EJERCICIOS ADMIN PROPUESTOS

1.

```
CREATE TABLESPACE COMPRAS DATAFILE 'COMPRAS.ORA' SIZE 5M;
```

2.

```
ALTER TABLESPACE COMPRAS ADD DATAFILE 'COMPRAS1.ORA' SIZE 1M
AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED;
```

3.

```
CREATE ROLE ROL;
GRANT INSERT, SELECT ON DEPART TO ROL;
GRANT INSERT, SELECT ON EMPLE TO ROL;
GRANT CREATE SESSION, CREATE DATABASE LINK,CREATE TABLE, CREATE
VIEW TO ROL;
```

4.

```
CREATE USER COMPRADOR IDENTIFIED BY COMPRADOR
DEFAULT TABLESPACE COMPRAS TEMPORARY TABLESPACE
TEMPORARY_DATA
QUOTA 1M ON COMPRAS QUOTA 0 ON SYSTEM;
GRANT ROL TO COMPRADOR;
```

5.
5.1
```
CREATE USER ADMINISTRADOR IDENTIFIED BY ADMINISTRADOR;
GRANT DBA TO ADMINISTRADOR;
```
5.2
```
CONNECT SYSTEM@SYSTEM
```
5.3
```
CREATE TABLE TABLA1 (N NUMBER(10));
CREATE TABLE TABLA2 (N NUMBER(10));
CREATE TABLE TABLA3 (N NUMBER(10));
```
5.4

```
CREATE USER USU1 IDENTIFIED BY USU1 DEFAULT TABLESPACE COMPRAS
QUOTA 2M ON COMPRAS;
```
5.5

```
REATE USER USU2 IDENTIFIED BY USU2 DEFAULT TABLESPACE COMPRAS
```

QUOTA 2M ON COMPRAS;

CREATE USER USU3 IDENTIFIED BY USU3 DEFAULT TABLESPACE COMPRAS QUOTA 2M ON COMPRAS;

CREATE USER USU4 IDENTIFIED BY USU4 DEFAULT TABLESPACE COMPRAS QUOTA 2M ON COMPRAS;

CREATE USER USU5 IDENTIFIED BY USU5 DEFAULT TABLESPACE COMPRAS QUOTA 2M ON COMPRAS;

Dar permisos a uno de los usuarios usu1, para que solo pueda conectarse a la BD.
GRANT CREATE SESSION TO USU1;
6.

CREATE ROLE ROL_USU;
GRANT CREATE SESSION TO ROL_USU;
GRANT SELECT ON TABLA1 TO ROL_USU;
GRANT SELECT ON TABLA2 TO ROL_USU;
GRANT SELECT ON TABLA3 TO ROL_USU;

7.
GRANT CREATE SESSION TO ROL_USU;
GRANT SELECT ON TABLA1 TO ROL_USU;
GRANT SELECT ON TABLA2 TO ROL_USU;
GRANT SELECT ON TABLA3 TO ROL_USU;

8.

GRANT INSERT, DELETE ON CLIENTE TO USU4 WITH GRANT OPTION;
GRANT INSERT, DELETE ON PROVEEDOR TO USU4 WITH GRANT OPTION;

9.
GRANT CREATE ANY TABLE,CREATE USER TO USU5 WITH ADMIN OPTION;
GRANT ALTER TABLESPACE, CREATE TABLESPACE TO USU5;

10.

GRANT CREATE ANY TABLE,CREATE USER TO USU5 WITH ADMIN OPTION;
GRANT ALTER TABLESPACE, CREATE TABLESPACE TO USU5;

GRANT UPDATE(N) ON TABLA1 TO PUBLIC;

```
GRANT UPDATE(N) ON TABLA2 TO PUBLIC;
```

11.

```
REVOKE ROL_USU FROM USU3;
REVOKE ALL ON TABLA1 FROM USU4;
REVOKE ALL ON TABLA2 FROM USU4;
```

12.

```
CREATE PROFILE LIM_SESION LIMIT SESSIONS_PER_USER 2;
ALTER USER USU5 IDENTIFIED BY USU5 PROFILE LIM_SESION;
```

13.

```
CREATE PROFILE TIEMPOLIMITE LIMIT CONNECT_TIME 5;
ALTER USER USU2 IDENTIFIED BY USU2 PROFILE TIEMPOLIMITE;
ALTER USER USU3 IDENTIFIED BY USU3 PROFILE TIEMPOLIMITE;
```

EJERCICIOS PROPUESTOS PSQL

1.

```
BEGIN
DBMS_OUTPUT.PUT_LINE('HOLA');
END;
```

2. Muestra por pantalla la cantidad total de productos; almacenados en la variable “v_num”.

3.

```
BEGIN
SELECT count(*) INTO v_num
```

```
FROM productos;
DBMS_OUTPUT.PUT_LINE(v_num);
END

SAVE A:\EJPROP.SQL REPLACE
```

4.

```
SELECT count(*) FROM productos;
```

5.

```
START A:\EJPROP.SQL
```

6.

La cabecera del procedimiento.

```
CREATE OR REPLACE PROCEDURE modificar_precio_producto
(codigoprod NUMBER, nuevoprecio NUMBER) AS
precioant NUMBER(5);
```

- El precioant. El precioant.

