**Consultas nivel básico**

**Ejercicio 3.1**

SELECT DISTINCT ciudad FROM clientes;

**Ejercicio 3.2**

SELECT COUNT(CodigoCliente) AS Num\_de\_Clientes

FROM Clientes;

**Ejercicio 3.3**

SELECT Nombre, CantidadEnStock AS Cantidad\_en\_almacen, PrecioProveedor AS Precio\_proveedor

FROM Productos

WHERE Gama = ‘Herramientas’

ORDER BY Nombre;

**Ejercicio 3.4**

SELECT Nombre, (CantidadEnStock \* PrecioProveedor) AS Importe

FROM Productos

WHERE Gama = ‘Herramientas’

ORDER BY CantidadEnStock \* PrecioProveedor, Nombre;

**Ejercicio 3.5**

SELECT Nombre, (PrecioVenta - PrecioProveedor) AS Beneficio

FROM Productos

WHERE Gama = ‘Herramientas’;

**Ejercicio 3.6**

SELECT MAX(PrecioVenta - PrecioProveedor) AS Beneficio

FROM Productos

WHERE CantidadEnStock IS NOT NULL;

**Ejercicio 3.7**

SELECT CodigoPedido AS Pedido, FechaPedido, CodigoCliente As Cliente, FechaEsperada

FROM Pedidos

WHERE FechaEntrega > FechaEsperada

ORDER BY FechaPedido;

**Ejercicio 3.8**

SELECT COUNT(\*) AS Num\_Pedidos

FROM Pedidos

WHERE CodigoCliente = 30;

**Ejercicio 3.9**

SELECT CodigoPedido AS Pedido

FROM Pedidos

WHERE FechaEntrega IS NULL AND Estado = ‘Entregado’

ORDER BY FechaPedido;

**Ejercicio 3.10**

SELECT NumeroLinea, CodigoProducto, Cantidad, PrecioUnidad

FROM DetallePedidos

WHERE CodigoPedido = 10

ORDER BY NumeroLinea;

**Consultas Nivel Medio**

**Ejercicio 2.1**

SELECT Ciudad, COUNT(CodigoCliente) AS Num\_de\_Clientes

FROM Clientes

GROUP BY Ciudad

ORDER BY 2 DESC;

**Ejercicio 2.2**

SELECT Ciudad, COUNT(CodigoCliente) AS Num\_de\_Clientes

FROM Clientes

WHERE Pais = ‘España’

GROUP BY Ciudad

ORDER BY Ciudad;

**Ejercicio 2.3**

SELECT Ciudad, COUNT(\*) AS Num\_de\_Clientes

FROM Clientes

WHERE Pais = ‘España’

GROUP BY Ciudad

HAVING COUNT(\*) > 1

ORDER BY 2 DESC;

**Ejercicio 2.4**

SELECT Gama, MAX(PrecioVenta - PrecioProveedor) AS Beneficio\_maximo

FROM Productos

WHERE CantidadEnStock IS NOT NULL

GROUP BY Gama

ORDER BY 2 DESC;

**Ejercicio 2.5**

SELECT CodigoCliente, COUNT(\*)

FROM Pedidos

GROUP BY CodigoCliente

ORDER BY COUNT(\*) DESC;

**Ejercicio 2.6**

SELECT CodigoCliente, COUNT(\*) AS Pedidos\_Rechazados

FROM Pedidos

WHERE Estado = ‘Rechazado’

GROUP BY CodigoCliente

ORDER BY 2 DESC;

**Ejercicio 2.7**

SELECT SUM(Cantidad \* PrecioUnidad) AS Importe\_Total

FROM DetallePedidos

WHERE CodigoPedido = 10;

**Ejercicio 2.8**

SELECT CodigoProducto, MAX(Cantidad) AS Cantidad\_Maxima

FROM DetallePedidos

GROUP BY CodigoProducto

HAVING MAX(Cantidad) >= 100

ORDER BY 2;

**Ejercicio 2.9**

SELECT CodigoProducto, SUM(Cantidad \* PrecioUidad) AS Importe\_total

FROM DetallePedidos

GROUP BY CodigoProducto

HAVING SUM(Cantidad \* PrecioUnidad) BETWEEN 800 AND 1000

ORDER BY 2;

