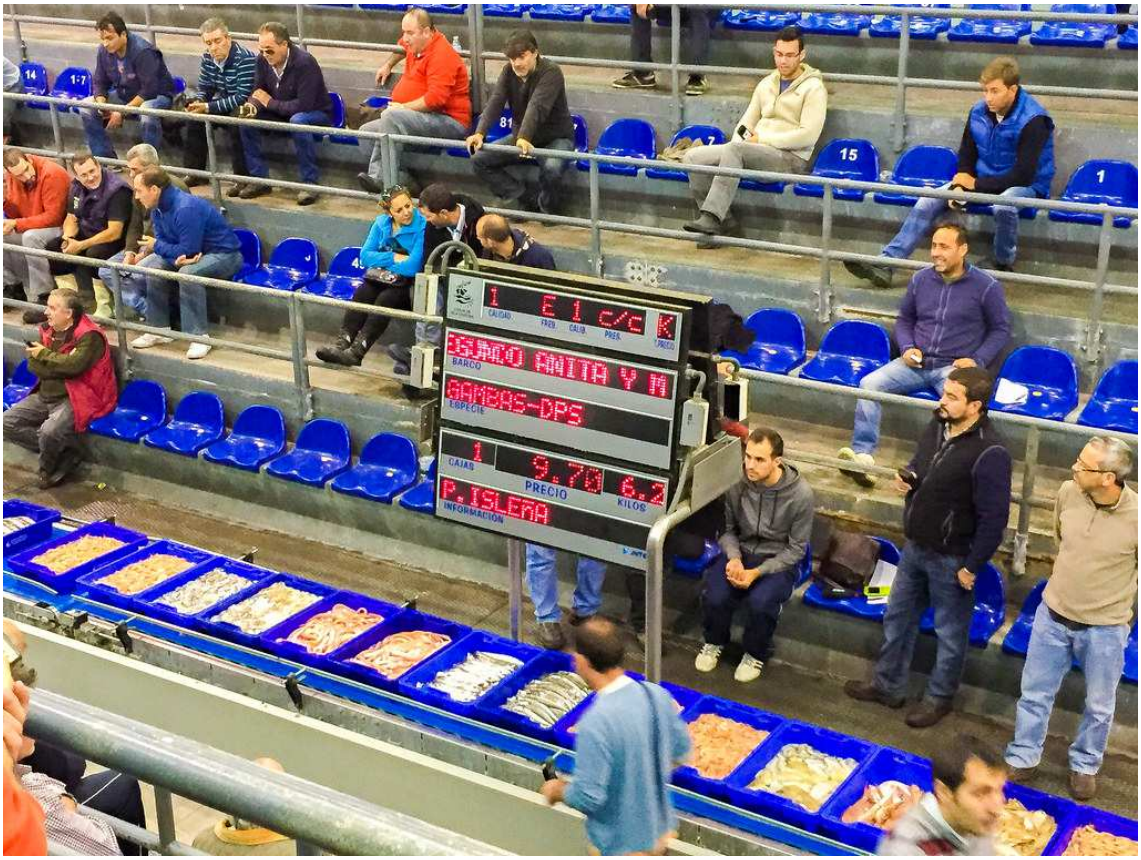
LONJA ALFOS



1ºDAW

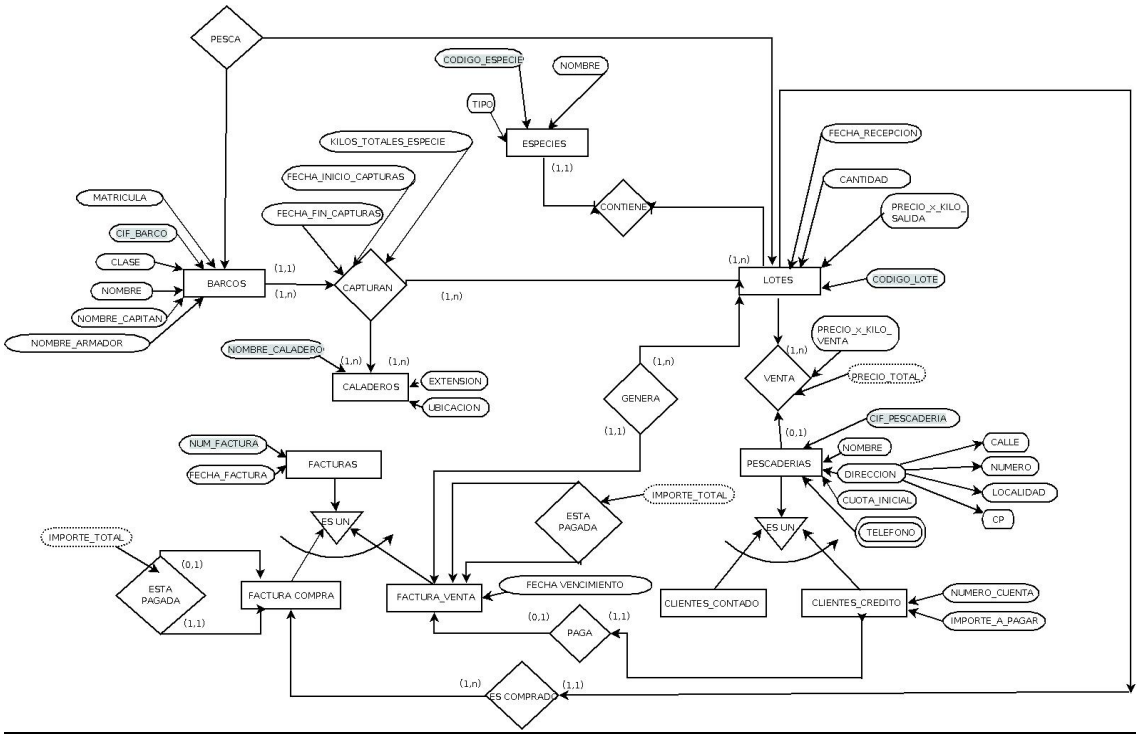
Alfonso Carrascosa
Oscar Delgado

Índice de contenidos.

Descripción del proyecto

Este proyecto trata de desarrollar una base de datos para llevar la gestión y administración de una lonja de pescado de un pueblo costero dado de alta como receptor y vendedor de productos marítimos recién pescados. Los barcos llevan la pesca a la lonja y allí se subasta a los compradores que generalmente son pescaderías de la zona.

Tras la lectura, análisis y comprensión del enunciado, elaboramos este modelo entidad relación que se presenta.



Normalización

1ª Forma Normal.

a) Eliminamos atributos compuestos. Del atributo Dirección salen varios atributos que los vamos a incluir en la entidad de la que depende, Pescaderías.
b) Transformamos el atributo Teléfono multi-evaluado, en otra entidad de tipo entidad débil. Es el atributo de PESCADERIAS.
c) Establecemos las claves primarias únicas de cada entidad y claves foráneas para relacionar de forma correcta las entidades de nuestro modelo.

Con esto hemos eliminado los atributos que tenían valores no atómicos.

2ª Forma Normal.