Una variable numérica.

- El nuevoprecio.

Un parámetro de entrada.

- El precio_uni.

Una columna de la tabla productos.

- Los parámetros del procedimiento.

codigoprod y nuevoprecio.

- NO_DATA_FOUND.

Una excepción que se produce cuando una sentencia SELECT..INTO no ha recuperado
ninguna fila.

- El nombre del procedimiento.

modificar_precio_producto

- ¿Dónde comienza el bloque?

Con la palabra reservada AS.

- ¿Qué hace la cláusula INTO?

Deposita el valor (o los valores) recuperado por la cláusula SELECT en las variables que
siguen a la cláusula INTO.

- ¿Qué hace la expresión: (precioant * 0.20) > ABS(precioant – nuevoprecio)?.

Comprueba si la diferencia entre precioant y precionuevo supera el veinte por ciento (del
primero).

- ¿Por qué no tiene la cláusula DECLARE? ¿Qué tiene en su lugar?
La cláusula DECLARE se utiliza con bloques anónimos con procedimientos y funciones se
utiliza en su lugar la cláusula IS o AS indistintamente. (Por concordancia gramatical se suele
utilizar AS con CREATE OR REPLACE PROCEDURE/FUNCTION).

1). Indicar los errores que aparecen en las siguientes instrucciones y la forma de corregirlos.

```
DECLARE
Num1 NUMBER(8,2) := 0
Num2 NUMBER(8,2) NOT NULL DEFAULT 0;
Num3 NUMBER(8,2) NOT NULL;
Cantidad INTEGER(3);
Precio, Descuento NUMBER(6);
Num4 Num1%ROWTYPE;
Dto CONSTANT INTEGER;
BEGIN
...
END;
```

```
Num3 NUMBER(8,2) NOT NULL DEFAULT 0;
Cantidad INTEGER;
Precio NUMBER(6);
Descuento NUMBER(6);
Num4 Num1%TYPE;
```

2). Escribir un procedimiento que reciba dos números y visualice su suma.

```
CREATE OR REPLACE PROCEDURE sumar_numeros (

num1 NUMBER,
num2 NUMBER)
```

```
IS
suma NUMBER(6);
BEGIN
suma := num1 + num2;
DBMS_OUTPUT.PUT_LINE('Suma: '|| suma);
END sumar_numeros;
```

3). Codificar un procedimiento que reciba una cadena y la visualice al revés.

```
CREATE OR REPLACE PROCEDURE cadena_reves(
vcadena VARCHAR2)
AS
vcad_reves VARCHAR2(80);
BEGIN
FOR i IN REVERSE 1..LENGTH(vcadena) LOOP
vcad_reves := vcad_reves || SUBSTR(vcadena,i,1);
END LOOP;
DBMS_OUTPUT.PUT_LINE(vcad_reves);
END cadena_reves;
```

4). Escribir una función que reciba una fecha y devuelva el año, en número, correspondiente a
esa fecha.

```
CREATE OR REPLACE FUNCTION anio (
fecha DATE)
RETURN NUMBER
AS
v_anio NUMBER(4);
BEGIN
v_anio := TO_NUMBER(TO_CHAR(fecha, 'YYYY'));
RETURN v_anio;



END anio;
```

5). Escribir un bloque PL/SQL que haga uso de la función anterior.

```
DECLARE
n NUMBER(4);
BEGIN
n := anio(SYSDATE);
DBMS_OUTPUT.PUT_LINE('AÑO : '|| n);
END;
```

6). Dado el siguiente procedimiento:

```
CREATE OR REPLACE PROCEDURE crear_depart (
v_num_dept depart.dept_no%TYPE,
v_dnombre depart.dnombre%TYPE DEFAULT 'PROVISIONAL',
v_loc depart.loc%TYPE DEFAULT ‘PROVISIONAL’)
IS
BEGIN
INSERT INTO depart
VALUES (v_num_dept, v_dnombre, v_loc);
END crear_depart;
```

Indicar cuáles de las siguientes llamadas son correctas y cuáles incorrectas, en este último caso
escribir la llamada correcta usando la notación posicional (en los casos que se pueda):

```
crear_depart;
```

Incorrecta: hay que pasar al menos el número de departamento.

```
crear_depart(50);
```

Correcta.

```
crear_depart('COMPRAS');
```

Incorrecta: hay que pasar también el número de departamento.

```
crear_depart(50,'COMPRAS');
```

Correcta.

```
crear_depart('COMPRAS', 50);
```

Incorrecta: los argumentos están en orden inverso. Solución: crear_depart(50, 'COMPRAS');

```
crear_depart('COMPRAS', 'VALENCIA');
```

Incorrecta: hay que pasar también el número.

```
crear_depart(50, 'COMPRAS', 'VALENCIA');
```

Correcta.

```
crear_depart('COMPRAS', 50, 'VALENCIA');
```

Incorrecta: el orden de los argumentos es incorrecto. Solución: crear_depart(50, 'COMPRAS',
'VALENCIA');

```
crear_depart('VALENCIA', ‘COMPRAS’);
```

Incorrecta: hay que pasar también el número de departamento.

```
crear_depart('VALENCIA', 50);
```

Incorrecta: los argumentos están en orden inverso. Solución: crear_depart(50, NULL,
'VALENCIA');

