra base de datos.

# Procedimiento (Parámetro IN)

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE(  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS  
    --DECLARACIÓN DE VARIABLES (SI ES QUE SE NECESITAN)  
BEGIN  
    INSERT INTO CLIENTE VALUES (RUTC, NOMBREC, APELLIDOSC, CORREOC, EDADC);  
    DBMS_OUTPUT.PUT_LINE('CLIENTE INGRESADO CON ÉXITO');  
END;
```

Este procedimiento recibe 5 parámetros de entrada (IN).

Una vez creado el procedimiento, podemos verlo almacenado en nuestra base de datos.

Para verlo, vamos al panel izquierdo del SQL Developer y buscamos la pestaña que diga “Procedimientos”.

# Procedimiento (Parámetro IN)

Para ejecutar este procedimiento, utilizamos la instrucción CALL y agregamos los parámetros de entrada.

# Procedimiento (Parámetro IN)

Para ejecutar este procedimiento, utilizamos la instrucción CALL y agregamos los parámetros de entrada.

```
CALL INGRESO_CLIENTE(171155240, 'GONZALO', 'PAREDES', 'GONZALO@UCM.CL', 29);
```



# Procedimiento (Parámetro IN)

Para ejecutar este procedimiento, utilizamos la instrucción CALL y agregamos los parámetros de entrada.

```
CALL INGRESO_CLIENTE(171155240, 'GONZALO', 'PAREDES', 'GONZALO@UCM.CL', 29);
```

Una vez ejecutado obtenemos el mensaje:

```
CLIENTE INGRESADO CON ÉXITO
```

```
Call terminado.
```

# Procedimiento (Parámetro IN)

Para ejecutar este procedimiento, utilizamos la instrucción CALL y agregamos los parámetros de entrada.

```
CALL INGRESO_CLIENTE(171155240, 'GONZALO', 'PAREDES', 'GONZALO@UCM.CL', 29);
```

Una vez ejecutado obtenemos el mensaje:

```
CLIENTE INGRESADO CON ÉXITO
```

```
Call terminado.
```



verificamos

```
SELECT * FROM CLIENTE  
WHERE RUT = 171155240;
```

# Procedimiento (Parámetro IN)

Para ejecutar este procedimiento, utilizamos la instrucción CALL y agregamos los parámetros de entrada.

```
CALL INGRESO_CLIENTE(171155240, 'GONZALO', 'PAREDES', 'GONZALO@UCM.CL', 29);
```

Una vez ejecutado obtenemos el mensaje:

CLIENTE INGRESADO CON ÉXITO

Call terminado.

verificamos

```
SELECT * FROM CLIENTE  
WHERE RUT = 171155240;
```

RUT	NOMBRE	APELLIDOS	CORREO	EDAD
171155240	GONZALO	PAREDES	GONZALO@UCM.CL	29

# Procedimiento (Parámetro IN)

Como se vio anteriormente, es necesario definir el tipo de dato que poseen los parámetros que ingresan a un procedimiento.

# Procedimiento (Parámetro IN)

Como se vio anteriormente, es necesario definir el tipo de dato que poseen los parámetros que ingresan a un procedimiento.

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE (  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS
```

# Procedimiento (Parámetro IN)

Como se vio anteriormente, es necesario definir el tipo de dato que poseen los parámetros que ingresan a un procedimiento.

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE (  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS
```

Es posible utilizar tipos de datos ya existentes de atributos pertenecientes a una tabla.

# Procedimiento (Parámetro IN)

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE(  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS
```

```
create table cliente(  
    rut number not null,  
    nombre varchar2(30),  
    apellidos varchar2(50),  
    correo varchar2(50),  
    edad number  
);
```

Sabemos (para este caso) que los datos de entrada para este procedimiento, están asociados a los tipos de datos de la tabla cliente. No es coincidencia que sean idénticos.

# Procedimiento (Parámetro IN)

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE(  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS
```

```
create table cliente(  
    rut number not null,  
    nombre varchar2(30),  
    apellidos varchar2(50),  
    correo varchar2(50),  
    edad number  
);
```

Sabemos (para este caso) que los datos de entrada para este procedimiento, están asociados a los tipos de datos de la tabla cliente. No es coincidencia que sean idénticos.

Es posible asociar los tipos de datos de los parámetros del procedimiento directamente con los tipos de datos de la tabla cliente.



# Procedimiento (Parámetro IN)

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE(  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS
```

```
create table cliente(  
    rut number not null,  
    nombre varchar2(30),  
    apellidos varchar2(50),  
    correo varchar2(50),  
    edad number  
);
```



```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE(  
    RUTC IN CLIENTE.RUT%TYPE,  
    NOMBREC IN CLIENTE.NOMBRE%TYPE,  
    APELLIDOSC IN CLIENTE.APELLIDOS%TYPE,  
    CORREOC IN CLIENTE.CORREO%TYPE,  
    EDADC IN CLIENTE.EDAD%TYPE  
)  
IS
```

Estas 2 formas de definir los tipos de datos son equivalentes.

# Procedimiento (Parámetro IN)

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE(  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS
```

```
create table cliente(  
    rut number not null,  
    nombre varchar2(30),  
    apellidos varchar2(50),  
    correo varchar2(50),  
    edad number  
);
```



```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE(  
    RUTC IN CLIENTE.RUT%TYPE,  
    NOMBREC IN CLIENTE.NOMBRE%TYPE,  
    APELLIDOSC IN CLIENTE.APELLIDOS%TYPE,  
    CORREOC IN CLIENTE.CORREO%TYPE,  
    EDADC IN CLIENTE.EDAD%TYPE  
)  
IS
```

Estas 2 formas de definir los tipos de datos son equivalentes.

La segunda forma tiene la ventaja que si el tipo de dato se cambia sobre la tabla, automáticamente se cambiará en el procedimiento.

# Procedimiento (Parámetro IN)

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE(  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS
```

```
create table cliente(  
    rut number not null,  
    nombre varchar2(30),  
    apellidos varchar2(50),  
    correo varchar2(50),  
    edad number  
);
```



```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE(  
    RUTC IN CLIENTE.RUT%TYPE,  
    NOMBREC IN CLIENTE.NOMBRE%TYPE,  
    APELLIDOSC IN CLIENTE.APELLIDOS%TYPE,  
    CORREOC IN CLIENTE.CORREO%TYPE,  
    EDADC IN CLIENTE.EDAD%TYPE  
)  
IS
```

Estas 2 formas de definir los tipos de datos son equivalentes.

La segunda forma tiene la ventaja que si el tipo de dato se cambia sobre la tabla, automáticamente se cambiará en el procedimiento.

# Procedimiento (Parámetro OUT)

Para el ejemplo, realizaremos un procedimiento que suma dos números y retorna la suma.

# Procedimiento (Parámetro OUT)

Para el ejemplo, realizaremos un procedimiento que suma dos números y retorna la suma.

Este procedimiento tendrá dos parámetros de entrada y un parámetro de salida. Este último contendrá la suma de ambos números.

# Procedimiento (Parámetro OUT)

Para el ejemplo, realizaremos un procedimiento que suma dos números y retorna la suma.

Este procedimiento tendrá dos parámetros de entrada y un parámetro de salida. Este último contendrá la suma de ambos números.

```
CREATE OR REPLACE PROCEDURE SUMA(  
    NUMERO1 IN NUMBER,  
    NUMERO2 IN NUMBER,  
    RESULTADO OUT NUMBER  
)  
IS  
BEGIN  
    RESULTADO := NUMERO1 + NUMERO2;  
END;
```

# Procedimiento (Parámetro OUT)

```
CREATE OR REPLACE PROCEDURE SUMA(  
    NUMERO1 IN NUMBER,  
    NUMERO2 IN NUMBER,  
    RESULTADO OUT NUMBER  
)  
IS  
BEGIN  
    RESULTADO := NUMERO1 + NUMERO2;  
END;
```

Los procedimientos que poseen parámetros OUT ó IN OUT, **deben (en SQL Developer)** ejecutarse utilizando bloques anónimos.

# Procedimiento (Parámetro OUT)

```
CREATE OR REPLACE PROCEDURE SUMA (  
    NUMERO1 IN NUMBER,  
    NUMERO2 IN NUMBER,  
    RESULTADO OUT NUMBER  
)  
IS  
BEGIN  
    RESULTADO := NUMERO1 + NUMERO2;  
END;
```

Los procedimientos que poseen parámetros OUT ó IN OUT, **deben (en SQL Developer)** ejecutarse utilizando bloques anónimos.

La ejecución de este procedimiento quedaría de la siguiente forma:

```
DECLARE  
    RESULTADO NUMBER;  
BEGIN  
    SUMA(1, 2, RESULTADO);  
    DBMS_OUTPUT.PUT_LINE('EL RESULTADO ES: ' || RESULTADO);  
END;
```



# Procedimiento (Parámetro OUT)

La ejecución del bloque anónimo anterior nos entrega el siguiente resultado:

```
EL RESULTADO ES: 3
```

```
Procedimiento PL/SQL terminado correctamente.
```

# Procedimiento (Parámetro OUT)

La ejecución del bloque anónimo anterior nos entrega el siguiente resultado:

```
EL RESULTADO ES: 3
```

```
Procedimiento PL/SQL terminado correctamente.
```

Es posible ingresar los parámetros de entrada del procedimiento por medio de una interfaz.

# Procedimiento (Parámetro OUT)

Para activar la interfaz, es necesario modificar nuestro procedimiento.

```
DECLARE
    RESULTADO NUMBER;
    NUMERO1 NUMBER;
    NUMERO2 NUMBER;
BEGIN
    SUMA (&NUMERO1, &NUMERO2, RESULTADO) ;
    DBMS_OUTPUT.PUT_LINE ('EL RESULTADO ES: ' || RESULTADO) ;
END;
```

# Procedimiento (Parámetro OUT)

Para activar la interfaz, es necesario modificar nuestro procedimiento.

```
DECLARE
    RESULTADO NUMBER;
    NUMERO1 NUMBER; ←
    NUMERO2 NUMBER; ←
BEGIN
    SUMA (&NUMERO1, &NUMERO2, RESULTADO) ;
    DBMS_OUTPUT.PUT_LINE ('EL RESULTADO ES: ' || RESULTADO) ;
END;
```

Primero, es necesario declarar las variables que almacenarán lo que ingresemos por medio de la interfaz.

# Procedimiento (Parámetro OUT)

Para activar la interfaz, es necesario modificar nuestro procedimiento.

```
DECLARE
    RESULTADO NUMBER;
    NUMERO1 NUMBER;
    NUMERO2 NUMBER;
BEGIN
    SUMA (&NUMERO1, &NUMERO2, RESULTADO) ;
    DBMS_OUTPUT.PUT_LINE ('EL RESULTADO ES: ' || RESULTADO);
END;
```

Primero, es necesario declarar las variables que almacenarán lo que ingresemos por medio de la interfaz.

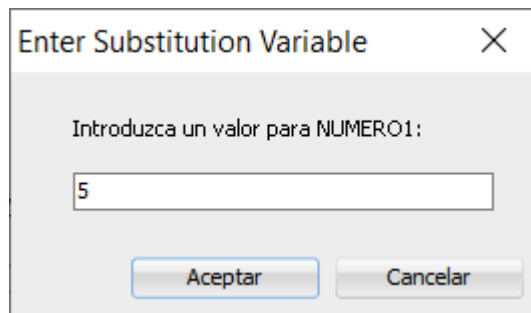
Luego, vinculamos esas variables a los parámetros de entrada del procedimiento anteponiendo un & antes de la variable.

# Procedimiento (Parámetro OUT)

Al ejecutar nuevamente nuestro bloque anónimo, aparecerán secuencialmente unas ventanas solicitando los datos de entrada para las variables NUMERO1 y NUMERO2.

# Procedimiento (Parámetro OUT)

Al ejecutar nuevamente nuestro bloque anónimo, aparecerán secuencialmente unas ventanas solicitando los datos de entrada para las variables NUMERO1 y NUMERO2.