**Ejercicio 2.10**

SELECT CodigoProducto, SUM(Cantidad \* PrecioUidad) AS Importe\_total

FROM DetallePedidos

WHERE PrecioUnidad BETWEEN 50 AND 100

GROUP BY CodigoProducto

HAVING SUM(Cantidad \* PrecioUnidad) BETWEEN 800 AND 1000

ORDER BY 1;

**Ejercicio 2.11**

SELECT C.CodigoCliente, C.NombreCliente, P.CodigoPedido

FROM Clientes C, Empleados E, Pedidos P

WHERE C.CodigoCliente = P.CodigoCliente AND E.CodigoEmpleado = C.CodigoEmpleadoRepVentas AND E.Nombre = ‘Emmanuel’;

**Ejercicio 2.12**

SELECT E.Nombre AS Empleado, COUNT(P.CodigoPedido) AS Num\_Pedidos

FROM empleados E, Clientes C, Pedidos P

WHERE E.CodigoEmpleado = C.CodigoEmpleadoRepVentas AND C.CodigoCliente = P.CodigoCliente

GROUP BY E.Nombre

ORDER BY 2;

**Ejercicio 2.13**

SELECT Nombre AS Producto, (PrecioVenta-PrecioProveedor) AS Beneficio

FROM Productos

WHERE CantidadEnStock IS NOT NULL

GROUP BY Nombre

HAVING (PrecioVenta - PrecioProveedor) >= ALL (SELECT (PrecioVenta-PrecioProveedor)

FROM Productos

WHERE CantidadEnStock IS NOT NULL

GROUP BY Nombre);

**Consultas de Nivel Avanzado**

**Ejercicio 2.1**

SELECT C.Nombre

FROM Clientes C

WHERE C.CodigoCliente IN (SELECT DISTINCT P.CodigoCliente

FROM Pedidos P)

ORDER BY C.Nombre;

**Ejercicio 2.2**

SELECT CodigoPedido AS Pedido, SUM(Cantidad \* PrecioUnidad) AS Importe\_Total

FROM DetallePedidos

WHERE CodigoPedido BETWEEN 100 AND 110

GORUP BY CodigoPedido

ORDER BY CodigoPedido;

**Ejercicio 2.3**

SELECT P.CodigoPedido, SUM(D.Cantidad \* D.PrecioUnidad) AS Importe\_Total

FROM Pedidos P, DetallePedidos D, Clientes C

WHERE D.CodigoPedido = P.CodigoPedido AND P.CodigoCliente = C.CodigoCliente

AND C.NombreCliente = ‘Beragua’

GROUP BY P.CodigoPedido

ORDER BY P.CodigoPedido;

**Ejercicio 2.4**

SELECT C.NombreCliente AS Cliente, SUM(D.Cantidad \* D.PrecioUnidad) AS Importe\_Total

FROM Clientes C, Pedidos P, DetallePedidos D

WHERE C.CodigoCliente = P.CodigoCliente AND P.CodigoPedido = D.CodigoPedido

GROUP BY C.CodigoCliente, C.NombreCliente

ORDER BT C.NombreCliente;

**Ejercicio 2.5**

SELECT C.NombreCliente AS Cliente, D.CodigoPedido AS Pedido, SUM(D.Cantidad \* D.PrecioUnidad) AS Base\_imponible, SUM(D.Cantidad \* D.PrecioUnidad)\*0.21 AS IVA, SUM(D.Cantidad \* D.PrecioUnidad) \* 1.21 AS Total

FROM Clientes C, Pedidos P, DetallePedidos D

WHERE C.CodigoCliente = P.CodigoCliente AND P.CodigoPedido = D.CodigoPedido

AND D.CodigoPedido IN (100, 103, 106, 109)

GROUP BY C.Nombrecliente, D.CodigoPedido;

**Ejercicio 2.6**

SELECT P.Nombre AS Producto, SUM(D.Cantidad) AS Unidades\_Pedidas

FROM Productos P, DetallePedidos D

WHERE P.CodigoProducto = D.CodigoProducto

GROUP BY P.Nombre

HAVING SUM(D.Cantidad) > 450

ORDER BY 3 DESC;