7). Desarrollar una función que devuelva el número de años completos que hay entre dos
fechas que se pasan como argumentos.

```
CREATE OR REPLACE FUNCTION anios_dif (
fecha1 DATE, fecha2 DATE)
RETURN NUMBER
AS
v_anios_dif NUMBER(6);
BEGIN
v_anios_dif := ABS(TRUNC(MONTHS_BETWEEN(fecha2,fecha1) / 12));
RETURN v_anios_dif;
END anios_dif;
```

8). Escribir una función que, haciendo uso de la función anterior devuelva los trienios que hay
entre dos fechas. (Un trienio son tres años completos).

```
CREATE OR REPLACE FUNCTION trienios (
fecha1 DATE,
fecha2 DATE)
RETURN NUMBER

AS
v_trienios NUMBER(6);
BEGIN
v_trienios := TRUNC(anios_dif(fecha1,fecha2) / 3);
RETURN v_trienios;
END;
```

9). Codificar un procedimiento que reciba una lista de hasta 5 números y visualice su suma.

```
CREATE OR REPLACE PROCEDURE sumar_5numeros (
Num1 NUMBER DEFAULT 0,
Num2 NUMBER DEFAULT 0,
Num3 NUMBER DEFAULT 0,
Num4 NUMBER DEFAULT 0,
Num5 NUMBER DEFAULT 0)
AS
BEGIN
DBMS_OUTPUT.PUT_LINE(Num1 + Num2 + Num3 + Num4 + Num5);
END sumar_5numeros;
```

10). Escribir una función que devuelva solamente caracteres alfabéticos sustituyendo cualquier
otro carácter por blancos a partir de una cadena que se pasará en la llamada.

```
CREATE OR REPLACE FUNCTION sust_por_blancos(
cad VARCHAR2)
RETURN VARCHAR2
```

```
AS
nueva_cad VARCHAR2(30);
car CHARACTER;
BEGIN
FOR i IN 1..LENGTH(cad) LOOP
car:=SUBSTR(cad,i,1);
IF (ASCII(car) NOT BETWEEN 65 AND 90)

BEGIN
OPEN c_emp;
oficio_ant:='*';
FETCH c_emp INTO vr_emp;
WHILE c_emp%FOUND LOOP
IF oficio_ant <> vr_emp.oficio THEN
oficio_ant := vr_emp.oficio;
i := 1;
END IF;
IF i <= 2 THEN
DBMS_OUTPUT.PUT_LINE(vr_emp.oficio||' * '

||vr_emp.apellido||' * '
||vr_emp.salario);
END IF;
FETCH c_emp INTO vr_emp;
i:=I+1;
END LOOP;
CLOSE c_emp;
END emp_2minsal;
```

6). Escribir un programa que muestre, en formato similar a las rupturas de control o secuencia
vistas en SQL*plus los siguientes datos:
- Para cada empleado: apellido y salario.
- Para cada departamento: Número de empleados y suma de los salarios del departamento.
- Al final del listado: Número total de empleados y suma de todos los salarios.

```
CREATE OR REPLACE PROCEDURE listar_emple
AS
CURSOR c1 IS
SELECT apellido, salario, dept_no FROM emple
ORDER BY dept_no, apellido;
vr_emp c1%ROWTYPE;
```

```
dep_ant EMPLE.DEPT_NO%TYPE;
cont_emple NUMBER(4) DEFAULT 0;
sum_sal NUMBER(9) DEFAULT 0;
tot_emple NUMBER(4) DEFAULT 0;
tot_sal NUMBER(10) DEFAULT 0;
BEGIN
OPEN c1;
FETCH c1 INTO vr_emp;
IF c1%FOUND THEN
dep_ant := vr_emp.dept_no;
END IF;
WHILE c1%FOUND LOOP

/* Comprobación nuevo departamento y resumen */

IF dep_ant <> vr_emp.dept_no THEN

DBMS_OUTPUT.PUT_LINE('*** DEPTO: ' || dep_ant ||

' NUM. EMPLEADOS: '||cont_emple ||

' SUM. SALARIOS: '||sum_sal);

dep_ant := vr_emp.dept_no;

tot_emple := tot_emple + cont_emple;

tot_sal:= tot_sal + sum_sal;
cont_emple:=0;

sum_sal:=0;
END IF;

/* Líneas de detalle */

DBMS_OUTPUT.PUT_LINE(RPAD(vr_emp.apellido,10)|| ' * '

||LPAD(TO_CHAR(vr_emp.salario,'9,999,999'),12));

/* Incrementar y acumular */

cont_emple := cont_emple + 1;

sum_sal:=sum_sal + vr_emp.salario;
```

```
FETCH c1 INTO vr_emp;
END LOOP;
CLOSE c1;
IF cont_emple > 0 THEN

/* Escribir datos del último departamento */
DBMS_OUTPUT.PUT_LINE('*** DEPTO: ' || dep_ant ||

' NUM EMPLEADOS: '|| cont_emple ||

' SUM. SALARIOS: '||sum_sal);

dep_ant := vr_emp.dept_no;

tot_emple := tot_emple + cont_emple;

tot_sal:= tot_sal + sum_sal;
cont_emple:=0;

sum_sal:=0;

/* Escribir totales informe */

DBMS_OUTPUT.PUT_LINE(' ****** NUMERO TOTAL EMPLEADOS: '
||tot_emple ||
' TOTAL SALARIOS: '|| tot_sal);
END IF;
END listar_emple;

/* Nota: este procedimiento puede escribirse de forma que la visualización de los
resultados
resulte más clara incluyendo líneas de separación, cabeceras de columnas,
etcétera. Por
razones didácticas no se han incluido estos elementos ya que pueden distraer y
dificultar la
comprensión del código. */
```

7). Desarrollar un procedimiento que permita insertar nuevos departamentos según las
siguientes especificaciones:
Se pasará al procedimiento el nombre del departamento y la localidad.