Todos los atributos no clave dependen completamente de la clave primaria. En otras palabras, no puede haber un subconjunto de la clave primaria que determine un atributo no clave. Cada atributo no clave debe depender de la clave primaria en su totalidad, y no solo de una parte de ella. Las tablas cumplen con esta forma normal.

3ª Forma Normal.

En la entidad BARCO existen dos posibles claves primarias, tanto MATRICULA que identifica inequívocamente a un Barco, como CIF_BARCO, que podría identificar fiscalmente al Barco pero que corresponde al Armador o dueño del barco. El dueño del Barco puede tener además, más de un BARCO, con lo que el atributo NOMBRE_ARMADOR Y CIF habría que considerarla una entidad o relación a parte, en nuestro modelo, donde BARCO dispondría de su clave primaria Matricula.

Barco (Matricula#, CIF, Clase, Nombre, Nombre_Capitan)

Armador (CIF#, Nombre, Apellidos, Telefono)

Una vez hecho esto, no se ven dependencias funcionales transitivas, es decir, todos los atributos no clave dependen únicamente de la clave primaria de la propia tabla, con lo que cumpliría con la esta forma normal y con la forma de Boyce-Codd.

Modelo Relacional

Nos disponemos a realizar la representación de ese Modelo Conceptual o también llamado Modelo de Entidad Relación, al Modelo Relacional. Tenemos en un principio y sin atender a las relaciones y demás consideraciones unas 10 entidades.

Se definen restricciones de integridad que irán encaminadas a establecer el flujo de trabajo con las restricciones de clave primaria, restricciones de clave foránea y cualquier otra restricción necesaria para garantizar la consistencia y la validez de los datos. Se asumen las siguientes consideraciones:

Se crea una entidad llamada **Capturas,** debido a la relación ternaria que existe entre las tres entidades que la conforma. En ella se relacionarán los lotes recibidos, los barcos que los pescaron y los caladeros donde se efectuó la pesca. Recibe como claves foráneas, MATRICULA, CODIGO Y NOBRE_CALADERO. Ellas tres harán de clave principal de la tabla, estableciendo gracias a ello una relación de (n:n) entre todas ellas.

La compra de los lotes por parte de la Lonja a los barcos se debería de hacer inmediatamente, generándose la factura correspondiente y el pago por nuestra parte. Aún así se establece un campo que indica si está pagada o no, en la nueva entidad debido a la relación

“paga” que se generará (**Facturas_Compra_Pagada**). Ella nos ayudará con ese campo pudiendo considerar, que puede que las facturas no se abonen al momento. Se crea un Trigger que recoge el importe de la entidad **Lotes**, para que el campo importe de la entidad **Facturas** tenga el mismo valor.

Consideramos que puede que haya lotes que no se vendan. Con lo que afectará a la creación de una entidad más, Ventas.

Un cliente de contado siempre pagará al contado, pero un cliente a crédito podrá abonar facturas al contado si quiere. Añadimos en la entidad **Facturas**, una forma de pago.

Un lote pertenece a una especie (1:1): agregamos una columna de clave externa CODIGO_ESPECIE a la tabla **Lotes**, que hará referencia al CODIGO_ESPECIE de clave principal de la tabla **Especies**. Esto establecerá una relación uno a uno entre lotes y especies, donde cada lote está asociado con una sola especie.

Un lote se vende a una pescadería (1:1): agregamos una columna de clave externa CODIGO_LOTE a la tabla **Venta**, que hará referencia a la clave principal CODIGO_LOTE de la tabla Lotes. Esto establecerá una relación uno a uno entre Lotes y Ventas, donde cada lote está asociado con una sola venta.

Una pescadería puede comprar varios lotes (1:n): Para implementar esta relación, necesitamos agregar una CIF_PESCADERIA de columna de clave foránea a la tabla **Ventas**, que hará referencia a la CIF_PESCADERIA de clave principal de la tabla **Pescaderías**. Esto establecerá una relación de uno a muchos entre Pescaderías y sus compras, donde cada pescadería puede tener múltiples compras asociadas (ventas para nosotros).

Una pescadería genera varias facturas (1:n): Agregamos los campos necesarios a la tabla **Facturas_Venta**. Añadimos como Foreign Key CODIGO_LOTE de la tabla **Ventas** y NUMERO_FACTURA de la tabla **Facturas**, siendo clave principal de dicha tabla **Facturas**. Esto establecerá una relación de uno a muchos entre Pescaderia y Facturas, donde cada pescadería puede tener varias facturas asociadas. Su Primary Key será el código NUMERO_FACTURA y el CODIGO_LOTE.

Paso a tablas

1-Escribiremos las entidades unas seguidas de otras con sus atributos correspondientes. Se establece antes el paso de las **jerarquías** existentes. En el caso de la entidad PESCADERIAS hemos elegido el paso c, donde la superclase absorbe los atributos individuales de sus subclases. Hay que establecer restricciones check para evitar errores junto con Triggers. En la entidad jerárquica FACTURAS usamos el tipo de paso jeráquico a, donde la superclase ofrece su Primary Key que la recogerán las subclases.

- Armador (CIF#, Nombre, Apellidos, Telefono)

- Barcos (Matricula#, CIF_Amador, Clase, Nombre, Nombre_capitan)
- Caladeros (Nombre#, Extensión, Ubicación)
- Especies (Cod_especie#, Nombre, Tipo)
- Lotes (Cod_lote#, cod_especie, matricula, fecha_recepción, cajas, kilos_totales, precio_kilo_salida)
- Facturas (Num_factura#, Fecha_factura, Tipo)
- Facturas_venta (Num_factura#, Codigo lote#, Fecha VTO, Codigo_lote)
- Facturas_compra (Num_factura#, cif barco#, Importe)
- Pescadería (Cif#, Nombre, Calle, Numero, Localidad, C.P, Cuota_inicial, Pagada, Numero cuenta, Fecha VTO)

2-Escribiremos las relaciones que pasan a ser entidad con sus atributos. La relación *capturan* para a la tabla CAPTURAS, la relación *ventas* pasa a la tabla VENTAS, y las dos relaciones *paga* pasan a las tablas Facturas_Ventas_Pagadas y Facturas_Compra_Pagada.