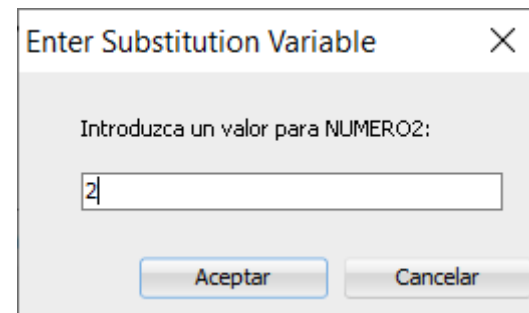


Enter Substitution Variable

Introduzca un valor para NUMERO1:

5

Aceptar Cancelar



Enter Substitution Variable

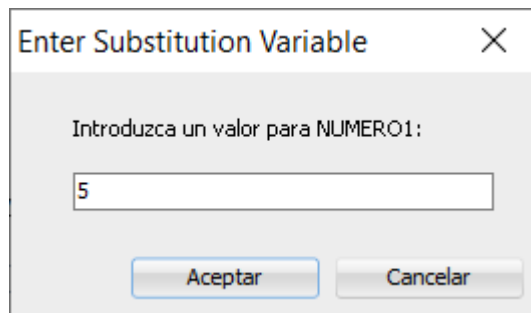
Introduzca un valor para NUMERO2:

2

Aceptar Cancelar

# Procedimiento (Parámetro OUT)

Al ejecutar nuevamente nuestro bloque anónimo, aparecerán secuencialmente unas ventanas solicitando los datos de entrada para las variables NUMERO1 y NUMERO2.

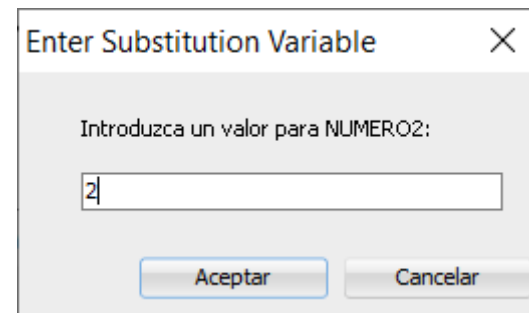


Enter Substitution Variable

Introduzca un valor para NUMERO1:

5

Aceptar Cancelar



Enter Substitution Variable

Introduzca un valor para NUMERO2:

2

Aceptar Cancelar

Obteniendo...

```
EL RESULTADO ES: 7
```

```
Procedimiento PL/SQL terminado correctamente.
```

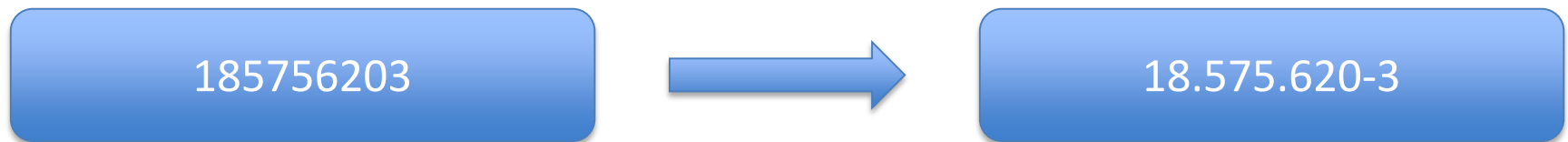


# Procedimiento (Parámetro IN OUT)

Para el ejemplo, realizaremos un procedimiento que dado un RUT en formato numérico, entregará el RUT con puntos y el guion.

# Procedimiento (Parámetro IN OUT)

Para el ejemplo, realizaremos un procedimiento que dado un RUT en formato numérico, entregará el RUT con puntos y el guion.



# Procedimiento (Parámetro IN OUT)

Para el ejemplo, realizaremos un procedimiento que dado un RUT en formato numérico, entregará el RUT con puntos y el guion.

185756203



18.575.620-3

```
CREATE OR REPLACE PROCEDURE RUT_FORMAT (  
    RUT IN OUT VARCHAR2  
)  
IS  
    RUT_AUX VARCHAR2 (12);  
BEGIN  
    RUT_AUX := CONCAT (CONCAT (SUBSTR (RUT, 0, LENGTH (RUT) - 1), '-'), SUBSTR (RUT, LENGTH (RUT), 1));  
    RUT_AUX := CONCAT (CONCAT (SUBSTR (RUT, 0, LENGTH (RUT) - 4), '.'), SUBSTR (RUT_AUX, LENGTH (RUT) - 3));  
    RUT_AUX := CONCAT (CONCAT (SUBSTR (RUT, 0, LENGTH (RUT) - 7), '.'), SUBSTR (RUT_AUX, LENGTH (RUT) - 6));  
    RUT := RUT_AUX;  
END;
```

# Procedimiento (Parámetro IN OUT)

La ejecución del procedimiento anterior se realiza también utilizando un bloque anónimo.

# Procedimiento (Parámetro IN OUT)

La ejecución del procedimiento anterior se realiza también utilizando un bloque anónimo.

```
DECLARE
    RUT VARCHAR2(12);
BEGIN
    RUT := 185756203;
    RUT_FORMAT(RUT);
    DBMS_OUTPUT.PUT_LINE(RUT);
END;
```

# Procedimiento (Parámetro IN OUT)

La ejecución del procedimiento anterior se realiza también utilizando un bloque anónimo.

```
DECLARE
    RUT VARCHAR2(12);
BEGIN
    RUT := 185756203;
    RUT_FORMAT(RUT);
    DBMS_OUTPUT.PUT_LINE(RUT);
END;
```

El resultado de la ejecución del procedimiento es:

# Procedimiento (Parámetro IN OUT)

La ejecución del procedimiento anterior se realiza también utilizando un bloque anónimo.

```
DECLARE
    RUT VARCHAR2(12);
BEGIN
    RUT := 185756203;
    RUT_FORMAT(RUT);
    DBMS_OUTPUT.PUT_LINE(RUT);
END;
```

El resultado de la ejecución del procedimiento es:

```
18.575.620-3
```

```
Procedimiento PL/SQL terminado correctamente.
```

# Funcionamiento de transacciones

Consideremos que tenemos un sistema de compra de productos.



# Funcionamiento de transacciones

Consideremos que tenemos un sistema de compra de productos.

Este modelo, está almacenado en una base de datos llamada laboratorio.

# Funcionamiento de transacciones

Consideremos que tenemos un sistema de compra de productos.

Este modelo, está almacenado en una base de datos llamada laboratorio.



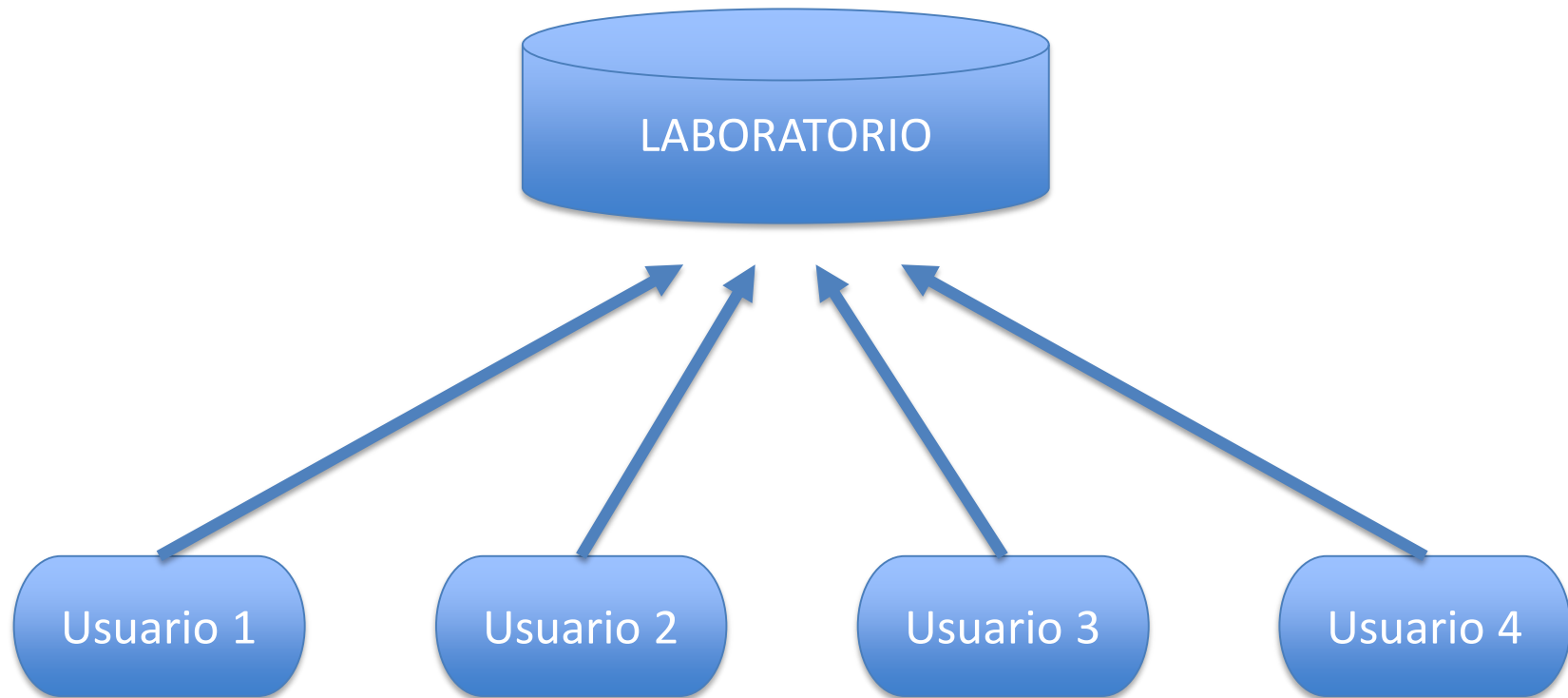
# Funcionamiento de transacciones

Supongamos que 4 personas se conectan a nuestro sistema para comprar un producto.

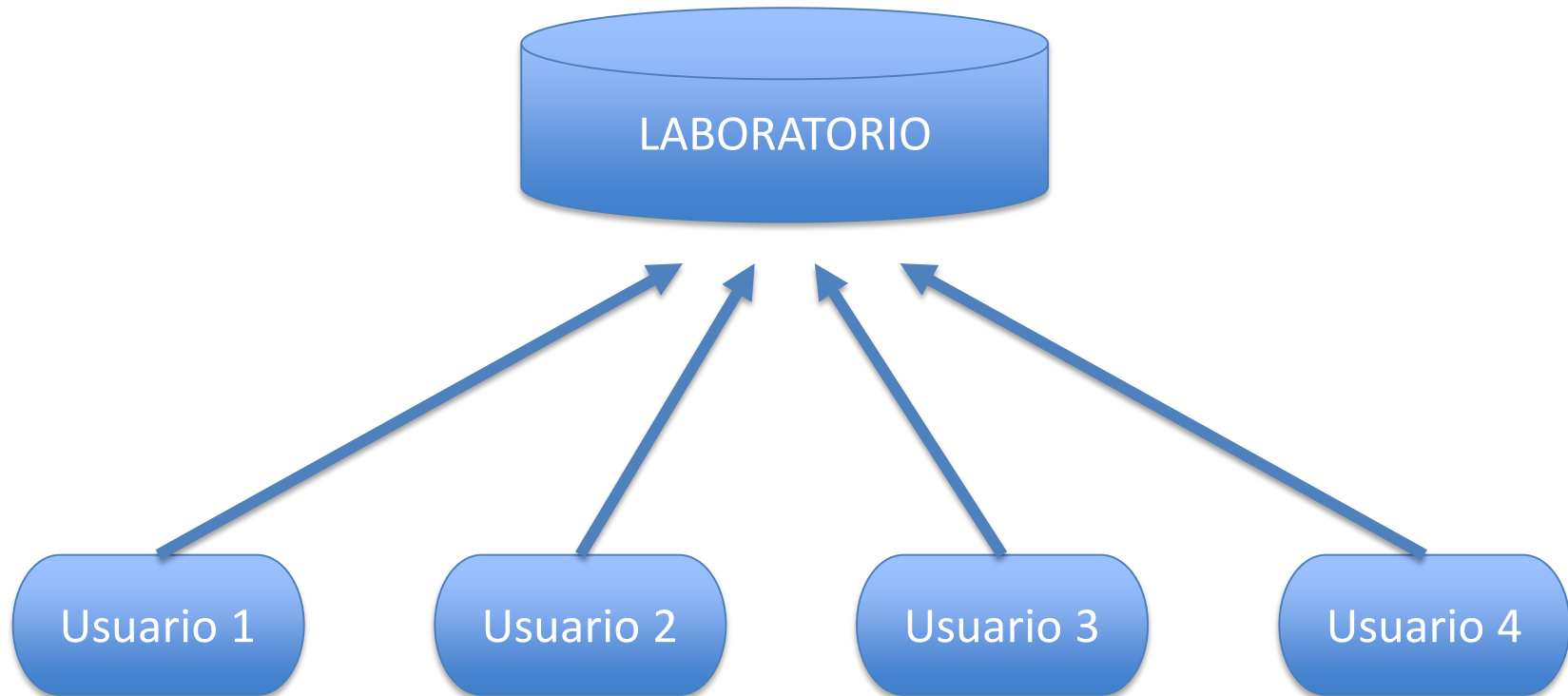


# Funcionamiento de transacciones

Supongamos que 4 personas se conectan a nuestro sistema para comprar un producto.