El procedimiento insertará la fila nueva asignando como número de departamento la decena
siguiente al número mayor de la tabla.
Se incluirá gestión de posibles errores.

```
CREATE OR REPLACE PROCEDURE insertar_depart(
nombre_dep VARCHAR2,
loc VARCHAR2)
AS
CURSOR c_dep IS SELECT dnombre
FROM depart WHERE dnombre = nombre_dep;
v_dummy DEPART.DNOMBRE%TYPE DEFAULT NULL;
v_ulti_num DEPART.DEPT_NO%TYPE;
nombre_duplicado EXCEPTION;
BEGIN
/* Comprobación de que el departamento no está duplicado */
OPEN c_dep;
FETCH c_dep INTO v_dummy;
CLOSE c_dep;
IF v_dummy IS NOT NULL THEN
RAISE nombre_duplicado;
END IF;

/* Captura del último número y cálculo del siguiente */
SELECT MAX(dept_no) INTO v_ulti_num FROM depart;
/* Inserción de la nueva fila */
INSERT INTO depart VALUES ((TRUNC(v_ulti_num, -1)+10)
, nombre_dep, loc);
EXCEPTION
WHEN nombre_duplicado THEN
DBMS_OUTPUT.PUT_LINE('Err. departamento duplicado');

RAISE;
WHEN OTHERS THEN


IF UPDATING ('APELLIDO') THEN
v_cad_inser := v_cad_inser
||:OLD.APELLIDO|| '*'||:NEW.APELLIDO;
END IF;
IF UPDATING ('OFICIO') THEN
v_cad_inser := v_cad_inser
||:OLD.OFICIO|| '*'||:NEW.OFICIO;
END IF;
IF UPDATING ('DIR') THEN
```

```
v_cad_inser := v_cad_inser
||:OLD.DIR|| '*'||:NEW.DIR;
END IF;
IF UPDATING ('FECHA_ALT') THEN
v_cad_inser := v_cad_inser
||:OLD.FECHA_ALT||:NEW.FECHA_ALT;
END IF;
IF UPDATING ('SALARIO') THEN
v_cad_inser := v_cad_inser
||:OLD.SALARIO|| '*'||:NEW.SALARIO;
END IF;
IF UPDATING ('COMISION') THEN
v_cad_inser := v_cad_inser
||:OLD.COMISION|| '*'||:NEW.COMISION;
END IF;

IF UPDATING ('DEPT_NO') THEN
v_cad_inser := v_cad_inser
||:OLD.DEPT_NO|| '*'||:NEW.DEPT_NO;
END IF;
INSERT INTO AUDITAREMPLE VALUES(v_cad_inser);
END;
```

3). Escribir un disparador de base de datos que haga fallar cualquier operación de modificación
del apellido o del número de un empleado, o que suponga una subida de sueldo superior al
10%.

```
CREATE OR REPLACE TRIGGER fallo_modif
BEFORE UPDATE OF apellido, emp_no, salario
ON emple
FOR EACH ROW
BEGIN
IF UPDATING('emp_no') OR UPDATING('apellido')
OR (UPDATING ('salario') AND
:new.salario>:old.salario*1.1)
THEN

RAISE_APPLICATION_ERROR
(-20001,'Err. Modificacion no permitida');
END IF;
END;
```

4). Suponiendo que disponemos de esta vista

```
CREATE VIEW DEPARTAM AS
SELECT DEPART.DEPT_NO, DNOMBRE, LOC, COUNT(EMP_NO) TOT_EMPLE
FROM EMPLE, DEPART
WHERE EMPLE.DEPT_NO (+) = DEPART.DEPT_NO
GROUP BY DEPART.DEPT_NO, DNOMBRE, LOC;
```

Construir un disparador que permita realizar operaciones de actualización en la tabla depart a
partir de la vista dptos, de forma similar al ejemplo del trigger t_ges_emplead. Se contemplarán las siguientes operaciones:

- Insertar departamento.
- Borrar departamento.
- Modificar la localidad de un departamento.

```
CREATE OR REPLACE TRIGGER ges_depart
INSTEAD OF DELETE OR INSERT OR UPDATE
ON DEPARTAM


FOR EACH ROW
BEGIN
IF DELETING THEN
DELETE FROM depart WHERE dept_no = :old.dept_no;
ELSIF INSERTING THEN
INSERT INTO depart
VALUES(:new.dept_no, :new.dnombre, :new.loc);
ELSIF UPDATING('loc') THEN
UPDATE depart SET loc = :new.loc
WHERE dept_no = :old.dept_no;
ELSE
RAISE_APPLICATION_ERROR
(-20001,'Error en la actualización');
END IF;
END;
```