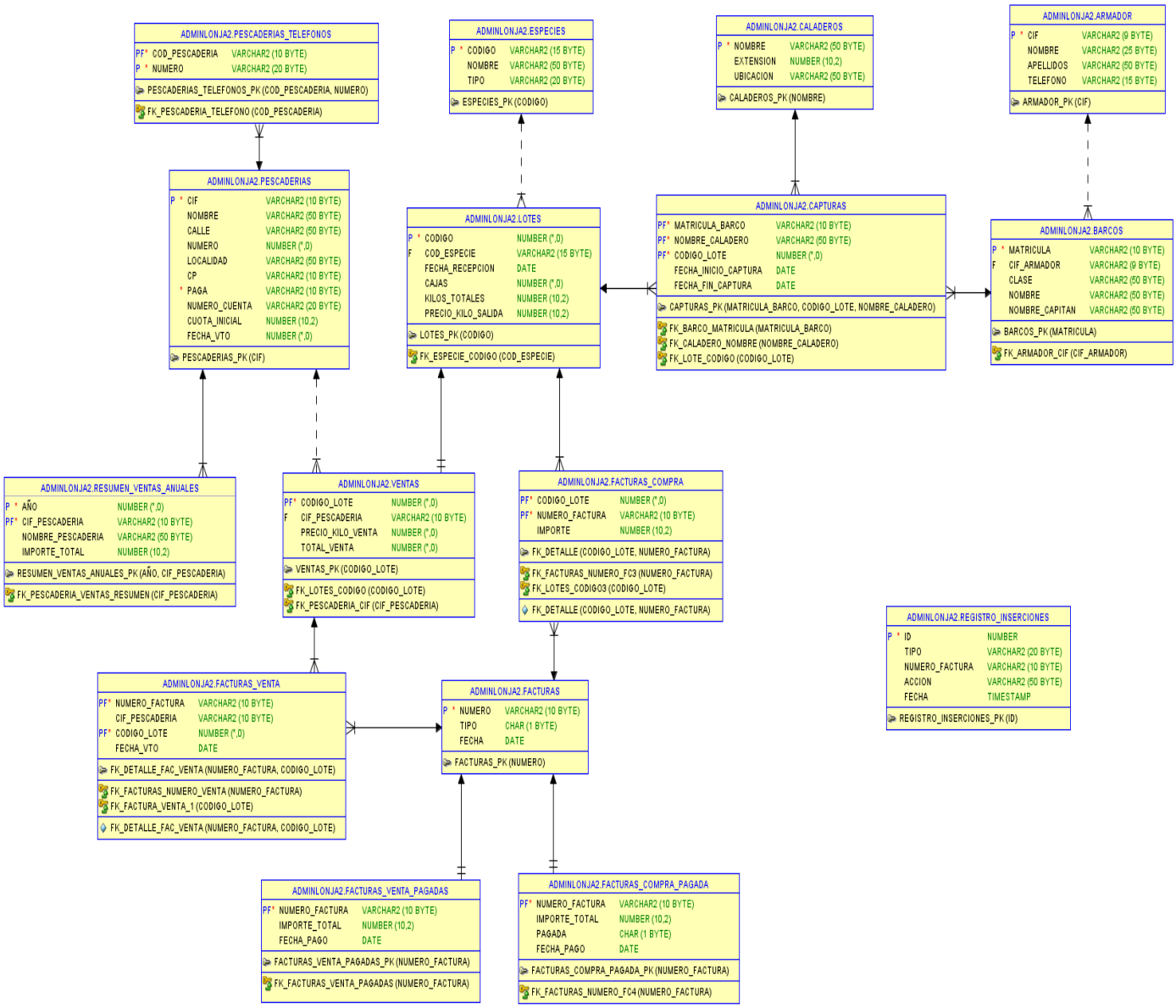
- Capturas (cif barcos#, Cod_especie#, Nombre_caladero#, Fecha_fin, Fecha_Inicio)
- Ventas (Codigo lote#, CIF pescaderia, Precio kilo venta, Total venta)
- Facturas_venta_pagadas (Numero factura#, Importe total, Fecha pago)
- Factura_compra_pagada (Numero factura#, Importe total, Pagada, Fecha pago)

3-Escribiremos las entidades débiles. Se crea una entidad PESCADERIAS_TELEFONOS.

- Pescaderias_Telefonos (Cod pescaderia#, Numero#)

4-Escribiremos los atributos compuestos en la entidad importante. Aquí hemos simplificando el atributo compuesto, incluyendo sus atributos a la entidad Pescaderías.

Una vez establecidas todas estas consideraciones, el modelo relacional gráficamente se quedaría establecido de la siguiente manera.


ADMINLONJA2.PESCADERIAS_TELEFONOS
PF* COD_PESCADERIA VARCHAR2 (10 BYTE)
P * NUMERO VARCHAR2 (20 BYTE)
PESCADERIAS_TELEFONOS_PK (COD_PESCADERIA, NUMERO)
FK_PESCADERIA_TELEFONO (COD_PESCADERIA)
ADMINLONJA2.ESPECIES
P * CODIGO VARCHAR2 (15 BYTE)
NOMBRE VARCHAR2 (60 BYTE)
TIPO VARCHAR2 (20 BYTE)
ESPECIES_PK (CODIGO)
ADMINLONJA2.CALADEROS
P * NOMBRE VARCHAR2 (50 BYTE)
EXTENSION NUMBER (10,2)
UBICACION VARCHAR2 (50 BYTE)
CALADEROS_PK (NOMBRE)
ADMINLONJA2.ARMADOR
P * CIF VARCHAR2 (9 BYTE)
NOMBRE VARCHAR2 (25 BYTE)
APELLIDOS VARCHAR2 (50 BYTE)
TELEFONO VARCHAR2 (15 BYTE)
ARMADOR_PK (CIF)
ADMINLONJA2.PESCADERIAS
P * CIF VARCHAR2 (10 BYTE)
NOMBRE VARCHAR2 (50 BYTE)
CALLE VARCHAR2 (50 BYTE)
NUMERO NUMBER (*,0)
LOCALIDAD VARCHAR2 (50 BYTE)
CP VARCHAR2 (10 BYTE)
* PAGA VARCHAR2 (10 BYTE)
NUMERO_CUENTA VARCHAR2 (20 BYTE)
CUOTA_INICIAL NUMBER (10,2)
FECHA_VTO NUMBER (*,0)
PESCADERIAS_PK (CIF)
ADMINLONJA2.LOTES
P * CODIGO NUMBER (*,0)
F COD_ESPECIE VARCHAR2 (15 BYTE)
FECHA_RECEPCION DATE
CAJAS NUMBER (*,0)
KILOS_TOTALES NUMBER (10,2)
PRECIO_KILO_SALIDA NUMBER (10,2)
LOTES_PK (CODIGO)
FK_ESPECIE_CODIGO (COD_ESPECIE)
ADMINLONJA2.CAPTURAS
PF* MATRICULA_BARCO VARCHAR2 (10 BYTE)
PF* NOMBRE_CALADERO VARCHAR2 (50 BYTE)
PF* CODIGO_LOTE NUMBER (*,0)
FECHA_INICIO_CAPTURA DATE
FECHA_FIN_CAPTURA DATE
CAPTURAS_PK (MATRICULA_BARCO, CODIGO_LOTE, NOMBRE_CALADERO)
FK_BARCO_MATRICULA (MATRICULA_BARCO)
FK_CALADERO_NOMBRE (NOMBRE_CALADERO)
FK_LOTE_CODIGO (CODIGO_LOTE)
ADMINLONJA2.BARCOS
P * MATRICULA VARCHAR2 (10 BYTE)
F CIF_ARMADOR VARCHAR2 (9 BYTE)
CLASE VARCHAR2 (50 BYTE)
NOMBRE VARCHAR2 (50 BYTE)
NOMBRE_CAPITAN VARCHAR2 (50 BYTE)
BARCOS_PK (MATRICULA)
FK_ARMADOR_CIF (CIF_ARMADOR)
ADMINLONJA2.RESUMEN_VENTAS_ANUALES
P * AÑO NUMBER (*,0)
PF* CIF_PESCADERIA VARCHAR2 (10 BYTE)
NOMBRE_PESCADERIA VARCHAR2 (50 BYTE)
IMPORTE_TOTAL NUMBER (10,2)
RESUMEN_VENTAS_ANUALES_PK (AÑO, CIF_PESCADERIA)
FK_PESCADERIA_VENTAS_RESUMEN (CIF_PESCADERIA)
ADMINLONJA2.VENTAS
PF* CODIGO_LOTE NUMBER (*,0)
F CIF_PESCADERIA VARCHAR2 (10 BYTE)
PRECIO_KILO_VENTA NUMBER (*,0)
TOTAL_VENTA NUMBER (*,0)
VENTAS_PK (CODIGO_LOTE)
FK_LOTES_CODIGO (CODIGO_LOTE)
FK_PESCADERIA_CIF (CIF_PESCADERIA)
ADMINLONJA2.FACTURAS_COMPRA
PF* CODIGO_LOTE NUMBER (*,0)
PF* NUMERO_FACTURA VARCHAR2 (10 BYTE)
IMPORTE NUMBER (10,2)
FK_DETALLE (CODIGO_LOTE, NUMERO_FACTURA)
FK_FACTURAS_NUMERO_FC3 (NUMERO_FACTURA)
FK_LOTES_CODIGO3 (CODIGO_LOTE)
FK_DETALLE (CODIGO_LOTE, NUMERO_FACTURA)
ADMINLONJA2.REGISTRO_INSERCIONES
P * ID NUMBER
TIPO VARCHAR2 (20 BYTE)
NUMERO_FACTURA VARCHAR2 (10 BYTE)
ACCION VARCHAR2 (50 BYTE)
FECHA TIMESTAMP
REGISTRO_INSERCIONES_PK (ID)
ADMINLONJA2.FACTURAS_VENTA
PF* NUMERO_FACTURA VARCHAR2 (10 BYTE)
CIF_PESCADERIA VARCHAR2 (10 BYTE)
PF* CODIGO_LOTE NUMBER (*,0)
FECHA_VTO DATE
FK_DETALLE_FAC_VENTA (NUMERO_FACTURA, CODIGO_LOTE)
FK_FACTURAS_NUMERO_VENTA (NUMERO_FACTURA)
FK_FACTURA_VENTA_1 (CODIGO_LOTE)
FK_DETALLE_FAC_VENTA (NUMERO_FACTURA, CODIGO_LOTE)
ADMINLONJA2.FACTURAS
P * NUMERO VARCHAR2 (10 BYTE)
TIPO CHAR (1 BYTE)
FECHA DATE
FACTURAS_PK (NUMERO)
ADMINLONJA2.FACTURAS_VENTA_PAGADAS
PF* NUMERO_FACTURA VARCHAR2 (10 BYTE)
IMPORTE_TOTAL NUMBER (10,2)
FECHA_PAGO DATE
FACTURAS_VENTA_PAGADAS_PK (NUMERO_FACTURA)
FK_FACTURAS_VENTA_PAGADAS (NUMERO_FACTURA)
ADMINLONJA2.FACTURAS_COMPRA_PAGADA
PF* NUMERO_FACTURA VARCHAR2 (10 BYTE)
IMPORTE_TOTAL NUMBER (10,2)
PAGADA CHAR (1 BYTE)
FECHA_PAGO DATE
FACTURAS_COMPRA_PAGADA_PK (NUMERO_FACTURA)
FK_FACTURAS_NUMERO_FC4 (NUMERO_FACTURA)