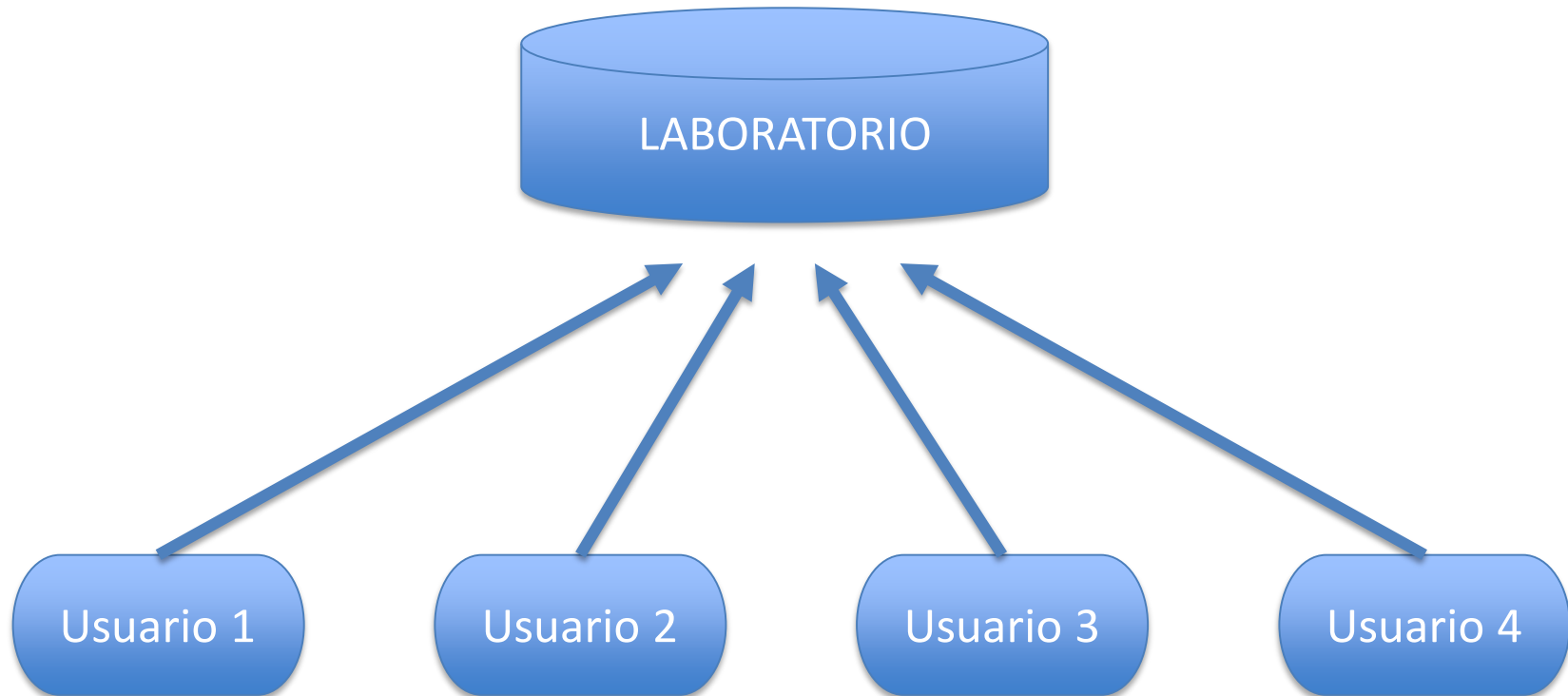


# Funcionamiento de transacciones



Todos estos usuarios, realizarán compras de productos. Los productos disponibles son:

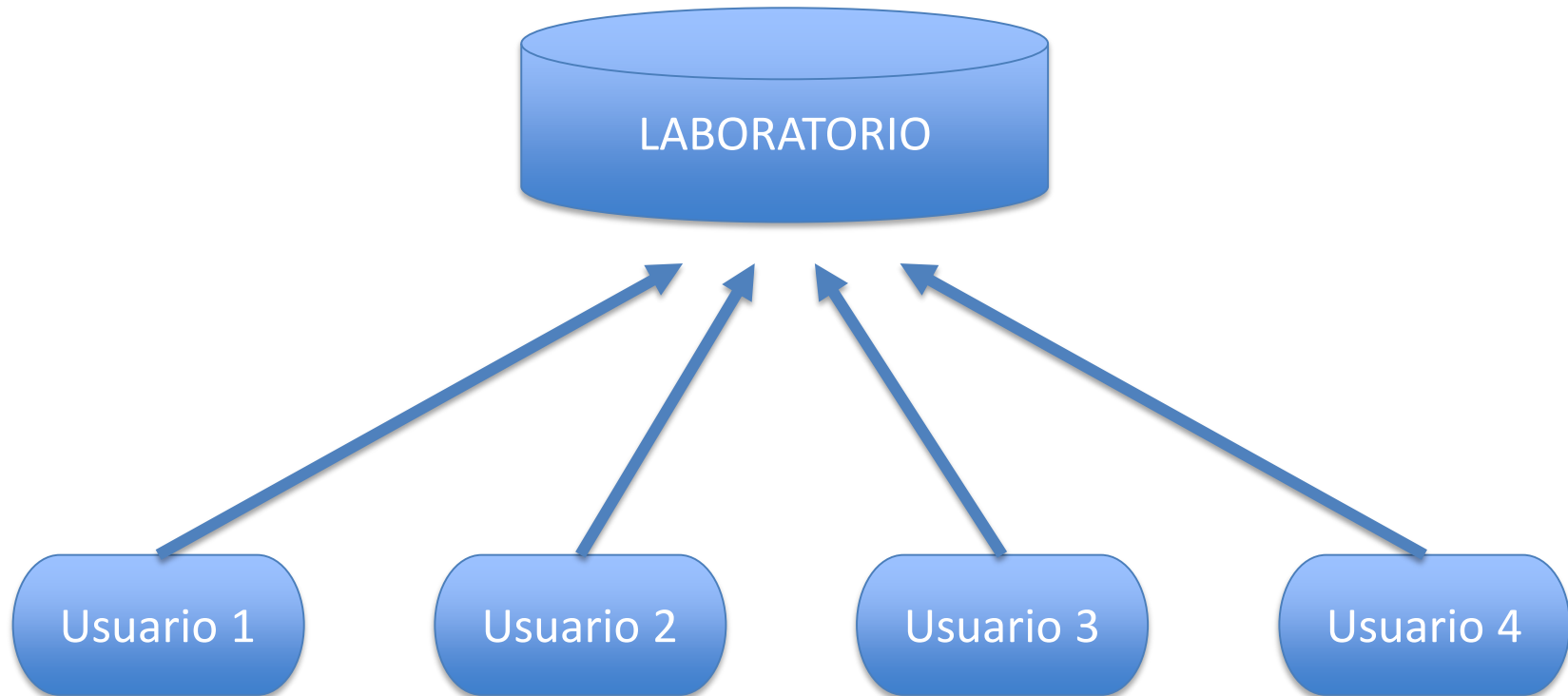
# Funcionamiento de transacciones



PRODUCTO_AUX			
CODIGO_PRODUCTO	NOMBRE	PRECIO	STOCK
1	NOTEBOOK HP	\$280.990	1
2	CELULAR IPHONE 6S	\$560.000	6
3	CELULAR LG	\$450.000	7
4	MOUSE BLUETOOTH	\$28.990	9

Anotar en pizarra

# Funcionamiento de transacciones



El usuario 1, quiere comprar 2 celulares IPHONE 6S.

El usuario 2, quiere comprar 3 celulares IPHONE 6S y 1 NOTEBOOK HP.

El usuario 3, quiere comprar 2 celulares IPHONE 6S y 5 celulares LG.

El usuario 4, quiere comprar 1 NOTEBOOK HP y 1 MOUSE BLUETOOTH.

Vamos a suponer que una compra significa reducir el stock del producto

# Funcionamiento de transacciones

Para el ejemplo, crearemos una tabla llamada PRODUCTO\_AUX.



# Funcionamiento de transacciones

Para el ejemplo, crearemos una tabla llamada PRODUCTO\_AUX.

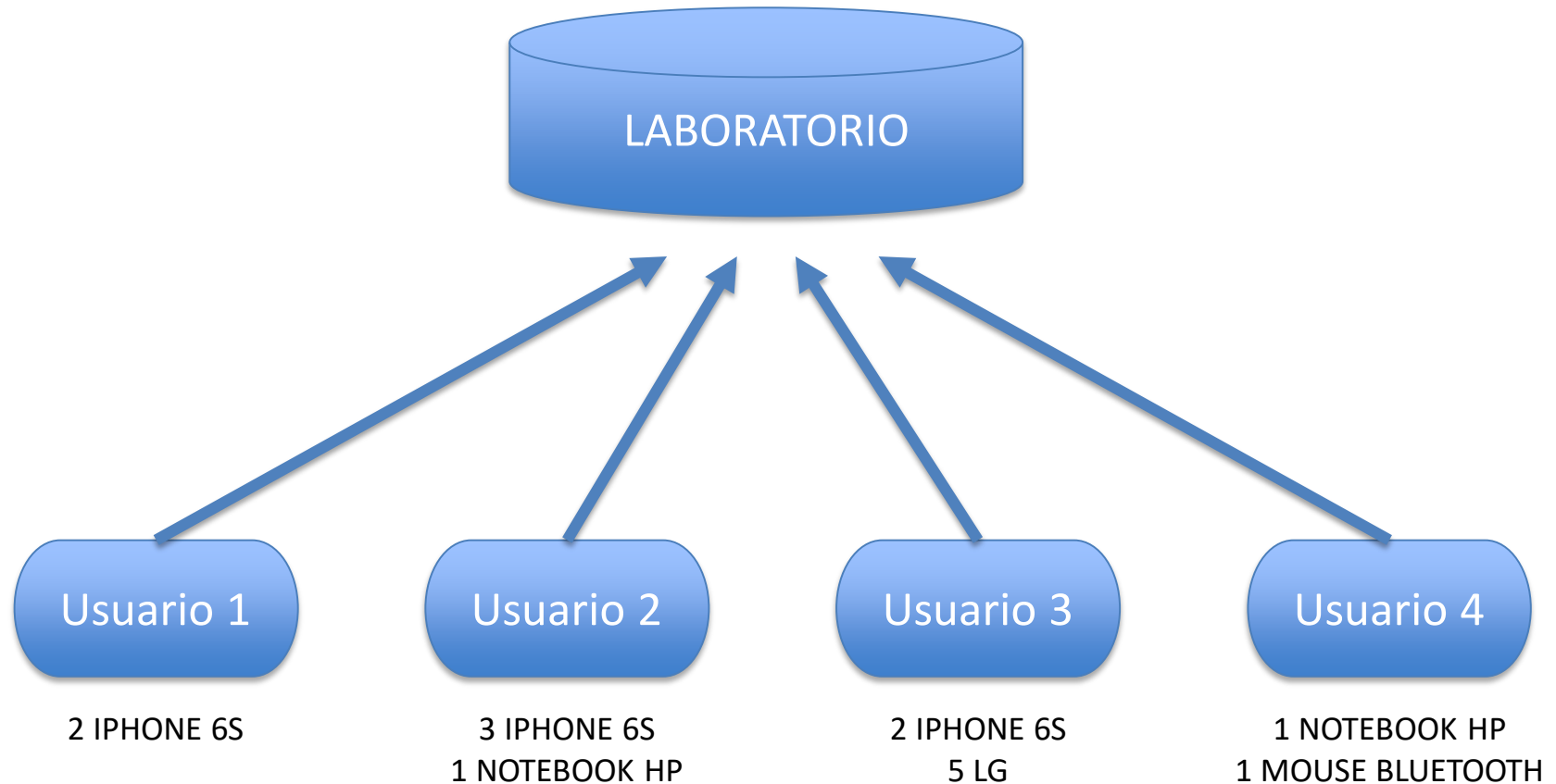
```
CREATE TABLE PRODUCTO_AUX (  
    CODIGO_PRODUCTO NUMBER,  
    NOMBRE VARCHAR2 (30) ,  
    PRECIO NUMBER,  
    STOCK NUMBER  
);
```

# Funcionamiento de transacciones

Y además, un procedimiento que emula la compra del producto.