EJERCICIO 1

Sea la siguiente BD:
PROVEEDORES(nro-p,nom-p,categoría,ciud-p)
ITEMS(nro-i,descripción-i,ciud-i)
PEDIDOS(nro-p,nro-c,nro-i,cantidad,precio)
CLIENTES(nro-c,nom-c,ciud-c)

1. Listar los proveedores de Córdoba.
π nom-p (σ ciud-p =córdoba (PROVEEDORES)

SELECT Nro_P, Nom_P FROM PROVEEDORES WHERE CIUD_P = 'CORDOBA';

2. Listar los proveedores que proveen el item "i1".
π nom-c (σ nro-p =”p1”) (CLIENTES) (PROVEEDORES)

SELECT P.Nro_P, P.Nom_P FROM ITEMS AS I, PEDIDOS AS PD, PROVEEDORES AS P WHERE I.Descripcion_I = 'I1' AND PD.Nro_I = I.Nro_I AND P.Nro_P = PD.Nro_P;

3. Listar los clientes que solicitan items provistos por "p1".
π nom-c (σ nro-p =”p1”) (CLIENTES) (PROVEEDORES)

SELECT C.NRO_C, C.Nom_C FROM CLIENTES AS C, PEDIDOS AS PD, PROVEEDORES AS P WHERE P.Nom_P = 'P1' AND PD.Nro_P = P.Nro_P AND C.Nro_C = PD.Nro_C;

4. Listar los clientes que solicitan algún ítem provisto por proveedores con categoria mayor que 4.
π nom-c (CLIENTES* (π descripción –i (ÍTEMS)* (σ categoría >4(ITEMS))

SELECT C.Nro_C, C.Nom_C FROM CLIENTES AS C WHERE C.Nro_C IN (SELECT DISTINCT PD.Nro_C FROM PEDIDOS AS PD, PROVEEDORES AS P WHERE P.Categoria > 4 AND PD.Nro_P = P.Nro_P);

5. Listar los ítems pedidos por clientes de Rosario.
π nro-i (PEDIDOS)- π nro-i (PEDIDOS * σ nom-p= Rosario (PROVEEDORES))

SELECT I.Nro_I, I.Descripcion_I FROM ITEMS AS I WHERE I.Nro_I IN (SELECT PD.Nro_I FROM CLIENTES AS C, PEDIDOS AS PD WHERE C.Ciud_C = 'Rosario' AND PD.Nro_C = C.Nro_C);

6. Listar los pedidos en los cuales un cliente de Rosario solicita artículos fabricados en Mendoza (ciud-i = "Mendoza").

π cantidad (PEDIDOS)* π nom-p =Rosario (PROVEEDORES)* σ ciud –i =”Mendoza” (ÍTEMS).
SELECT PR.nom_p, CL.nom_c, IT.descripcion_i, PE.cantidad, PE.precio FROM proveedores AS PR, clientes AS CL, items AS IT, pedidos AS PE WHERE PR.nro_p = PE.nro_p AND CL.nro_c = PE.nro_c AND IT.nro_i = PE.nro_i AND CL.ciud_c = 'rosario' AND IT.ciud_i = 'mendoza';

7. Listar los pedidos en los que el cliente "23" solicita items no solicitados por el cliente "30".
π cantidad (σ nro-c =23) (PEDIDOS) π cantidad (π nro-c=30) (PEDIDOS)

SELECT PR1.nom_p, CL1.nom_c, IT1.descripcion_i, PE1.cantidad, PE1.precio FROM proveedores PR1, clientes CL1, items IT1, pedidos PE1 WHERE PR1.nro_p = PE1.nro_p AND CL1.nro_c = PE1.nro_c AND IT1.nro_i = PE1.nro_i AND PE1.nro_c = 23 AND PE1.nro_i not in (SELECT IT2.nro_i FROM items IT2, pedidos PE2 WHERE IT2.nro_i = PE2.nro_i AND PE2.nro_c = 30);

8. Listar las ciudades en la forma (ciu1,ciu2) tales que un proveedor en ciu1 provea items solicitados por clientes de ciu2.
σ Ciud-p=ciu1 (π ciud-c=ciu2)(PROVEEDORES)(CLIENTES)

SELECT DISTINCT P.Ciud_P AS Ciudad1, C.Ciud_C AS Ciudad2 FROM PROVEEDORES AS P, CLIENTES AS C, PEDIDOS AS PD WHERE P.Nro_P = PD.Nro_P AND C.Nro_C = PD.Nro_C;

9. Listar los números de proveedores cuya categoría sea mayor que la de todos los proveedores que proveen el item "cuaderno".
π nro-p (σ categoría> Todos proveedores) (π descripción-i =cuaderno)(PROVEEDORES)(ítems)

SELECT P.Nro_P, P.Nom_P FROM PROVEEDORES AS P WHERE P.Categoria > (SELECT MAX(P.Categoria) FROM PROVEEDORES AS P, PEDIDOS as PD, ITEMS AS I WHERE I.Descripcion_I = 'Cuaderno' AND PD.Nro_I = I.Nro_I AND P.Nro_P = PD.Nro_P;