Instalación

Desde el fichero Lonja.sql se ejecutan todas las sentencias que aparecen.

-Se crea el tablespace, y el usuario Administrador adminLonja con el superusuario, en este caso SYSTEM. (La creación de este usuario está más abajo, en la parte de *creación de usuarios*)

Crear el espacio de tabla

CREATE TABLESPACE Lonja DATAFILE 'C:\Lonja' SIZE 100m AUTOEXTEND ON NEXT 500k MAXSIZE 1000M;

-Creamos la conexión LonjaAdmin, logándonos como adminLonja.

-Creamos todos los objetos: **tablas**, **vistas**, **secuencias, funciones**, **procedimientos**, **triggers** y **jobs**.

-Desde este esquema/usuario Administrador (adminLonja) daremos acceso a los objetos que convenga tanto a los clientes como al trabajador que usarán de la base de datos. Esto se hace un poco más abajo, en la parte de *creación de usuarios*. Hay que crear sus propias conexiones para que se loguen independientemente. Seguimos estando logados como adminLonja.

Desde el fichero LonjaAdmin.sql, haremos lo que sigue a continuación:

-Se crean las tablas, vistas y secuencias, funciones, procedimientos, triggers y el job. El orden es importante, ya que ciertos objetos hacen referencia a otros, y su falta puede ocasiones errores de funcionamiento.

-Se crean los **usuarios** con sus privilegios que se le otorgan.

Se crean 4 usuarios para la simulación del proyecto, cuando todos los objetos a los que dar permisos se han creado. Un administrador con todos los privilegios llamado adminLonja para el manejo de la base de datos. El usuario trabajador realizará las consultas sobre las tablas de la base de datos sobre las que tiene acceso y los usuarios clientes, podrán ejecutar las vistas creadas para ellos.

adminLonja

CREATE USER adminLonja IDENTIFIED BY LonjaajnoL default tablespace Lonja QUOTA UNLIMITED ON Lonja;

Privilegios otorgados.

GRANT DBA adminLonja;

Cliente1

CREATE USER CIF001 IDENTIFIED BY usuCIF001 default tablespace Lonja QUOTA 10M ON Lonja;

Privilegios otorgados.

GRANT CREATE SESSION to CIF001;

GRANT SELECT ON facturasCliente TO CIF001;

Cliente2

CREATE USER CIF002 IDENTIFIED BY usuCIF002 default tablespace Lonja QUOTA 10M ON Lonja;

Privilegios otorgados.

GRANT CREATE SESSION to CIF002;

GRANT SELECT ON facturas_Pendientes_Pago TO CIF002;

GRANT SELECT ON facturasCliente TO CIF002;

GRANT EXECUTE ON Consultar_Facturas_Credito TO CIF002;

Trabajadores

Se crea un rol para unas tablas específicas que serán las que puedan usar los **Trabajadores**. Existen tablas a las que no deben acceder los trabajadores.

```
CREATE ROLE select_tablas1;


GRANT SELECT ON Armador TO select_tablas1;
GRANT SELECT ON Barcos TO select_tablas1;
GRANT SELECT ON Caladeros TO select_tablas1;
GRANT SELECT ON Capturas TO select_tablas1;
GRANT SELECT ON Especies TO select_tablas1;
GRANT SELECT ON Lotes TO select_tablas1;
GRANT SELECT ON Pescaderias TO select_tablas1;
GRANT SELECT ON Pescaderias_Telefonos TO select_tablas1;
GRANT SELECT ON Facturas TO select_tablas1;
GRANT SELECT ON Facturas_Compra TO select_tablas1;
GRANT SELECT ON Facturas_Compra_Pagada TO select_tablas1;
GRANT SELECT ON Ventas TO select_tablas1;
GRANT SELECT ON Facturas_Venta TO select_tablas1;
GRANT SELECT ON Facturas_Venta_Pagadas TO select_tablas1;
```

GRANT SELECT ON Resumen_Ventas_Anuales TO select_tablas1;

trabajador1Lonja

CREATE USER trabajador1Lonja IDENTIFIED BY tra1Lonja default tablespace Lonja QUOTA 100M ON Lonja;

Privilegios otorgados.