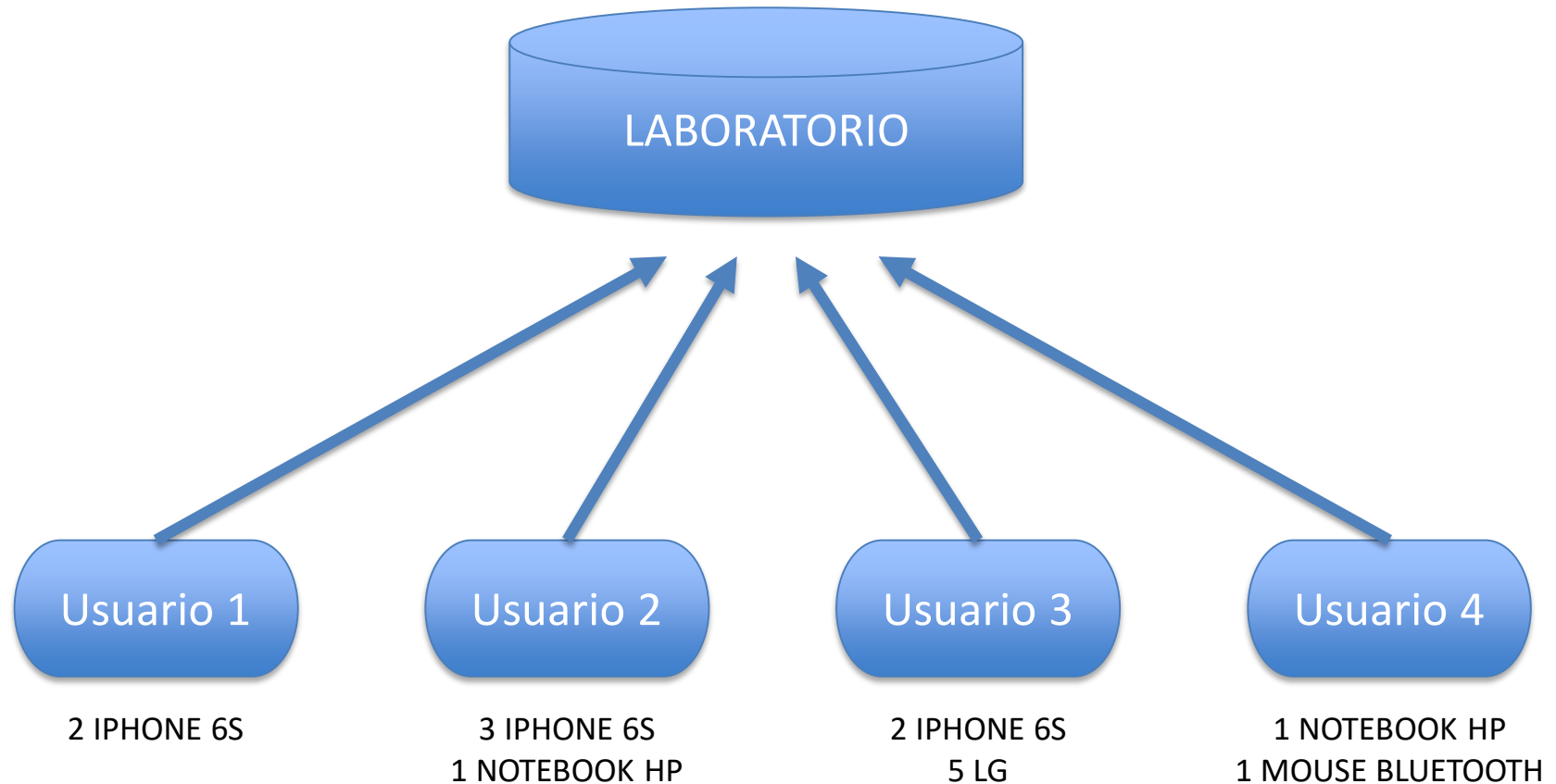
```
CREATE OR REPLACE PROCEDURE COMPRAR (  
    CODIGOP IN PRODUCTO_AUX.CODIGO_PRODUCTO%TYPE,  
    CANTIDAD NUMBER  
)  
IS  
BEGIN  
    UPDATE PRODUCTO_AUX  
    SET STOCK = (STOCK - CANTIDAD)  
    WHERE CODIGO_PRODUCTO = CODIGOP;  
    DBMS_OUTPUT.PUT_LINE('PRODUCTOS COMPRADOS');  
END;
```

# Funcionamiento de transacciones



En este sistema, se realiza una simulación de 4 transacciones. Donde puede ocurrir que los 4 usuarios realicen la compra exactamente al mismo tiempo. (Control de concurrencia).

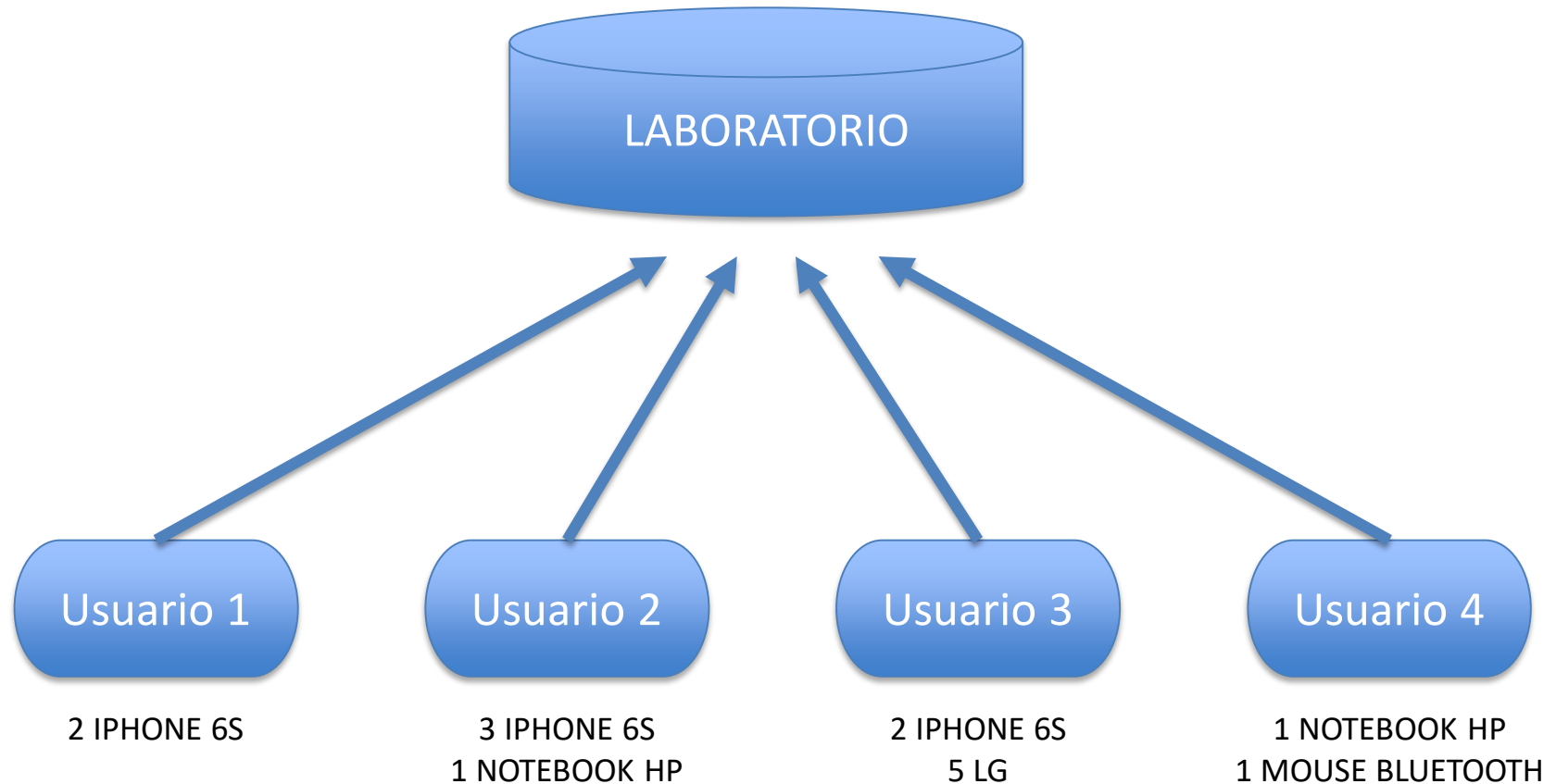
# Funcionamiento de transacciones



En este sistema, se realiza una simulación de 4 transacciones. Donde puede ocurrir que los 4 usuarios realicen la compra exactamente al mismo tiempo. (Control de concurrencia).

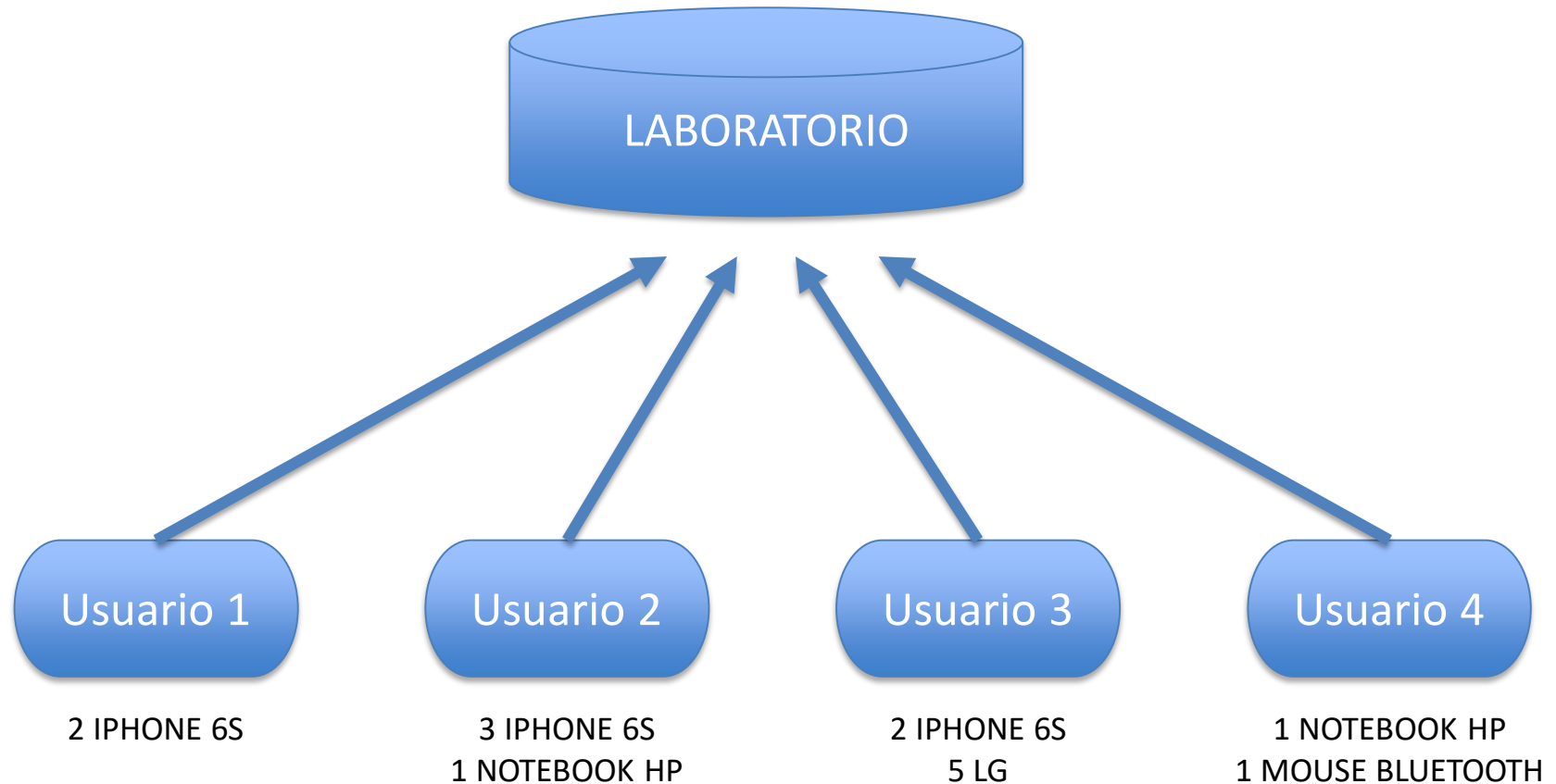
Oracle ya se encarga de la concurrencia realizando bloqueos sobre las transacciones. El problema, es que al ser automático, puede provocar bloqueos múltiples por cada transacción, lo que finalmente se traduce en un bloqueo total.