10. Listar los clientes que han pedido 2 o más ítems distintos.
π nom-c (σ cantidad ≥ 2)* π nro-i ≥ 2 (PEDIDOS)

SELECT DISTINCT PD1.Nro_C, C.Nom_C FROM PEDIDOS AS PD1, PEDIDOS AS PD2, CLIENTES AS C WHERE PD1.Nro_C = PD2.Nro_C AND PD1.Nro_I <> PD2.Nro_I AND C.Nro_C = PD1.Nro_C;

11. Listar los proveedores que proveen a todos los clientes de Córdoba una cantidad mayor que el promedio de las cantidades pedidas por los clientes de Rosario.
π nom-p (σ ciud-p=córdoba)* π cantidad> Σ cantidades (σ nom-p=N.de cantidad Rosario) (PROVEEDORES, PEDIDOS)

SELECT P.Nro_P, P.Nom_P FROM PROVEEDORES AS P, PEDIDOS AS PD, CLIENTES AS C WHERE C.Ciud_C = 'Cordoba' AND PD.Nro_C = C.Nro_C AND PD.Cantidad > (SELECT AVG(SUMA) FROM TEMP) AND P.Nro_P = PD.Nro_P GROUP BY P.Nro_P, P.Nom_P HAVING COUNT(C.Nro_C) = (SELECT COUNT(C2.Nro_C) FROM CLIENTES C2 , PEDIDOS PD2 WHERE C2.Ciud_C = 'Cordoba' AND C2.Nro_C = PD2.Nro_C AND PD2.Nro_P = P.Nro_P);

TABLA TEMP

SELECT C2.Nro_C, SUM(PD2.Cantidad) AS Suma FROM PEDIDOS AS PD2, CLIENTES AS C2 WHERE PD2.Nro_C = C2.Nro_C AND C2.Ciud_C = 'Rosario' GROUP BY C2.Nro_C;

EJERCICIO 2

Sea la siguiente Base de Datos:
VUELOS (nro-vuelo,desde,hasta)
AVION-UTILIZADO(nro-vuelo,tipo-avión,nro-avión)
INFO-PASAJEROS(nro-vuelo,dni,nombre,origen,destino)
Los vuelos no pueden tener más de dos escalas y no hay cambio de tipo de avión para un mismo número de vuelo. Realizar las siguientes consultas:

1. Listar los números de vuelo de A hasta F.
σ nro-vuelo (π desde=A, hasta=F) (VUELOS)

SELECT nro-vuelo FROM VUELOS WHERE desde =“A%” AND hasta=”F%”;

2. Listar los tipos de avión que no son utilizados en ningún vuelo que pase por B.
π tipo-avión (σ destino ≠B) (AVION-UTILIZADO)(INFO-PASAJEROS)

SELECT A.tipo-avión from AVION-UTILIZADO A, INFO-PASAJEROS I WHERE I.destino not in “B”;

3. Listar los pasajeros y números de vuelo para aquellos pasajeros que viajan de A a D pasando por B.
π nombre, nro-vuelo (σ origen=A, π destino=D) (INFO-PASAJEROS)

SELECT P.nombre, V.nro-vuelo FROM INFO-PASAJEROS P, VUELOS V WHERE P.origen=”A” AND P.destino=”D”.

4. Listar los tipos de avión que son utilizados en todos los vuelos que pasan por C.
π tipo-avión (σ destino=C)(AVION-UTILIZADO)(INFO-PASAJEROS)

SELECT A.tipo-avión FROM AVION-UTILIZADO A, INFO-PASAJEROS I WHERE I.destino=”C”;

EJERCICIO 5

Sea la siguiente Base de Datos:
Personas(nro_doc, nombre, domicilio)
Supervisa_a(nro_doc_supervisor,nro_doc_supervisado)
Asumir que cada persona es supervisada a lo sumo por un supervisor.
Escribir en AR la siguiente consulta: Listar los nombres de las personas que trabajan con Juan Perez (suponer que existe un único Juan Perez)

π nombre (σ nro_doc_supervisor=”Juan Perez” (Supervisa)) (Personas)

SELECT P.nombre FROM Personas P, Supervisa S WHERE P.nro_doc=S.nro_doc_supervisado AND S.nro_doc_supervisor=”Juan Perez”;

EJERCICIO 6

Sea la siguiente Base de Datos:
Empleados(nro_e, nombre, domicilio, ciudad)
Asignado_a(nro_e, cod_tarea, cant_horas)
Tareas(cod_tarea, desc_tarea)
Escribir en AR la siguiente consulta, sin utilizar operadores derivados: Listar los nombres de las personas asignadas a todas las tareas

π nombre (σ nro_e =(σ group by cod_tarea (Tareas) (Asignado)(Empleado))

SELECT E.nombre from Empleados E, Asignado A, Tareas T WHERE E.nro_e=A.nro_e AND A.cod_tarea=T.cod tarea AND E.nro_e in (SELECT T.cod_tarea FROM Tareas GROUP BY cod_tarea);

EJERCICIO 7

Para la siguiente base de datos:
ALUMNOS(nroLeg,nombre,domicilio,planDeEstudios)
MATERIAS(nroMat,nombre,planDeEstudios)
EXAMENES(nroLeg,nroMat,nota,fecha)
CURSAN(nroLeg, nroMat)
CORRELATIVA(nroMat,nroCorrelat)
Expresar en Algebra Relacional la siguiente consulta: Listar los nombres de los alumnos que no cursan ninguna materia de la cual hayan rendido su correlativa al menos dos veces.