GRANT CREATE SESSION to trabajador1Lonja;

GRANT UPDATE any TABLE TO trabajador1Lonja;
GRANT DELETE any TABLE TO trabajador1Lonja;
GRANT INSERT any TABLE TO trabajador1Lonja;

GRANT select_tablas1 TO trabajador1Lonja;
GRANT SELECT ON Historial_Ventas_Por_Año TO trabajador1Lonja;
GRANT EXECUTE ON Actualizar_Ventas_Anuales TO trabajador1Lonja;
GRANT EXECUTE ON Codigo_Lote TO trabajador1Lonja;
GRANT EXECUTE ON Total_Facturas_Pendientes TO trabajador1Lonja;

Una vez hecho esto, se crean las conexiones para que cada usuario pueda entrar a su espacio y que únicamente usen los objetos dados en "propiedad o multipropiedad". Una vez logado el usuario, se deberá teclear,

ALTER SESSION SET CURRENT_SCHEMA = adminLonja;

siendo esta sentencia la que da acceso al uso de los objetos a los que se les concedió permiso.

Desde el fichero Insert.sql se incluyen los insert a las tablas.

Proceso y funcionalidad.

En este apartado se describe el proceso desde que entra un barco en la Lonja hasta que sale el pescado de la misma.

Como se indica en el documento "Gestión de Actividades Pesqueras (nivel 1)" del Ministerio de Fomento relativo a los Puertos del Estado, todo el pescado fresco ha de realizar su primera **venta** (compra para nosotros) en una lonja autorizada por la Comunidad Autónoma donde se circunscriba, a través de las Consejerías de Pesca encargadas del sector, como se recoge en Real Decreto 1822/2009, de 27 de noviembre, por el que se regula la primera venta de los productos pesqueros. En este lugar que recibe la pesca, también se efectuará la

exposición y primera **venta** de los productos pesqueros frescos, así como donde se deben preparar, refrigerar y congelar o depositar los productos pesqueros para la misma finalidad. Se realizará la **compra** de la misma, (venta para nosotros) por los clientes dados de alta en la Lonja como pescaderías autorizadas. Con lo que tanto, vendedores (barcos pesqueros) como compradores (pescaderías) deben de estar dados de alta en el sistema con sus correspondientes identificaciones.

Compra. Cuando un barco entra, deposita la mercancía en la Lonja. Se genera una entrada en la base de datos con los datos de las capturas. Para ello se rellena la tabla **Lotes** con las características de las capturas que ingresan. Se rellenan con el número de lote, código de especie, fecha del día, número de cajas, kilos totales, precio de kilo de salida que será el importe de la nuestra compra. Un código de lote se relaciona con una sola especie. Inmediatamente se realiza la factura de compra. Hay que generar el número de factura en la tabla **Facturas.** Las tablas **Facturas_Compra** y **Facturas_Venta** comparten el número que se crea automáticamente de la siguiente manera. Se hace una inserción en la tabla **Facturas** simplemente indicando la fecha de la factura y la indicación de 'v' o 'c'. Se genera en dicha tabla un número secuencial gracias a un *Trigger* (tr_facturas_numero_1) que se encarga de asignar el indicativo de FC o FV según sea compra o venta. Luego con ese número generado hacemos una inserción manual en **Facturas_Compra** o **Facturas_Venta** con los datos correspondientes.

Vamos a la tabla **Facturas_Compra**. Introducimos el código de lote y el número de factura que se ha generado. El cálculo del importe se calcula automáticamente con un *Trigger* (tr_importe_factura_compras) que será el número de kilos de cada especie por el precio por kilo estipulado. Estos dos datos están en la tabla **Lotes.** Lo normal es que las facturas de compra se paguen en el acto por la Lonja, o sea por nosotros. Aún así, se ha creado otra una nueva tabla **Facturas_Compra_Pagada** por si no se abonan en el acto. Un *Trigger* controla la inserción en esta tabla. Hace un insert de forma automática con el número de factura introducido en **Facturas_Compra** y la suma de los importes de cada lote correspondientes a dicha factura. Se deja el campo Fecha_Pago vacío e indicando una 'n' de forma predeterminada por defecto en el campo Pagada hasta indicar la fecha de pago.

Venta. Los lotes se ponen a disposición de los clientes inmediatamente. Quien gane la subasta adquiere el lote. Los lotes entran y salen de la Lonja el **mismo día,** con lo que la fecha de compra de los lotes a los barcos y la fecha de venta de los lotes a los clientes será la misma y coincidirá. La venta se graba en la tabla **Ventas,** relacionada con la tabla **Pescadería** y **Lotes,** incluyendo el precio total de venta, gracias a un *Trigger* (tr_importe_factura_venta) que consulta la cantidad de kilos en **Lotes** y lo multiplica por el precio de venta de la propia tabla **Ventas**.

Como hicimos en la compra, creamos un número de factura en la tabla **Facturas** introduciendo la fecha y el indicativo `v` de venta. Insertamos el número generado en la tabla **Facturas_Venta**, junto con el CIF de la pescadería y el lote que está adquiriendo. La tabla Facturas_Ventas tiene un campo con la fecha de vencimiento de la factura. La fecha de

vencimiento se calcula con otro *Trigger* (tr_actualizar_fecha_vto) donde suma el dato de los días del campo de fecha de vencimiento obtenido de la tabla **Pescadería** a la fecha de factura.

Una vez registrada en **Factura_Venta**, hay un *Trigger* (tr_calcular_importe_total_v) que crea un registro en la tabla **Facturas_Venta_Pagadas**, donde se insertan el número de factura, el importe total de la venta, proveniente de la tabla **Ventas**, y la fecha de pago por parte del cliente en la que se ha saldado definitivamente la factura. No confundir con la fecha de vto en los clientes a crédito, ya que un cliente puede o no abonar la factura en su fecha. Si la factura de venta es generada por un cliente al contado, la fecha se recupera de la fecha de la tabla Facturas, anteriormente creada, ya que al no tener crédito, se **debe** de saldar obligatoriamente el mismo día de la venta. Si la factura es generada por un cliente de crédito, la fecha se dejará en "null" hasta que se haya realizado el pago.