# Funcionamiento de transacciones



Para solucionar el problema, es posible realizar bloqueos explícitos sobre una tabla cuando una transacción se procesa en la base de datos.

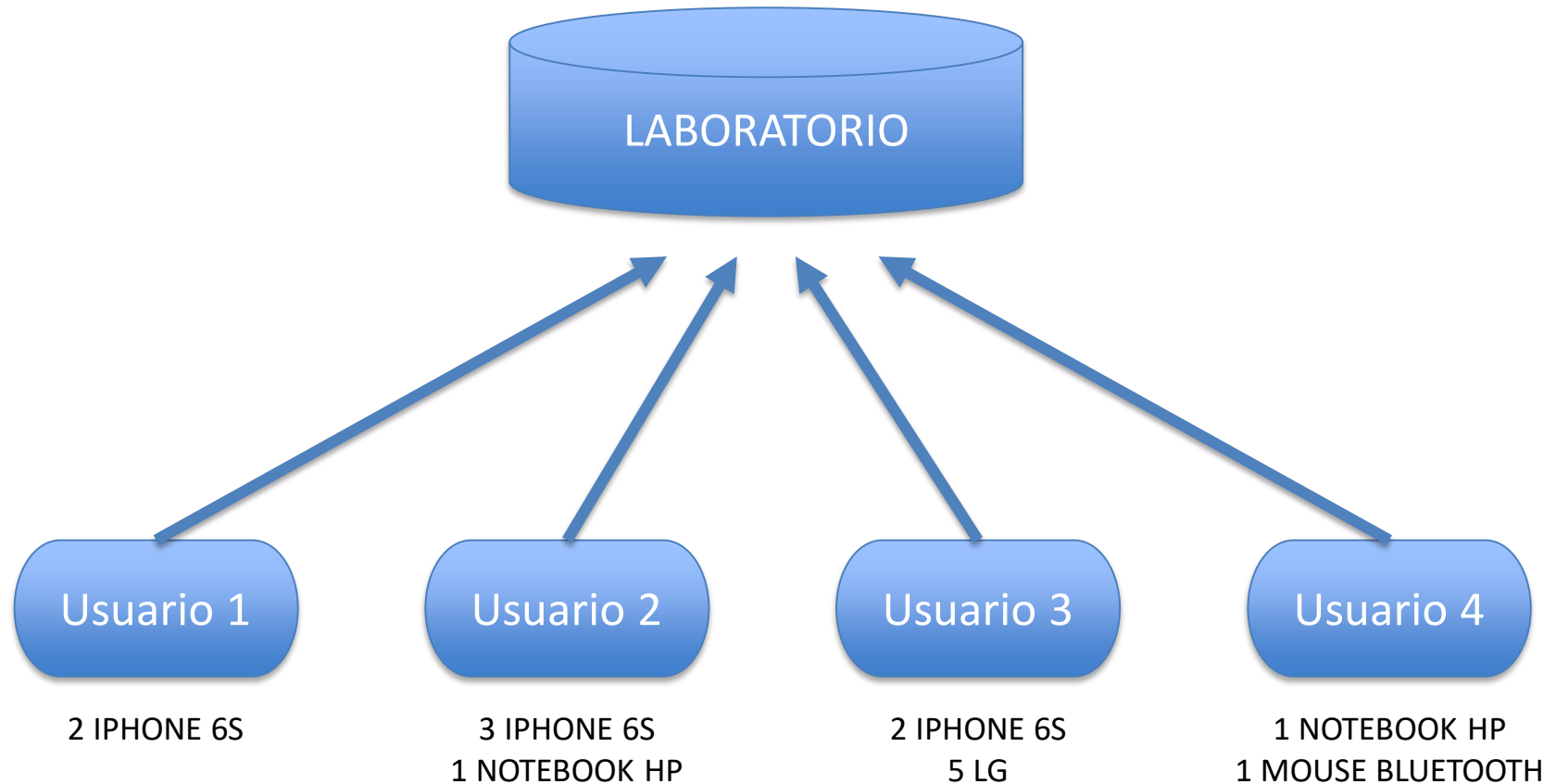
# Funcionamiento de transacciones



Para solucionar el problema, es posible realizar bloqueos explícitos sobre una tabla cuando una transacción se procesa en la base de datos.

El bloqueo se traduce en utilizar una sentencia llamada LOCK TABLE que da acceso a la tabla a un usuario mientras el resto debe esperar hasta que finalice la transacción.

# Funcionamiento de transacciones



Para solucionar el problema, es posible realizar bloqueos explícitos sobre una tabla cuando una transacción se procesa en la base de datos.

El bloqueo se traduce en utilizar una sentencia llamada LOCK TABLE que da acceso a la tabla a un usuario mientras el resto debe esperar hasta que finalice la transacción.

Una vez finalizada la transacción, la tabla se desbloquea y queda libre para otro usuario.

# Concepto de bloqueo de tablas

Para trabajar la concurrencia de las transacciones, es necesario trabajar con bloqueos. Tal como el sistema operativo realiza bloqueos para la ejecución de los programas, es posible realizar bloqueos explícitos para trabajar sobre una tabla.



# Concepto de bloqueo de tablas

Para trabajar la concurrencia de las transacciones, es necesario trabajar con bloqueos. Tal como el sistema operativo realiza bloqueos para la ejecución de los programas, es posible realizar bloqueos explícitos para trabajar sobre una tabla.

Oracle provee la instrucción `LOCK TABLE`, que permite realizar un bloqueo sobre una tabla para obtener acceso exclusivo o compartido.

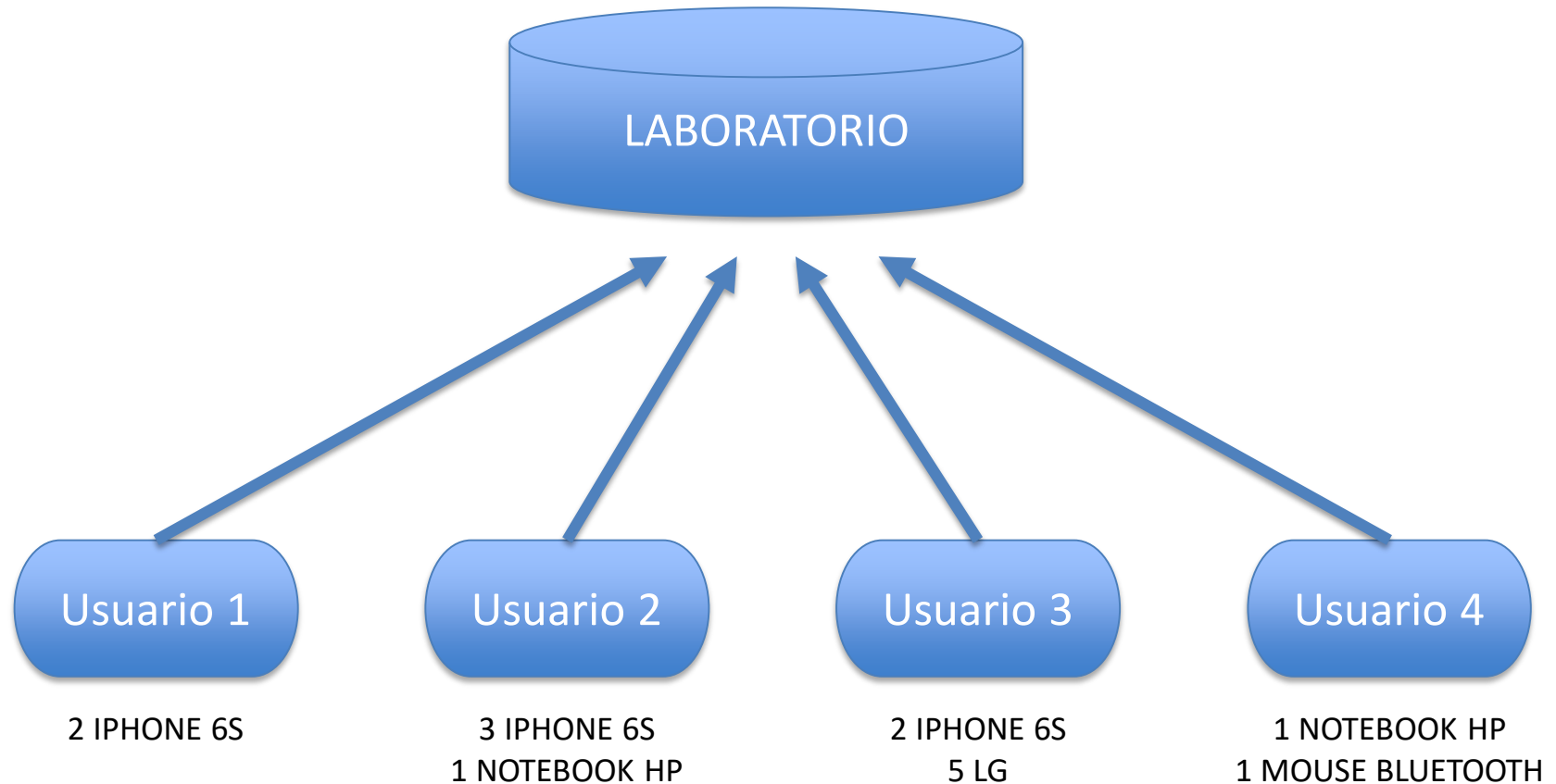
# Concepto de bloqueo de tablas

Para trabajar la concurrencia de las transacciones, es necesario trabajar con bloqueos. Tal como el sistema operativo realiza bloqueos para la ejecución de los programas, es posible realizar bloqueos explícitos para trabajar sobre una tabla.

Oracle provee la instrucción `LOCK TABLE`, que permite realizar un bloqueo sobre una tabla para obtener acceso exclusivo o compartido.

El bloqueo de una tabla termina cuando la transacción en curso finaliza.