π nombre (σ norCorrelat >= 2 (CORRELATIVA)(MATERIAS)(EXAMENES)(ALUMNOS)(CURSAN)

SELECT A.nombre FROM ALUMNOS A, MATERIAS M, EXAMENES E, CURSAN C, CORRELATIVA CO WHERE A.nroLeg=E.nroLeg AND E.nroMat=M.nroMat AND M.nroMat=C.nroMat AND CO.norCorrelta >= 2;

EJERCICIO 8

Sea la siguiente base de datos:
ALUMNOS(nroLeg,nombre,domicilio,planDeEstudios)
MATERIAS(nroMat,nombre,planDeEstudios)
EXAMENES(nroLeg,nroMat,nota,fecha)
CURSAN(nroLeg, nroMat)
Expresar en Algebra Relacional la siguiente consulta: Listar los nombres de los alumnos que no cursan ninguna materia en la cual ningún alumno que rindió la materia al menos 2 veces obtuvo una nota mayor que 6 en un final.

π nombre (σ nroMat not in (σ nroLeg>=2, nota>=6
(Tareas)(CURSAN)(MATERIAS)(EXAMENES)(ALUMNOS))

SELECT A.nombre FROM ALUMNOS A, MATERIAS M, EXAMENES E, CURSAN C WHERE A.nroLeg=E.nroLeg AND E.nroMat=M.nroMat AND M.nroMat=C.nroMat AND C.nroMat not in (SELECT nroMat FROM EXAMENES WHERE nroLeg>=2 AND nota>=6);

EJERCICIO 9

Para la siguiente base de datos:
ALUMNOS(dni, nombre,apellido,nacionalidad)
FACULTADES(codfacultad, nombre)
CARRERAS(codcarrera, nombre,codfacultad,cantidadDeAlumnos) codfacultad foreign key references facultades(codfacultad)
ESTUDIANTEDE(dni,codcarrera)
Expresar en Algebra Relacional la consulta: Listar los nombres de los estudiantes que no estudian carreras que se dictan en facultades en las que estudia algún estudiante español.

π nombre (σ codfacultad not in (σ nacionalidad=”Español”)
(FACULTAD)(CARRERAS)(EXAMENES)(ALUMNOS))

SELECT A.nombre FROM ALUMNOS A, CARRERAS C, FACULTADES F, ESTUDIANTES E WHERE A.dni=E.dni AND E.codcarrera=C.codcarrera AND C.codfacultad=F.codfacultad AND F.codfacultad not in (SELECT F.codfacultad FROM ALUMNOS A, CARRERAS C, FACULTADES F, ESTUDIANTES E WHERE A.dni=E.dni AND E.codcarrera=C.codcarrera AND C.codfacultad=F.codfacultad AND A.nacionalidad=”Español”);

EJERCICIO 10

Sea la siguiente base de datos:
ALUMNOS(nro_leg,nombre,domicilio,plan_de_estudios)
CURSOS(nro_curso,nombre,horario)
MATERIAS(nro_mat,nombre,plan_de_estudios)
APROBO(nro_leg,nro_mat,nota)
Expresar la siguiente consulta en Algebra Relacional:
1. Listar los nombres de los alumnos que aprobaron solamente materias correspondientes a su plan de estudios.

π nombre (σ nroleg in (π nroleg (APROBO))(MATERIAS)(ALUMNOS))

SELECT A.nombre FROM ALUMNOS A, MATERIAS M, APROBO AP WHERE A.plan_de_estudios=M.plan_de_estudios AND M.nro_mat=AP.nro_mat AND A.nroleg in (SELECT nroleg FROM APROBO);

2. Listar las materias aprobadas con 9 puntos por al menos 2 alumnos que no hayan cursado ninguna materia del plan 96.

π nombre (σ nota=9(APROBO))(σ plan_de_estudios=96(ALUMNOS))(MATERIAS))

SELECT M.nombre FROM MATERIAS M, ALUMNOS A, APROBO AP WHERE M.plan_de_estudios=A.plan_de_estudios AND A.nroleg=AP.nroleg AND AP.nota=9 AND A.plan_de_estudios=96;

3. Listar las materias no aprobadas por ningún alumno que haya obtenido mas de 8 puntos en alguna materia correspondiente a su mismo plan de estudios.

π nombre (σ nroleg in (π nroleg (σ nota>8)(MATERIAS)(ALUMNOS)(APROBO)))

SELECT M.nombre FROM MATERIAS M, ALUMNOS A, APROBO AP, WHERE M.nro_mat=AP.nro_mat AND AP.plan_de_estudios=A.plan_de_estudios AND A.nroleg in (SELECT nroleg FROM APROBO WHERE nota>8);

EJERCICIO 12

Sea la BD:
PERSONAS(tipo-doc,num-doc,nomyap,dir,tel,fnac,sexo)
PROGENITOR(tipo-doc,num-doc,tipo-doc-hijo,num-doc-hijo)
1. Listar para cada Juan Perez los tipo y número de documento, nombre y apellido y teléfonos de todos sus hijos.