Para hacer el seguimiento de los insert de compras y ventas, se deja este cuadro con los input y output de nuestra lonja:

Flujo de compras y ventas

COMPRAS				VENTAS			
m1	100	30/04/2024	FC1000	CIF001	100	FV1001	30/04/2024
m1	101	30/04/2024	FC1000	CIF001	101	FV1001	30/04/2024
m2	102	01/05/2024	FC1002	CIF002	102	FV1003	01/05/2024
m2	103	01/05/2024	FC1002				
m2	104	01/05/2024	FC1002	CIF003	103	FV1004	01/05/2024
				CIF003	104	FV1004	01/05/2024
m4	105	02/05/2024	FC1005	CIF001	105	FV1006	02/05/2024
m4	106	02/05/2024	FC1005				
m4	107	02/05/2024	FC1005	CIF002	106	FV1007	02/05/2024
				CIF002	107	FV1007	02/05/2004
m3	108	03/05/2024	FC1008	CIF004	108	FV1009	03/05/2024
m3	109	03/05/2024	FC1008				
m3	110	03/05/2024	FC1008	CIF003	109	FV1010	03/05/2024
				CIF003	110	FV1010	03/05/2024

Lotes

Columnas | Datos | Model | Restricciones | Permisos | Estadísticas | Disparadores | Flashback | Dependencias | Detalles | Particiones | Índi

Ordenar... | Filtrar:

	CODIGO	COD_ESPECIE	FECHA_RECEPCION	CAJAS	KILOS_TOTALES	PRECIO_KILO_SALIDA
1	100	2	30/04/24	50	100,5	25
2	101	1	30/04/24	20	25,5	50
3	102	1	01/05/24	30	10	75
4	103	1	01/05/24	30	10	45
5	104	1	01/05/24	60	20	60
6	105	3	02/05/24	15	100	60
7	106	4	02/05/24	30	50	50
8	107	5	02/05/24	40	40	70
9	108	6	02/05/24	20	20	100
10	109	7	02/05/24	10	50,5	120
11	110	7	02/05/24	45	25,5	100

Ventas

	CODIGO_LOTE	CIF_PESCADERIA	PRECIO_KILO_VENTA	TOTAL_VENTA
1	100	CIF001	40	4020
2	101	CIF001	80	2040
3	102	CIF002	120	1200
4	103	CIF003	60	600
5	104	CIF003	90	1800
6	105	CIF001	95	9500
7	106	CIF002	75	3750
8	107	CIF002	100	4000
9	108	CIF004	120	2400
10	109	CIF003	150	7575
11	110	CIF003	130	3315

Trigger mutante

Origen

1- CREATE OR REPLACE TRIGGER tr_factura_compras_pago

```
AFTER INSERT ON Facturas_Compra
FOR EACH ROW
BEGIN
        insert into facturas_compra_pagada (numero_factura) values (:new.numero_factura);
END;
```

2- CREATE OR REPLACE TRIGGER tr_calcular_importe_total_c

```
BEFORE INSERT ON Facturas_Compra_Pagada
FOR EACH ROW

 DECLARE
     v_importe_total DECIMAL(10,2);

 BEGIN

         SELECT SUM(Importe) INTO v_importe_total FROM Facturas_Compra WHERE
        Numero_Factura = :NEW.Numero_Factura;
```

```
        :NEW.Importe_Total := v_importe_total;
END;
```

Explicación

Como vemos, cuando se inserta en la tabla **Facturas_Compra** se activa el *Trigger* número 1. Este hace intenta realizar un insert en la tabla **Facturas_Compra_Pagada.** Esta útima tabla activa otro *Trigger* , que hace que el flujo del primer *Trigger* no haya finalizado. Este nuevo *Trigger* que se activa, consulta a la tabla **Facturas_Compra,** intentando calcular el importe total sumando los importes de las facturas con el mismo número en dicha tabla. Al ser la tabla que activó el primer *Trigger* y cuyos datos no están confirmados porque el flujo no acabó dará un error de mutación. Para dejarlo más claro el *Trigger* tr_calcular_importe_total_c, realiza una consulta a `Facturas_Compra`, mientras esta misma tabla está siendo modificada **por la inserción que inició el proceso**.

En definitiva, un *Trigger* no puede acceder a leer o modificar datos que están siendo usados por otro *Trigger*. Al no poder realizar acciones **commit** por su propia naturaleza, los datos se quedan en un limbo, sin existir para otros y no pueden darlos por hechos hasta que el Trigger termine, después de acabar todos sus flujos.

Oracle detecta este acceso concurrente (lectura y escritura simultánea) en la misma transacción, lo que provoca un error de mutación, ya que la tabla Facturas_Compra no puede ser consultada de esta manera en un trigger de fila cuando está siendo modificada. Oracle garantiza que los datos no se encuentren en un estado intermedio o incoherente. Es decir, un *Trigger* no puede leer de una tabla si esa tabla está siendo modificada por la instrucción que activó el *Trigger*.

Como ya se ha explicado el *Trigger before* **antes** de insertar en la tabla **facturas_compra** recojo el importe que será el cálculo de kilos de especie por el precio de salida de la tabla lotes. Una vez que teníamos este dato, el *Trigger after* **después** de insertar en la tabla **facturas_compra**, manejará datos que a su vez se deberían de insertar en la tabla **Facturas_Compra_Pagada.**

Una **solución** es la de usar jobs que son trabajos que pueden ejecutarse indicando una temporalidad en su ejecución.

Procedemos de esta forma:

-Se crea una tabla de apoyo

Hemos tenido que crear una tabla nueva, **Registro_Inserciones**, en ella se va a grabar, un id que vendrá dado por una secuencia automática, y luego los datos enviados, que será el Tipo, coincidiendo con el dato 'Facturas_Compra' o cualquier dato que se establezca. Será siempre el mismo para reutilizar la tabla diferenciando el tipo de movimiento grabado en la tabla. Otro campo que será el número de factura. Otro campo será la acción que desembocará el jobs en nuestro caso 'ENABLE_VERIFICAR_JOB' y por último, algo muy útil, es indicar una fecha con todos los datos necesario para fiscalizar luego los flujos de trabajo.

```
CREATE TABLE Registro_Inserciones (
        ID NUMBER PRIMARY KEY,
        Tipo VARCHAR2(20),
        Numero_Factura VARCHAR2(10),
        accion VARCHAR2(50),
        fecha TIMESTAMP
);
```

-Se modifica el Trigger after.

```
CREATE OR REPLACE TRIGGER tr_factura_compras_pago
AFTER INSERT ON Facturas_Compra
FOR EACH ROW
BEGIN

  Insertar_Registro_Insercion('Facturas_Compra', :NEW.Numero_Factura);

END;
```

Se ha eliminado la lógica de inserción en la tabla y simplemente se ha llevado un par de datos a un procedimiento que hará lo siguiente:

-Se crea un procedimiento

Se crea Insertar_Registro_Insercion, donde se le pasa el dato de número de factura y una referencia a lo que es en este caso se decide llamarlo 'Facturas_Compra'. Este procedimiento hace una grabación del registro en **Registro_Inserciones,** por cada fila que el *Trigger* le vaya enviando.

```
CREATE OR REPLACE PROCEDURE Insertar_Registro_Insercion (
        p_Tipo in registro_inserciones.tipo%type,
        p_Numero_Factura IN registro_inserciones.numero_factura%type)
AS
  v_contador int;
BEGIN

  SELECT COUNT(*) INTO v_contador FROM Registro_Inserciones
  WHERE Numero_Factura = p_Numero_Factura AND Tipo = p_Tipo;

  IF v_contador = 0 THEN

        INSERT INTO Registro_Inserciones (ID, Tipo, Numero_Factura, Accion,
        Fecha) VALUES (SEQ_REGISTRO_INS.NEXTVAL, p_Tipo,
        p_Numero_Factura, 'ENABLE_VERIFICAR_JOB', SYSTIMESTAMP);

  END IF;

END;
```

(nota: para recoger el dato del número de la factura, necesitábamos que sólo apareciera una vez el número de factura. Como se generaba una línea por lote, aparecían más de una vez ese

número de factura en la tabla Registros_Inserciones, con lo que cuando se activaba el job, nos daba violación de constraint por clave primaria en la tabla **Facturas_Compra_Pagada.** De ahí que contemos las veces que aparece (count(*)) en la tabla **Registro_Inserciones**, si no aparece ninguna, hacemos el insert.