# Funcionamiento de transacciones



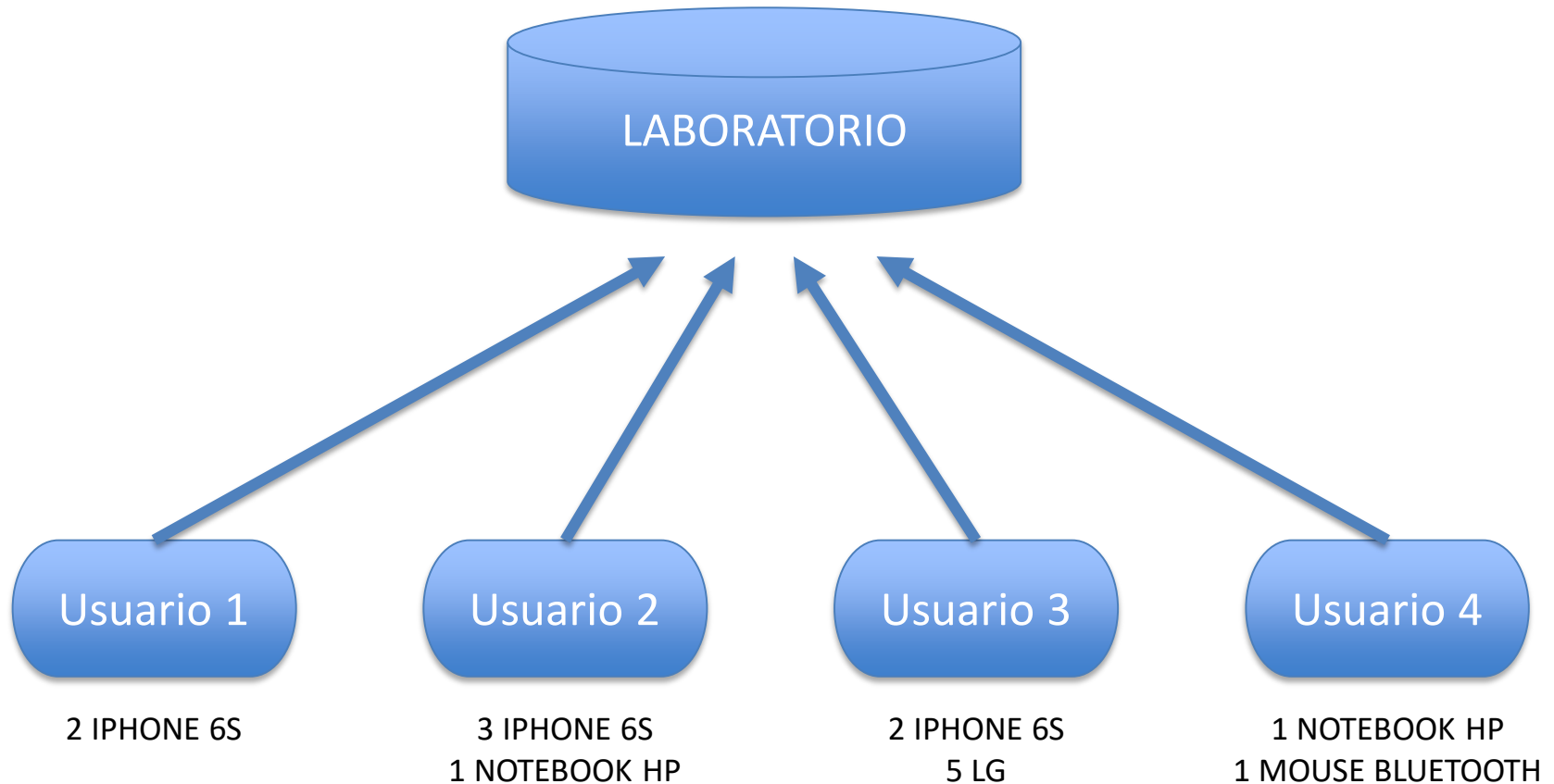
Utilizando el bloqueo de las tablas, podemos modificar nuestro procedimiento y agregar la instrucción de bloqueo **LOCK TABLE PRODUCTO\_AUX IN ROW EXCLUSIVE MODE.**

# Funcionamiento de transacciones

Esta modificación permitirá que la base de datos realice el control de concurrencia cuando existan múltiples solicitudes sobre la tabla PRODUCTO\_AUX.

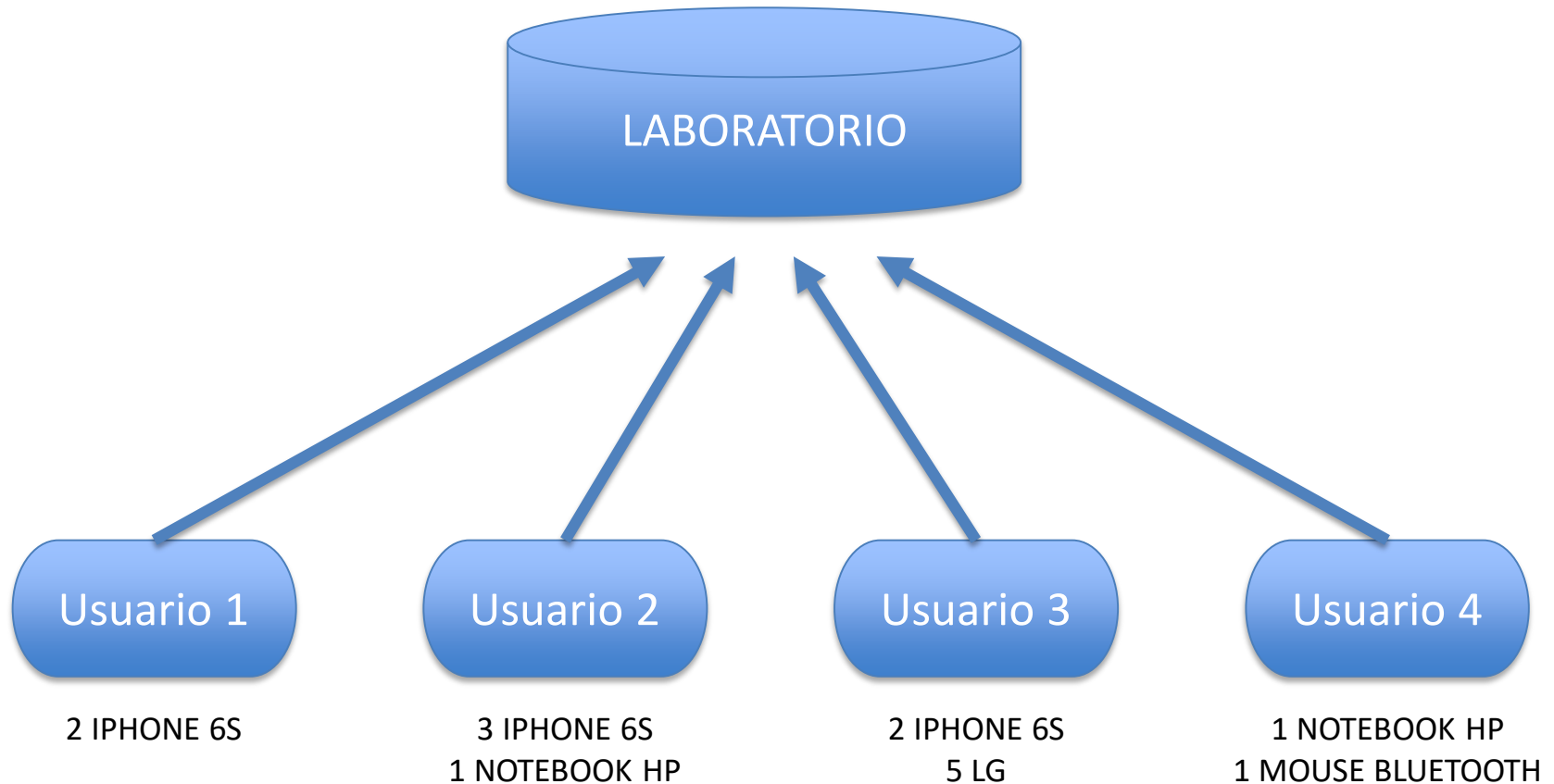
```
CREATE OR REPLACE PROCEDURE COMPRAR (  
    CODIGOP IN PRODUCTO_AUX.CODIGO_PRODUCTO%TYPE,  
    CANTIDAD NUMBER  
)  
IS  
BEGIN  
    LOCK TABLE PRODUCTO_AUX IN ROW EXCLUSIVE MODE;  
    UPDATE PRODUCTO_AUX  
    SET STOCK = (STOCK - CANTIDAD)  
    WHERE CODIGO_PRODUCTO = CODIGOP;  
    DBMS_OUTPUT.PUT_LINE('PRODUCTOS COMPRADOS');  
END;
```

# Funcionamiento de transacciones



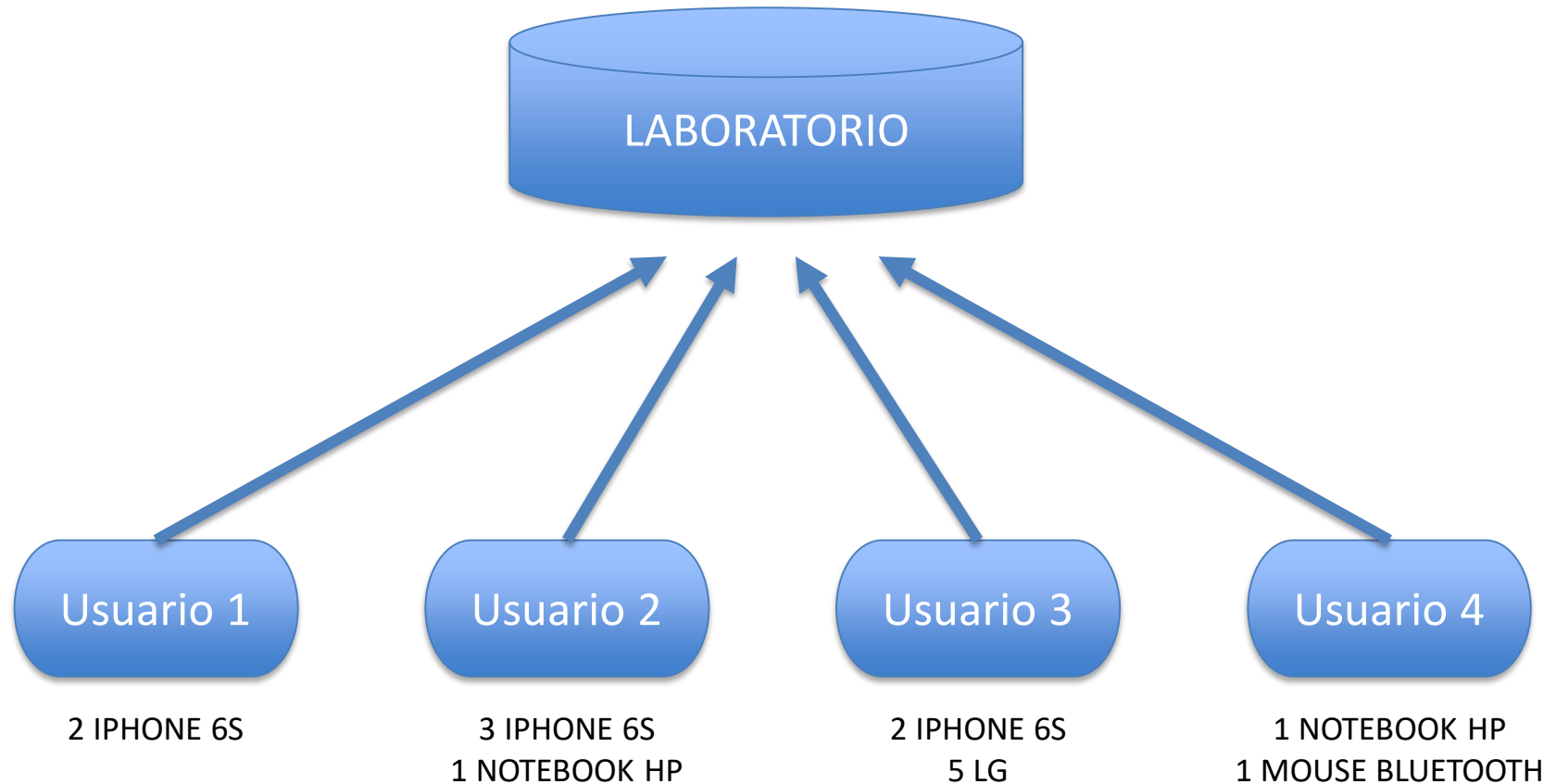
Con esas modificaciones, hemos solucionado el primer problema que correspondía a la concurrencia.