**Ejercicio 2.7**

SELECT P1.Nombre AS Producto, P1.PrecioVenta AS Precio

FROM Productos P1

WHERE P1.PrecioVenta = (SELECT MAX(P2.PrecioVenta)

FROM Productos P2)

**Ejercicio 2.8**

SELECT MAX(Total.Importe) AS Importe\_maximo, MIN(Total.Importe) AS Importe\_minimo

FROM (SELECT SUM(Cantidad \* PrecioUnidad) AS Importe

FROM DetallePedidos

GROUP BY D.CodigoPedido) Total;

**Extensiones en MySQL**

**Ejercicio 1.1**

DROP FUNCTION IF EXISTS fRanking\_por\_precio;

DELIMTIER $$

CREATE FUNCTION fRanking\_por\_precio(

iProdID INTEGER)

RETURNS INTEGER

NOT DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE posicion INTEGER;

SELECT COUNT(\*) INTO posicion

FROM productos P1

WHERE P1.precio > (SELECT P2.precio FROM productos P2

WHERE prodID = iProdID);

RETURN posicion+1;

END$$

DELIMITER ;

CREATE OR REPLACE VIEW vi\_ranking

AS SELECT prodID, nombre, precio, fRanking\_por\_recio(prodID) AS Posicion\_por\_precio

FROM productos

ORDER BY 4;

**Ejercicio 1.2**

DROP PROCEDURE IF EXISTS pRanking\_por\_precio;

DELIMITER $$

CREATE PROCEDURE pRanking\_por\_precio (

IN pNombre VARCHAR(20))

BEGIN

DECLARE cantidad INTEGER;

DECLARE posicion INTEGER;

SELECT COUNT(\*) INTO cantidad

FROM productos

WHERE nombre = pNombre;

CASE

WHEN cantidad = 0 THEN

SIGNAL SQLSTATE ‘45000’ SET MESSAGE\_TEXT = ‘No existen productos con ese nombre’;

WHEN cantidad > 1 THEN

SIGNAL SQLSTATE ‘45000’ SET MESSAGE\_TEXT = ‘Existe mas de 1 producto con ese nombre’;

WHEN cantidad = 1 THEN

SELECT COUNT(\*) INTO posicion

FROM productos P1

WHERE P1.precio > (SELECT P2.precio

FROM productos P2

WHERE P2.nombre = pNombre);

SELECT posición+1 AS posición\_por\_precio;

END CASE;

END$$

DELIMITER ;

**Ejercicio 1.3**

DROP PROCEDURE IF EXISTS pBlackFriday;

DELIMITER $$

CREATE PROCEDURE pBlackFriday (

IN accion VARCHAR(3))

BEGIN

CASE accion

WHEN ‘ON’ THEN

UPDATE gaming

SET precio\_copia = precio, precio = precio + (precio\*0.15), estado = ‘en oferta’

WHERE estado = ‘disponible’;

SELECT \* FROM gaming

WHERE estado = ‘en oferta’;

WHEN ‘OFF’ THEN

UPDATE gaming

SET precio = precio\_copia, estado = ‘disponible

WHERE estado = ‘en oferta;

SELECT \* FROM gaming

WHERE estado = ‘disponible;

END CASE;

END$$

DELIMTIER ;

**Ejercicio 1.4**

Función 1

DROP FUNCTION IF EXISTS fRanking\_coste;

DELIMITER $$

CREATE FUNCTION fRanking\_coste (

fprodID INTEGER))

RETURNS INTEGER

NOT DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE posicion INTEGER;

SELECT COUNT(\*) INTO posicion

FROM productos P1

WHERE P1.coste > (SELECT P2.coste

FROM productos P2

WHERE prodID = fprodID);

RETURN posicion+1;

END$$

DELIMITER ;

Función 2

DROP FUNCTION IF EXISTS fRanking\_beneficio;

DELIMITER $$

CREATE FUNCTION fRanking\_beneficio (

fProdID INTEGER)

RETURNS INTEGER

NOT DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE posicion INTEGER;

SELECT COUNT(\*) INTO posicion

FROM productos P1

WHERE (p1.precio - P1.coste) > (SELECT (P2.precio - P2.coste)