Una vez tenemos la tabla de apoyo, creada para poder salvar la mutación, necesitamos un job, que desencadene de forma automática la grabación definitiva en **Facturas_Compra_Pagada.**

-Se crea el Job o programador

```
BEGIN
        DBMS_SCHEDULER.CREATE_JOB (
                job_name       => 'VERIFICAR_JOB',
                job_type       => 'PLSQL_BLOCK',
                job_action     => 'BEGIN
                          FOR linea IN (SELECT * FROM Registro_Inserciones WHERE
                          accion = ''ENABLE_VERIFICAR_JOB'')
                     LOOP
                       Procesar_Registro_Inserciones;
                       DELETE FROM Registro_Inserciones WHERE id = linea.id;

                     END LOOP;
                       END;',
                start_date     => SYSTIMESTAMP,
                repeat_interval => 'FREQ=SECONDLY; INTERVAL=20',
                enabled        => TRUE
    );
 END;
```

Este Job se ejecuta cada 20 segundos comprobando si encuentra algún registro en la tabla **Registro_Inserciones** donde la acción sea la que le pusimos 'ENABLE_VERIFICAR_JOB'. Si existe algún registro, según esta lógica, llamará al procedimiento Procesar_Registro_Inserciones. Tenemos este procedimiento, donde se buscará en el campo Tipo de la tabla Registro_Inserciones que coincida con 'Facturas_Compra', pero podemos incluir otras llamada a otros procedimientos que tengamos desde este mismo Jobs reutilizable, filtrando por aquel campo.

-Se crea el procedimiento que desencadena la inserción.

```
CREATE OR REPLACE PROCEDURE Procesar_Registro_Inserciones AS
          v_Numero_Factura Registro_Inserciones.Numero_Factura%TYPE;
          v_Tipo Registro_Inserciones.Tipo%TYPE;
          CURSOR c_registro IS SELECT tipo,numero_factura FROM
        Registro_Inserciones;
BEGIN
   OPEN c_registro;
```

```
    LOOP
      FETCH c_registro INTO v_Tipo, v_Numero_Factura;
      EXIT WHEN c_registro%NOTFOUND;

          IF v_Tipo = 'Facturas_Compra' THEN
            INSERT INTO Facturas_Compra_Pagada (Numero_Factura) VALUES
      (v_Numero_Factura);
          END IF;

          DELETE FROM Registro_Inserciones WHERE Tipo = v_Tipo AND
          Numero_Factura = v_Numero_Factura;

     END LOOP;

    CLOSE c_registro;

  END;
```

Una vez aquí, después de salvar la mutación, ya se podrá hacer de forma automática la inserción en la tabla **Facturas_Compra_Pagada**. Se insertará el número de factura que queríamos haciendo uso del *Trigger* que daba problemas, tr_calcular_importe_total_c que calculará el importe total de la factura. El resto de los dos campos de esta tabla, son insertados por defecto según nuestra lógica de negocio.

Después de que se haya hecho esta inserción, el job o programador, pasa a la siguiente línea de ejecución que será la de borra de la tabla **Registro_Inserciones** el registro con el id correspondiente. Es sólo una tabla de apoyo, y no necesitamos esos datos.

Los jobs tienen como objetivos:

- Mantener la independencia de ejecución con los *Trigger*, eliminando el problema de mutación.
- Permiten la confirmación de datos (commit) del *Trigger* y la ejecución posterior de acciones sobre esos datos de forma automática. Se garantiza la integridad y consistencia de los datos.
- Flexibilidad, pudiendo ser programados para ejecutarse inmediatamente o en otro momento establecido.

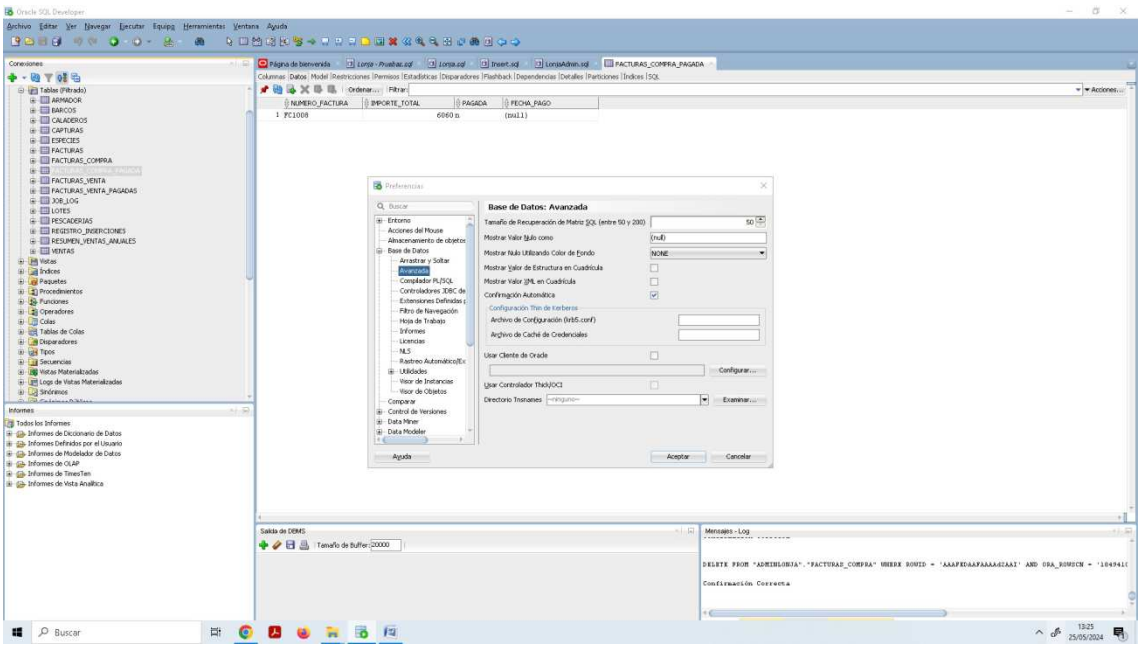
Debemos tener claro que, como otros objetos, son globales en la base de datos y se almacenan en el diccionario de datos de Oracle. Cuando creas un trabajo, se crea un registro en la tabla SYS.SCHEDULER$_JOB que contiene la información del trabajo, incluyendo su nombre, tipo, prioridad, etc. Este registro es único en toda la base de datos, independientemente del esquema o tablespace en el que se creó, pero sólo se ejecutan en el contexto o esquema o usuario en el que fueron creados.