# Funcionamiento de transacciones



Con esas modificaciones, hemos solucionado el primer problema que correspondía a la concurrencia. Pero ahora, hay otro problema... (Mostrar DEMO)

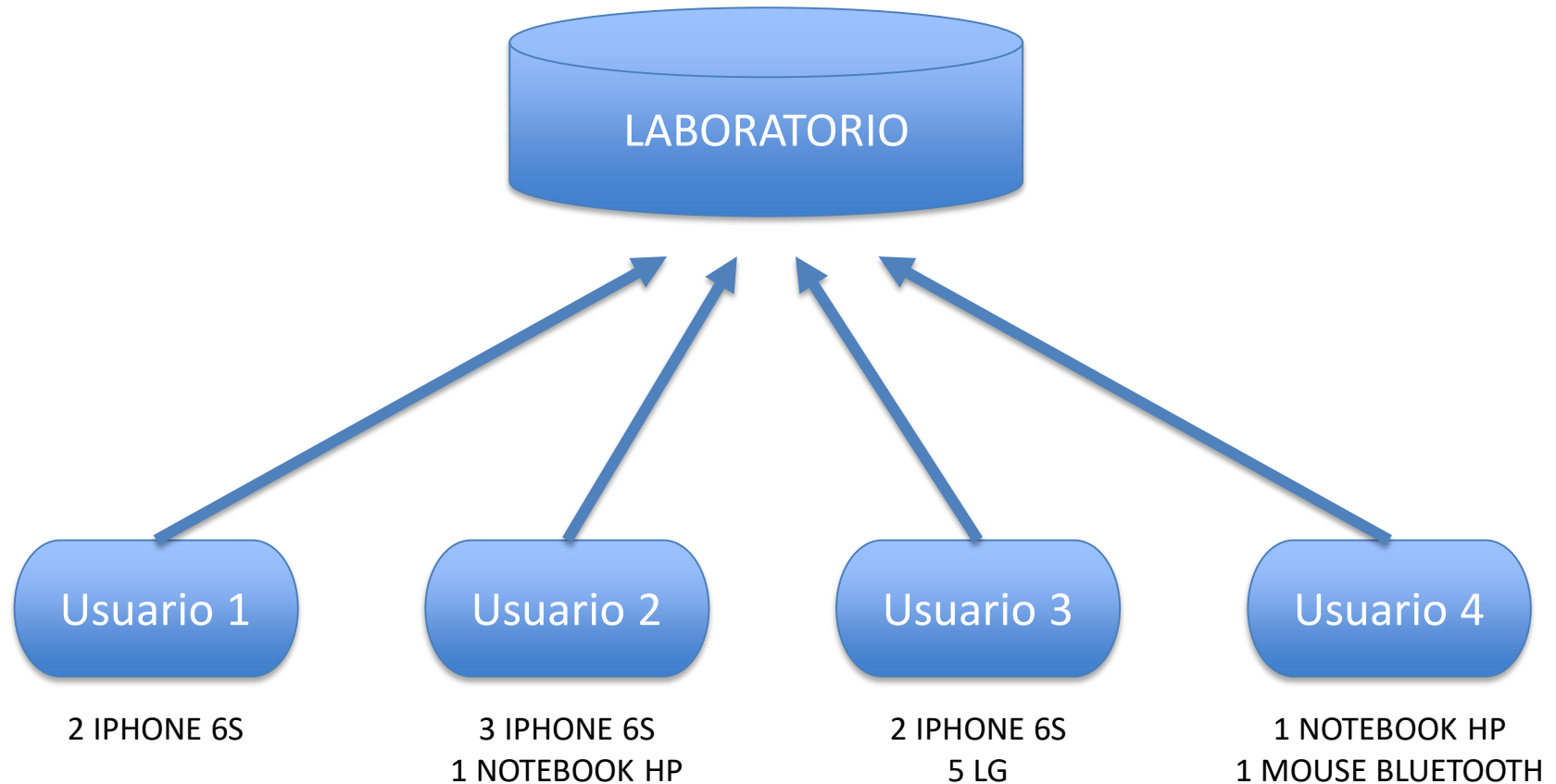
# Funcionamiento de transacciones



Con esas modificaciones, hemos solucionado el primer problema que correspondía a la concurrencia. Pero ahora, hay otro problema...

Hemos solucionado el acceso para cada usuario bloqueando la tabla, pero ahora es necesario liberar la tabla del usuario actual para que otro usuario pueda ejecutar su propia transacción.

# Funcionamiento de transacciones



Con esas modificaciones, hemos solucionado el primer problema que correspondía a la concurrencia. Pero ahora, hay otro problema...

Hemos solucionado el acceso para cada usuario bloqueando la tabla, pero ahora es necesario liberar la tabla del usuario actual para que otro usuario pueda ejecutar su propia transacción.

Aquí aparece un nuevo concepto llamado COMMIT y ROLLBACK.



# Concepto de Commit

Cada vez que un usuario realiza una transacción, está “pidiendo prestada” la base de datos para trabajar sobre ella.

# Concepto de Commit

Cada vez que un usuario realiza una transacción, está “pidiendo prestada” la base de datos para trabajar sobre ella.

Una vez que termina de trabajar, llámese esto realizar un insert, un update, crear un procedimiento, etc. Debe indicarle a la base de datos que guarde los cambios realizados.

# Concepto de Commit

Cada vez que un usuario realiza una transacción, está “pidiendo prestada” la base de datos para trabajar sobre ella.

Una vez que termina de trabajar, llámese esto realizar un insert, un update, crear un procedimiento, etc. Debe indicarle a la base de datos que guarde los cambios realizados.

Esta indicación se realiza utilizando la instrucción COMMIT.

# Concepto de Commit

El propósito de la instrucción COMMIT es hacer permanentes los cambios realizados en una transacción.

# Concepto de Commit

El propósito de la instrucción COMMIT es hacer permanentes los cambios realizados en una transacción.

Del ejemplo anterior, cuando el primer usuario realiza la llamada al procedimiento para comprar un producto, la tabla se bloquea. Pero en ningún momento se notifica a la base de datos que ya terminó de usar la tabla.

# Concepto de Commit

Es por esto, que el segundo usuario al intentar acceder a comprar otro producto, queda en espera.

# Concepto de Commit

Es por esto, que el segundo usuario al intentar acceder a comprar otro producto, queda en espera.

La solución es que cada vez que se utilice un LOCK TABLE y la transacción finalice de forma exitosa, debe utilizarse la instrucción COMMIT para liberar la tabla y hacer permanentes los cambios sobre la base de datos.

# Funcionamiento de transacciones

Del ejemplo anterior, modificamos el procedimiento y agregamos la instrucción COMMIT.

```
CREATE OR REPLACE PROCEDURE COMPRAR (  
    CODIGOP IN PRODUCTO_AUX.CODIGO_PRODUCTO%TYPE,  
    CANTIDAD NUMBER  
)  
IS  
BEGIN  
    LOCK TABLE PRODUCTO_AUX IN ROW EXCLUSIVE MODE;  
    UPDATE PRODUCTO_AUX  
    SET STOCK = (STOCK - CANTIDAD)  
    WHERE CODIGO_PRODUCTO = CODIGOP;  
    DBMS_OUTPUT.PUT_LINE('PRODUCTOS COMPRADOS');  
    COMMIT;  
END;
```



# Funcionamiento de transacciones

Cuando sabemos que la transacción será exitosa, utilizamos la instrucción COMMIT para guardar los cambios.

# Funcionamiento de transacciones

Cuando sabemos que la transacción será exitosa, utilizamos la instrucción COMMIT para guardar los cambios.

¿Pero qué ocurre cuando durante la transacción surge algún error y debe anularse la operación?

# Control de errores

Para analizar los errores, considere el siguiente procedimiento:

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE (  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS  
    --DECLARACIÓN DE VARIABLES (SI ES QUE SE NECESITAN)  
BEGIN  
    LOCK TABLE CLIENTE IN ROW EXCLUSIVE MODE;  
    INSERT INTO CLIENTE VALUES (RUTC, NOMBREC, APELLIDOSC, CORREOC, EDADC);  
    DBMS_OUTPUT.PUT_LINE('CLIENTE INGRESADO CON ÉXITO');  
    COMMIT;  
END;
```

# Control de errores

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE (  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS  
    --DECLARACIÓN DE VARIABLES (SI ES QUE SE NECESITAN)  
BEGIN  
    LOCK TABLE CLIENTE IN ROW EXCLUSIVE MODE;  
    INSERT INTO CLIENTE VALUES (RUTC, NOMBREC, APELLIDOSC, CORREOC, EDADC);  
    DBMS_OUTPUT.PUT_LINE('CLIENTE INGRESADO CON ÉXITO');  
    COMMIT;  
END;
```

Este procedimiento corresponde al visto en el ejemplo de parámetros de entrada. En ese momento ingresamos al cliente con rut: 171155240.

```
CALL INGRESO_CLIENTE(171155240, 'GONZALO', 'PAREDES', 'GONZALO@UCM.CL', 29);
```

# Control de errores

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE (  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS  
    --DECLARACIÓN DE VARIABLES (SI ES QUE SE NECESITAN)  
BEGIN  
    LOCK TABLE CLIENTE IN ROW EXCLUSIVE MODE;  
    INSERT INTO CLIENTE VALUES (RUTC, NOMBREC, APELLIDOSC, CORREOC, EDADC);  
    DBMS_OUTPUT.PUT_LINE('CLIENTE INGRESADO CON ÉXITO');  
    COMMIT;  
END;
```

¿Qué sucede si volvemos a ejecutar nuevamente la sentencia que está abajo?

```
CALL INGRESO_CLIENTE(171155240, 'GONZALO', 'PAREDES', 'GONZALO@UCM.CL', 29);
```

# Control de errores

```
CALL INGRESO_CLIENTE(171155240,'GONZALO','PAREDES','GONZALO@UCM.CL',29)
Informe de error -
ORA-00001: unique constraint (CLASE.PK_CLIENTE) violated
ORA-06512: at "CLASE.INGRESO_CLIENTE", line 12
00001. 00000 - "unique constraint (%s.%s) violated"
*Cause:      An UPDATE or INSERT statement attempted to insert a duplicate key.
              For Trusted Oracle configured in DBMS MAC mode, you may see
              this message if a duplicate entry exists at a different level.
*Action:     Either remove the unique restriction or do not insert the key.
```

Llamar nuevamente al procedimiento INGRESO\_CLIENTE intentando insertar el mismo usuario provoca un error debido a que se repite la clave primaria RUT.

# Control de errores

```
CALL INGRESO_CLIENTE(171155240,'GONZALO','PAREDES','GONZALO@UCM.CL',29)
Informe de error -
ORA-00001: unique constraint (CLASE.PK_CLIENTE) violated
ORA-06512: at "CLASE.INGRESO_CLIENTE", line 12
00001. 00000 - "unique constraint (%s.%s) violated"
*Cause:      An UPDATE or INSERT statement attempted to insert a duplicate key.
              For Trusted Oracle configured in DBMS MAC mode, you may see
              this message if a duplicate entry exists at a different level.
*Action:     Either remove the unique restriction or do not insert the key.
```

Llamar nuevamente al procedimiento INGRESO\_CLIENTE intentando insertar el mismo usuario provoca un error debido a que se repite la clave primaria RUT. Este error en Oracle está catalogado bajo el código de error: **ORA-00001.**

# Control de errores

```
CALL INGRESO_CLIENTE(171155240,'GONZALO','PAREDES','GONZALO@UCM.CL',29)
Informe de error -
ORA-00001: unique constraint (CLASE.PK_CLIENTE) violated
ORA-06512: at "CLASE.INGRESO_CLIENTE", line 12
00001. 00000 - "unique constraint (%s.%s) violated"
*Cause:      An UPDATE or INSERT statement attempted to insert a duplicate key.
              For Trusted Oracle configured in DBMS MAC mode, you may see
              this message if a duplicate entry exists at a different level.
*Action:     Either remove the unique restriction or do not insert the key.
```

Llamar nuevamente al procedimiento INGRESO\_CLIENTE intentando insertar el mismo usuario provoca un error debido a que se repite la clave primaria RUT. Este error en Oracle está catalogado bajo el código de error: **ORA-00001.**

En Oracle cada error está representado por un código. Estos códigos poseen el formato de ORA-XXXXX y cada código representa un error distinto.



# Control de errores

```
CALL INGRESO_CLIENTE(171155240,'GONZALO','PAREDES','GONZALO@UCM.CL',29)
Informe de error -
ORA-00001: unique constraint (CLASE.PK_CLIENTE) violated
ORA-06512: at "CLASE.INGRESO_CLIENTE", line 12
00001. 00000 - "unique constraint (%s.%s) violated"
*Cause:      An UPDATE or INSERT statement attempted to insert a duplicate key.
              For Trusted Oracle configured in DBMS MAC mode, you may see
              this message if a duplicate entry exists at a different level.
*Action:     Either remove the unique restriction or do not insert the key.
```

Este es un ejemplo de error que no fue capturado por el procedimiento. Es nuestro deber como programadores capturar los errores que puedan ocurrir durante la transacción.

# Control de errores

```
CALL INGRESO_CLIENTE(171155240,'GONZALO','PAREDES','GONZALO@UCM.CL',29)
Informe de error -
ORA-00001: unique constraint (CLASE.PK_CLIENTE) violated
ORA-06512: at "CLASE.INGRESO_CLIENTE", line 12
00001. 00000 - "unique constraint (%s.%s) violated"
*Cause:      An UPDATE or INSERT statement attempted to insert a duplicate key.
              For Trusted Oracle configured in DBMS MAC mode, you may see
              this message if a duplicate entry exists at a different level.
*Action:     Either remove the unique restriction or do not insert the key.
```

Este es un ejemplo de error que no fue capturado por el procedimiento. Es nuestro deber como programadores capturar los errores que puedan ocurrir durante la transacción.

Algunos de los ORA-XXXXX más comunes tienen asociado un “nombre” que puede ser utilizado para capturar el error. En el caso del ORA-00001 su nombre asociado es DUP\_VAL\_ON\_INDEX.

# Control de errores