FROM productos P2

WHERE P2.prodID = fProdID);

RETURN posición+1;

END$$

DELIMITER ;

Vista

CREATE OR REPLACE VIEW vi\_ranking\_completo AS

SELECT prodID, nombre,

CONCAT(precio, ‘ (ranking: ‘, fRanking\_por\_precio(prodID), ‘)’) AS Precio

CONCAT(coste, ‘ (ranking: ‘, fRanking\_coste(prodID), ‘)’) AS Coste

CONCAT(precio-coste, ‘ (ranking: ‘, fRanking\_beneficio(prodID), ‘)’) AS Beneficio

FROM productos

ORDER BY precio-coste DESC;

**Ejercicio 1.5**

Función coste menor

DROP FUNCTION IF EXISTS fRanking\_coste\_menor;

DELIMITER $$

CREATE FUNCTION fRanking\_coste\_menor (

fprodID INTEGER))

RETURNS INTEGER

NOT DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE posicion INTEGER;

SELECT COUNT(\*) INTO posicion

FROM productos P1

WHERE P1.coste <= (SELECT P2.coste

FROM productos P2

WHERE prodID = fprodID);

RETURN posicion;

END$$

DELIMITER ;

DROP PROCEDURE IF EXISTS pInforme;

DELIMITER $$

CREATE PROCEDURE pInforme()

BEGIN

SELECT prodID, nombre AS ‘Producto de precio maximo’, precio

FROM productos

WHERE fRanking\_por\_precio(prodID) = 1;

SELECT prodID, nombre AS ‘Producto de precio maximo’, precio

FROM productos

WHERE fRanking\_coste\_menor(prodID) = 1;

SELECT prodID, nombre AS ‘Producto de precio maximo’, precio

FROM productos

WHERE fRanking\_beneficio(prodID) = 1;

END$$

DELIMITER ;

**Ejercicio 1.6**

DROP FUNCTION IF EXISTS fDiaSemana;

DELIMITER $$

CREATE FUNCTION fDiaSemana (

numero INTEGER)

RETURNS VARCHAR(20)

DETERMINISTIC

NO SQL

BEGIN

DECLARE diaSemana VARCHAR(20);

CASE numero

WHEN 1 THEN SET diaSemana = ‘Lunes’;

WHEN 2 THEN SET diaSemana = ‘Martes’;

WHEN 3 THEN SET diaSemana = ‘Miercoles’;

WHEN 4 THEN SET diaSemana = ‘Jueves’;

WHEN 5 THEN SET diaSemana = ‘Viernes’;

WHEN 6 THEN SET diaSemana = ‘Sabado’;

WHEN 7 THEN SET diaSemana = ‘Domingo’;

ELSE THEN

SET diaSemana = ‘Dia incorrecto!’;

END CASE;

RETURN diaSemana;

END$$

DELIMITER ;

**Ejercicio 1.7**

DROP FUNCTION IF EXISTS fCadena\_sin\_tildes;

DELIMITER $$

CREATE FUNCTION fCadena\_sin\_tildes (

entrada VARCHAR(1000))

RETURNS VARCHAR(1000)

DETERMINISTIC

NO SQL

BEGIN

DECLARE cadenaSinTildes VARCHAR(1000);

SET cadenaSinTildes = entrada;

SET cadenaSinTildes = REPLACE(entrada, ‘á’, ‘a’);

SET cadenaSinTildes = REPLACE(entrada, ‘Á’, ‘A’);

Repetir para el resto de vocales…

RETURN cadenaSinTildes;

END$$

DELIMITER ;

**Ejercicio 2.1**

Trigger 1

DROP TRIGGER IF EXISTS tCheck\_nota\_before\_insert;

DELIMITER $$

CREATE TRIGGER tCheck\_nota\_before\_insert

BEFORE INSERT ON alumnado

FOR EACH ROW

BEGIN

IF (NEW.nota < 0) THEN

SET NEW.nota = 0;

END IF;

IF (NEW.nota > 10) THEN

SET NEW.nota = 10;

END IF;

END$$

DELIMITER ;

Trigger 2

DROP TRIGGER IF EXISTS tCheck\_nota\_before\_update;