π nomyap, tipo-doc (σ nomyap=”Juan Perez”(PERSONAS)(PROGENITOR))

SELECT P.nomyap, PR.tipo-doc, PR.num-doc FROM PERSONAS P, PROGENITOR PR WHERE P.tipo-doc=PR.tipo-doc AND P.nomyap=”Juan Perez” GROUP BY P.nomyap.

2. b.-Idem a ,de :
    1. todos sus hermanos (los hijos de su padre y/o su madre).

π nomyap, tipo-doc (σ nomyap=”Juan Perez”(σ sexo=”h” U σ sexo=”m” (PERSONAS)(PROGENITOR)))

SELECT P.nomyap, PR.tipo-doc, PR.num-doc FROM PERSONAS P, PROGENITOR PR WHERE P.tipo-doc=PR.tipo-doc AND P.nomyap=”Juan Perez” AND (P.sexo=”h” OR P.sexo=”m”)GROUP BY P.nomyap.

    2. su madre.

π nomyap, tipo-doc (σ nomyap=”Juan Perez”(PERSONAS)(σ tipo-doc=”madre”PROGENITOR)))

SELECT P.nomyap, PR.tipo-doc, PR.num-doc FROM PERSONAS P, PROGENITOR PR WHERE P.tipo-doc=PR.tipo-doc AND P.nomyap=”Juan Perez” AND PR.tipo-doc=”madre”GROUP BY P.nomyap.

    3. su abuelo materno.

π nomyap, tipo-doc (σ nomyap=”Juan Perez”(PERSONAS)(σ tipo-doc=”abuelo materno”PROGENITOR)))

SELECT P.nomyap, PR.tipo-doc, PR.num-doc FROM PERSONAS P, PROGENITOR PR WHERE P.tipo-doc=PR.tipo-doc AND P.nomyap=”Juan Perez” AND PR.tipo-doc=”abuelo materno”GROUP BY P.nomyap.

    4. todos sus nietos.

π nomyap, tipo-doc (σ nomyap=”Juan Perez”(PERSONAS)(σ tipo-doc=”abuelo materno”PROGENITOR)))

SELECT P.nomyap, PR.tipo-doc, PR.num-doc FROM PERSONAS P, PROGENITOR PR WHERE P.tipo-doc=PR.tipo-doc AND P.nomyap=”Juan Perez” AND PR.tipo-doc=”nieto”GROUP BY P.nomyap.

EJERCICIO 13

Una oficina gubernamental desea construir un complejo habitacional, para lo cual elaboró la siguiente Base de Datos:
TRAMO(c_ciudadA, c_ciudadB, Distancia).
CIUDADES(c_ciudad, nombre, cant_escuelas, cant_fábricas).
Para decidir dónde instalarlo, desea conocer los siguientes datos:
1. Las ciudades alcanzables desde la ciudad con mayor cantidad de fábricas, recorriendo no más de 2 tramos, c/u de los cuales no puede tener más de 10Km de longitud.

π nombre (σ ciudadB<=2, distancia<=10(TRAMO)(σ max(cant_fabricas)(CIUDADES)))

SELECT C.nombre FROM TRAMO T, CIUDADES C WHERE C.cant_fabricas=(SELECT MAX(cant_fabricas) FROM CIUDADES) T.ciudadB<=2 AND T.distancia<=10;

2. Las ciudades con más de 10 fábricas, que estén conectadas en forma directa con todas las demás, siempre que ningún tramo supere los 50 Km.

π nombre (σ ciudadB<=2, distancia<=10(TRAMO)(CIUDADES))

SELECT C.nombre FROM TRAMO T, CIUDADES C WHERE C.cant_fabricas >10 AND T.ciudadB=21 AND T.distancia<=50;

3. Los pares de ciudades, de la forma (ciudad1,ciudad2), que son alcanzables a través de, a lo sumo, otra ciudad, llamemos a ésta, ciudad3 (esto es, encontrar lo caminos ciudad1-ciudad3-ciudad2).

π nombre (σ ciudadA=1, ciudadB=1(TRAMO)(CIUDADES))

SELECT T.ciudadA, T.ciudadB, C.nombre FROM TRAMO T, CIUDADES C WHERE T.ciudadA=1 AND T.ciudadB=1 ;

EJERCICIO 14

Sea la siguiente Base de Datos:
Artículos(nro_art,descripcion,color,peso,precio_unit,fabricado_en)
Factura(nro_fact,nro_cli,fecha, fecha_venc)
DetalleFactura(nro_fact,nro_art,cant)
Clientes(nro_cli, nombre, domicilio, ciudad, País, Ocupación)
Escribir en AR y SQL la siguiente consulta:
Listar la descripción de los artículos que no fueron facturados a ningún cliente al que se le facturó más de 2(DOS) veces algún artículo de color rojo.

π descripcion (σ nro_art not in(π nro_cli (FACTURA)(CLIENTE)(DETALLE)(ARTICULOS))

SELECT A.descripcion FROM Artículos A, Factura F, Detalle D, Clientes C WHERE A.nro_art=D.nro_art AND D.nro_fact=F.nro_fact AND F.nro_cli=C.nro_cli AND nro_art NOT IN (SELECT nro_cli FROM Factura);