```
CALL INGRESO_CLIENTE(171155240,'GONZALO','PAREDES','GONZALO@UCM.CL',29)
Informe de error -
ORA-00001: unique constraint (CLASE.PK_CLIENTE) violated
ORA-06512: at "CLASE.INGRESO_CLIENTE", line 12
00001. 00000 - "unique constraint (%s.%s) violated"
*Cause:      An UPDATE or INSERT statement attempted to insert a duplicate key.
              For Trusted Oracle configured in DBMS MAC mode, you may see
              this message if a duplicate entry exists at a different level.
*Action:     Either remove the unique restriction or do not insert the key.
```

Este es un ejemplo de error que no fue capturado por el procedimiento. Es nuestro deber como programadores capturar los errores que puedan ocurrir durante la transacción.

Algunos de los ORA-XXXXX más comunes tienen asociado un “nombre” que puede ser utilizado para capturar el error. En el caso del ORA-00001 su nombre asociado es DUP\_VAL\_ON\_INDEX.

Algunos de los errores más comunes pueden revisarse [aquí](#)

# Control de errores

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE (  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS  
    --DECLARACIÓN DE VARIABLES (SI ES QUE SE NECESITAN)  
BEGIN  
    LOCK TABLE CLIENTE IN ROW EXCLUSIVE MODE;  
    INSERT INTO CLIENTE VALUES (RUTC, NOMBREC, APELLIDOSC, CORREOC, EDADC);  
    DBMS_OUTPUT.PUT_LINE('CLIENTE INGRESADO CON ÉXITO');  
    COMMIT;  
    EXCEPTION  
        WHEN DUP_VAL_ON_INDEX THEN  
            DBMS_OUTPUT.PUT_LINE('CLIENTE YA EXISTE');  
        WHEN OTHERS THEN  
            DBMS_OUTPUT.PUT_LINE('ERROR NO CAPTURADO');  
END;
```

# Control de errores

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE (  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS  
    --DECLARACIÓN DE VARIABLES (SI ES QUE SE NECESITAN)  
BEGIN  
    LOCK TABLE CLIENTE IN ROW EXCLUSIVE MODE;  
    INSERT INTO CLIENTE VALUES (RUTC, NOMBREC, APELLIDOSC, CORREOC, EDADC);  
    DBMS_OUTPUT.PUT_LINE('CLIENTE INGRESADO CON ÉXITO');  
    COMMIT;  
    EXCEPTION  
        WHEN DUP_VAL_ON_INDEX THEN  
            DBMS_OUTPUT.PUT_LINE('CLIENTE YA EXISTE');  
        WHEN OTHERS THEN  
            DBMS_OUTPUT.PUT_LINE('ERROR NO CAPTURADO');  
END;
```

WHEN DUP\_VAL\_ON\_INDEX THEN captura la excepción de repetición de clave primaria.

WHEN OTHERS THEN corresponde a la excepción general. Se activa cuando ninguna de las excepciones anteriores pudo capturar el error.

# Control de errores

Sabemos que una transacción puede finalizar con éxito o presentar algún error durante la ejecución.

# Control de errores

Sabemos que una transacción puede finalizar con éxito o presentar algún error durante la ejecución.

Oracle posee control de errores (Exceptions) para bloques anónimos, procedimientos y funciones.

# Control de errores

Sabemos que una transacción puede finalizar con éxito o presentar algún error durante la ejecución.

Oracle posee control de errores (Exceptions) para bloques anónimos, procedimientos y funciones.

Usualmente se ubican al final de cada bloque anónimo, procedimiento o función.



# Control de errores

## Anonymous

```
[DECLARE]

BEGIN
    --statements

[EXCEPTION]

END ;
```

## Procedure

```
PROCEDURE name
IS

BEGIN
    --statements

[EXCEPTION]

END ;
```

## Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
    --statements
    RETURN value;

[EXCEPTION]

END ;
```

# Control de errores

Cuando una transacción falla, es necesario revertir los cambios realizados (si es que se hicieron cambios).

# Control de errores

Cuando una transacción falla, es necesario revertir los cambios realizados (si es que se hicieron cambios).

Para realizar esto utilizamos la instrucción **ROLLBACK**.

# Concepto de ROLLBACK

La instrucción ROLLBACK es lo opuesto a la instrucción COMMIT.

# Concepto de ROLLBACK

La instrucción ROLLBACK es lo opuesto a la instrucción COMMIT.

Con esta instrucción podemos revertir parcial o totalmente los cambios realizados durante una transacción.

# Concepto de ROLLBACK

La instrucción ROLLBACK es lo opuesto a la instrucción COMMIT.

Con esta instrucción podemos revertir parcial o totalmente los cambios realizados durante una transacción.

Usualmente, la instrucción se ubica en la sección de EXCEPTION, ya que allí es donde los errores son capturados y donde también, deben revertirse los cambios realizados.

# Control de errores

## Modificando el procedimiento anterior.

```
CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE (  
    RUTC IN NUMBER,  
    NOMBREC IN VARCHAR2,  
    APELLIDOSC IN VARCHAR2,  
    CORREOC IN VARCHAR2,  
    EDADC IN NUMBER  
)  
IS  
    --DECLARACIÓN DE VARIABLES (SI ES QUE SE NECESITAN)  
BEGIN  
    LOCK TABLE CLIENTE IN ROW EXCLUSIVE MODE;  
    INSERT INTO CLIENTE VALUES (RUTC, NOMBREC, APELLIDOSC, CORREOC, EDADC);  
    DBMS_OUTPUT.PUT_LINE('CLIENTE INGRESADO CON ÉXITO');  
    COMMIT;  
    EXCEPTION  
        WHEN DUP_VAL_ON_INDEX THEN  
            DBMS_OUTPUT.PUT_LINE('CLIENTE YA EXISTE');  
            ROLLBACK;  
        WHEN OTHERS THEN  
            DBMS_OUTPUT.PUT_LINE('ERROR NO CAPTURADO');  
            ROLLBACK;  
END;
```

# Control de errores

Es posible crear nuestras propias excepciones dentro de un procedimiento.



# Control de errores

Es posible crear nuestras propias excepciones dentro de un procedimiento.

Del ejemplo anterior, crearemos un par de excepciones adicionales para hacer más robusto el procedimiento.

```

CREATE OR REPLACE PROCEDURE INGRESO_CLIENTE (
    RUTC IN NUMBER,
    NOMBREC IN VARCHAR2,
    APELLIDOSC IN VARCHAR2,
    CORREOC IN VARCHAR2,
    EDADC IN NUMBER
)
IS
    LARGO_RUT EXCEPTION;
    MENOR_EDAD EXCEPTION;
BEGIN
    LOCK TABLE CLIENTE IN ROW EXCLUSIVE MODE;
    IF EDADC > 18 THEN
        IF LENGTH(RUTC)=8 OR LENGTH(RUTC)=9 THEN
            INSERT INTO CLIENTE VALUES (RUTC, NOMBREC, APELLIDOSC, CORREOC, EDADC);
            DBMS_OUTPUT.PUT_LINE('CLIENTE INGRESADO CON ÉXITO');
            COMMIT;
        ELSE
            RAISE LARGO_RUT;
        END IF;
    ELSE
        RAISE MENOR_EDAD;
    END IF;
    EXCEPTION
        WHEN LARGO_RUT THEN
            DBMS_OUTPUT.PUT_LINE('EL RUT ES INCORRECTO');
            ROLLBACK;
        WHEN MENOR_EDAD THEN
            DBMS_OUTPUT.PUT_LINE('CLIENTE NO PUEDE SER MENOR DE EDAD');
            ROLLBACK;
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('CLIENTE YA EXISTE');
            ROLLBACK;
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('ERROR NO CAPTURADO');
            ROLLBACK;
END;

```

# Continuando con el ejercicio inicial...

```
CREATE TABLE PRODUCTO_AUX (  
    CODIGO_PRODUCTO NUMBER,  
    NOMBRE VARCHAR2 (30) ,  
    PRECIO NUMBER,  
    STOCK NUMBER  
);
```

PRODUCTO_AUX			
CODIGO_PRODUCTO	NOMBRE	PRECIO	STOCK
1	NOTEBOOK HP	\$280.990	1
2	CELULAR IPHONE 6S	\$560.000	6
3	CELULAR LG	\$450.000	7
4	MOUSE BLUETOOTH	\$28.990	9

El usuario 1, quiere comprar 2 celulares IPHONE 6S.

El usuario 2, quiere comprar 3 celulares IPHONE 6S y 1 NOTEBOOK HP.

El usuario 3, quiere comprar 2 celulares IPHONE 6S y 5 celulares LG.

El usuario 4, quiere comprar 1 NOTEBOOK HP y 1 MOUSE BLUETOOTH.

Vamos a suponer que una compra significa reducir el stock del producto

```

CREATE OR REPLACE PROCEDURE COMPRAR (
    CODIGOP IN PRODUCTO_AUX.CODIGO_PRODUCTO%TYPE,
    CANTIDAD NUMBER
)
IS
    PRODUCTO_NO_EXISTE EXCEPTION;
    STOCK_INSUFICIENTE EXCEPTION;
    CANTIDAD_STOCK NUMBER;
    CONTADOR NUMBER;
BEGIN
    SELECT COUNT(*) INTO CONTADOR FROM PRODUCTO_AUX
    WHERE CODIGO_PRODUCTO = CODIGOP;
    IF CONTADOR > 0 THEN
        SELECT STOCK INTO CANTIDAD_STOCK FROM PRODUCTO_AUX
        WHERE CODIGO_PRODUCTO = CODIGOP;
        IF CANTIDAD_STOCK >= CANTIDAD THEN
            LOCK TABLE PRODUCTO_AUX IN ROW EXCLUSIVE MODE;
            UPDATE PRODUCTO_AUX
            SET STOCK = (STOCK - CANTIDAD)
            WHERE CODIGO_PRODUCTO = CODIGOP;
            DBMS_OUTPUT.PUT_LINE('PRODUCTOS COMPRADOS');
            COMMIT;
        ELSE
            RAISE STOCK_INSUFICIENTE;
        END IF;
    ELSE
        RAISE PRODUCTO_NO_EXISTE;
    END IF;
    EXCEPTION
        WHEN PRODUCTO_NO_EXISTE THEN
            DBMS_OUTPUT.PUT_LINE('PRODUCTO NO EXISTE');
            ROLLBACK;
        WHEN STOCK_INSUFICIENTE THEN
            DBMS_OUTPUT.PUT_LINE('STOCK NO DISPONIBLE');
            ROLLBACK;
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('ERROR NO CONTROLADO');
            ROLLBACK;
END;

```

## Procedimiento de Compra

# Ejercicios simples

Realice un procedimiento que reciba 2 números como entrada y entregue como salida el número mayor.

Realice un procedimiento que reciba una palabra en minúscula y la devuelva en mayúsculas.

Realice un procedimiento que retorne la cantidad de clientes que existen en la base de datos.

# Ejercicios simples

Realice un procedimiento que pida al usuario dos números enteros, y luego entregue la suma de todos los números que están entre ellos. Ejemplo, si los números son 1 y 7, debe entregar como resultado  $2+3+4+5+6 = 20$ .

Realice un procedimiento que entregue el rut, nombre del cliente que más dinero ha gastado comprando en la tienda.

# Ejercicios

Utilizando el modelo inicial de esta diapositiva, desarrolle los siguientes procedimientos.

- Procedimiento para ingreso de un producto.
  - El control de errores debe capturar:
    - El producto ingresado ya existe.
    - El precio ingresado es negativo.
    - El stock mínimo debe ser 1.
    - La categoría asignada no existe.
  - El procedimiento debe poseer una variable de salida que indique un mensaje de éxito o fracaso.
- Procedimiento para generar una boleta.
  - El control de errores debe capturar:
    - La boleta ingresada ya existe.
    - El rut ingresado no existe.
  - El procedimiento debe poseer una variable de salida que indique un mensaje de éxito o fracaso.

# Ejercicios

Utilizando el modelo inicial de esta diapositiva, desarrolle los siguientes procedimientos.

- Procedimiento para vincular un producto a una boleta
  - El control de errores debe capturar:
    - La boleta ingresada no existe.
    - El producto ingresado no existe.
    - La cantidad ingresada es negativa.
    - La cantidad ingresada es superior al stock del producto.
    - No hay stock del producto seleccionado.
  - El procedimiento debe poseer una variable de salida que indique un mensaje de éxito o fracaso.