DELIMITER $$

CREATE TRIGGER tCheck\_nota\_before\_update

BEFORE UPDATE ON alumnado

FOR EACH ROW

BEGIN

IF (NEW.nota < 0) THEN

SET NEW.nota = 0;

END IF;

IF (NEW.nota > 10) THEN

SET NEW.nota = 10;

END IF;

END$$

DELIMITER ;

**Ejercicio 2.2**

DROP FUNCTION IF EXISTS fFormato\_email;

DELIMITER $$

CREATE FUNCTION fFormato\_email (

cadenaTexto VARHCAR(1000))

RETURNS VARCHAR(1000)

DETERMINISTIC

NO SQL

BEGIN

DECLARE cadenaFinal VARCHAR(1000);

SET cadenaFinal = LOWER(REPLACE(cadenaTexto, ‘ ‘, ‘’));

RETURN cadenaFinal;

END$$

DELIMITER ;

**Ejercicio 2.3**

DROP PROCEDURE IF EXISTS pCrear\_email;

DELIMITER $$

CREATE PROCEDURE pCrear\_email (

IN nombre VARCHAR(50),

IN apellido1 VARCHAR(50),

IN apellido2 VARCHAR(50),

IN dominio VARCHAR(20),

OUT email VARCHAR(200))

BEGIN

SET email = CONCAT(fFormato\_email(nombre), LEFT(fFormato\_email(apellido1), 3), LEFT(fFormato\_email(apellido2), 3), dominio);

RETURN email;

END$$

DELIMITER ;

**Ejercicio 2.4**

DROP TRIGGER IF EXISTS tCrear\_email\_before\_insert;

DELIMITER $$

CREATE TRIGGER tCrear\_email\_before\_insert

BEFORE INSERT ON alumnado

FOR EACH ROW

BEGIN

DECLARE nuevoCorreo VARCHAR(200);

If (NEW.email IS NULL) THEN

CALL pCrear\_email(NEW.nombre, NEW.apellido1, NEW.apellido2, ‘@alumno.com’) INTO nuevoCorreo;

SET NEW.email = nuevoCorreo;

END IF;

END$$

DELIMITER ;

**Ejercicio 2.5**

DROP TRIGGER IF EXISTS tGuardar\_email\_after\_update;

DELIMITER $$

CREATE TRIGGER tGuardar\_email\_after\_update

AFTER UPDATE ON alumnado

FOR EACH ROW

BEGIN

IF (OLD.email <> NEW.email) THEN

INSERT INTO log\_cambios\_email (ida, old\_email, new\_email) VALUES

(OLD.ida, OLD.email, NEW.email);

END IF;

END$$

DELIMITER ;

**Ejercicio 2.6**

DROP TRIGGER IF EXISTS tGuardar\_alumnado\_after\_delete;

DELIMITER $$

CRETE TRIGGER tGuardar\_alumnado\_after\_delete

AFTER DELETE ON alumnado

FOR EACH ROW

BEGIN

INSERT INTO log\_alumnado\_eliminado (ida, nombre, apellido1, apellido2, email) VALUES

(OLD.ida, OLD.nombre, OLD.apellido1, OLD.apellido2, OLD.email);

END$$

DELIMITER ;

**Ejercicio 2.7**

Trigger 1

DROP TRIGGER IF EXISTS tNo\_updates;

DELIMITER $$

CREATE TRIGGER tNo\_updates

BEFORE UPDATE ON matricula

FOR EACH ROW

BEGIN

SIGNAL SQLSTATE ‘45000’ SET MESSAGE\_TEXT = ‘No se puede actualizar la fila’;

END$$

DELIMITER ;

Trigger 2

DROP TRIGGER IF EXISTS tBefore\_delete;

DELIMITER $$

CREATE TRIGGER tBefore\_delete

BEFORE DELETE ON matricula

BEGIN

IF (

((SELECT COUNT(\*)

FROM matricula

WHERE IDM = OLD.IDM) = 1)

OR

((SELECT COUNT(\*)

FROM matricula

WHERE IDA = OLD.IDA) = 1)

)

THEN

SIGNAL SQLSTATE ‘45000’ SET MESSAGE\_TEXT = ‘No se puede borrar la fila’;

END IF;

END$$

DELIMITER ;