Problemas encontrados

- No se puede habilitar un trabajo directamente desde un *Trigger* en Oracle, ya que puede causar un bloqueo recursivo.

- El paquete dbms_output no muestra la salida cuando se ejecuta dentro de un job programado automáticamente en Oracle. Esto se debe a que los jobs programados se ejecutan en un contexto diferente, donde la salida de dbms_output no se redirige automáticamente a la consola del usuario. Hay que crear una tabla e insertar esos output en campos establecidos.

- Ese contexto propio, nos ha traído de cabeza ya que un job, ya que las operaciones DML (INSERT, UPDATE, DELETE) dentro del job no se confirman hasta que el job se complete con éxito. Lo hemos solucionado, obligando a la base de datos a realizar commit automáticamente:



- Lo podríamos haber hecho de otra forma, deshabilitar y habilitar el job, fuera de su contexto. Si hacemos esto se **fuerza** a realizar un commit como cuando salimos del esquema y nos dice que hay operaciones por confirmar y parece que no es lo más conveniente. Pero lo que queríamos era que fuera lo más automático posible y no tener que depende de nuevo de realizar un bloque anónimo o procedimiento para eso. También se puede hacer simplemente un commit manual en línea de comandos, que parece ser suficiente para que esas transacciones se confirmen. Pinchando en esa casilla evitamos esto, pero hay ciertos problemas al realizar esta acción, se pierde manejo de la base de datos, y pueden confirmarse transacciones que no queríamos que pasaran.

¿Por qué nos hemos complicado la existencia?:

- Para mantener **Facturas_Compra_Pagada** actualizada con los números de factura únicos y sus importes totales.
- Evitar duplicados y mantener la integridad referencial.
- Automatizar el procesamiento de inserciones de facturas con jobs que se gestionan dinámicamente.

Sentencias Útiles

SELECT job_name, log_date, status FROM dba_scheduler_job_run_details WHERE job_name = 'VERIFICAR_JOB' AND OWNER = 'ADMINLONJA' ORDER BY log_date DESC;

SELECT job_name, enabled, state, owner FROM dba_scheduler_jobs WHERE job_name = 'VERIFICAR_JOB';

VISTAS

Vistas comunes a ambos clientes.

CREATE OR REPLACE VIEW facturasCliente as select * from facturas_venta where cif_pescaderia = user;

Vista para el cliente crédito.

CREATE OR REPLACE VIEW Facturas_Pendientes_Pago AS SELECT fv.Numero_Factura, fv.CIF_Pescaderia, fv.Fecha_VTO, fvp.Fecha_Pago FROM Facturas_Venta fv

LEFT JOIN Facturas_Venta_Pagadas fvp ON fv.Numero_Factura = fvp.Numero_Factura

WHERE fvp.Fecha_Pago IS NULL and CIF_Pescaderia = user;

Vista creada para el trabajador de la Lonja.

CREATE OR REPLACE VIEW Historial_Ventas_Por_Año AS SELECT EXTRACT(YEAR FROM f.Fecha) AS Año, p.CIF AS CIF_Pescaderia, p.Nombre AS Nombre_Pescaderia, fv.Numero_Factura, v.Total_Venta, v.codigo_lote, fv.Fecha_VTO

FROM Facturas_Venta fv

JOIN Ventas v ON fv.Codigo_Lote = v.Codigo_Lote

JOIN Pescaderias p ON fv.CIF_Pescaderia = p.CIF

JOIN Facturas f ON f.Numero = fv.Numero_factura

ORDER BY Año, p.CIF, fv.Numero_Factura;

CONSULTAS

Estas consultas las podrán realizar solo los trabajadores.

-- Histórico de ventas con el importe total de ventas por pescadería y cantidad de lotes vendidos a cada una.

SELECT p.Nombre AS Pescaderia, SUM(v.Total_Venta) AS Importe_Total_Ventas, COUNT(v.Codigo_Lote) AS Numero_Lotes FROM Pescaderias p

JOIN Ventas v ON p.CIF = v.CIF_Pescaderia GROUP BY p.Nombre

ORDER BY Importe_Total_Ventas DESC;

-- Histórico de ventas por pescaderías con detalle de pagos

```
SELECT distinct p.Nombre AS
Pescaderia,fv.Numero_Factura,fv.Fecha_VTO,fvp.Importe_Total,fvp.Fecha_Pago FROM
Pescaderias p
JOIN Ventas v ON p.CIF = v.CIF_Pescaderia
JOIN Facturas_Venta fv ON v.Codigo_Lote = fv.Codigo_Lote
LEFT JOIN Facturas_Venta_Pagadas fvp ON fv.Numero_Factura = fvp.Numero_Factura
ORDER BY p.Nombre, fv.Numero_Factura;
```

-- Historico de ventas de lotes a pescaderias

```
SELECT distinct p.Nombre AS
Pescaderia,fv.Numero_Factura,fv.Fecha_VTO,fvp.Importe_Total,fvp.Fecha_Pago, v.codigo_lote
FROM Pescaderias p
JOIN Ventas v ON p.CIF = v.CIF_Pescaderia
JOIN Facturas_Venta fv ON v.Codigo_Lote = fv.Codigo_Lote
LEFT JOIN Facturas_Venta_Pagadas fvp ON fv.Numero_Factura = fvp.Numero_Factura
ORDER BY p.Nombre, fv.Numero_Factura,v.codigo_lote;
```

-- Facturas de venta pendientes de pago por pescadería

```
SELECT distinct fv.CIF_Pescaderia, fv.Numero_Factura, fv.Fecha_VTO FROM Facturas_Venta fv
LEFT JOIN Facturas_Venta_Pagadas fvp ON fv.Numero_Factura = fvp.Numero_Factura
WHERE fvp.Numero_Factura IS NULL;
```

-- Importe total y el número de facturas pendientes de pago por parte de la Lonja

```
SELECT COUNT(Numero_Factura) AS Nº_Fac_Pendientes, Importe_Total AS Saldo_Total_Pendiente FROM Facturas_Compra_Pagada
WHERE Pagada = 'n' GROUP by Numero_Factura,Importe_Total;


-- Historio de capturas realizadas por cada barco en cada caladero.

SELECT c.Matricula_Barco, c.Nombre_Caladero, COUNT(c.Codigo_Lote) AS Numero_Capturas, SUM(l.Kilos_Totales) AS Total_Kilos FROM Capturas c
JOIN Lotes l ON c.Codigo_Lote = l.Codigo GROUP BY c.Matricula_Barco, c.Nombre_Caladero
ORDER BY c.Matricula_Barco, c.Nombre_Caladero;


-- Historico de kilos capturados de cada especie.

SELECT e.Nombre AS Especie, SUM(l.Kilos_Totales) AS Total_Kilos FROM Especies e
JOIN Lotes l ON e.Codigo = l.Cod_especie GROUP BY e.Nombre
ORDER BY Total_Kilos DESC;
```

FUNCIONES

--Función para clientes a crédito. Para el usuario CIF002. Se pide un año por teclado y busca en en las tablas **Facturas_Venta y Pescadería** y nos devuelve un listado de facturas pagadas y no pagadas del cliente.

```
create or replace FUNCTION Consultar_Facturas_Credito(año IN int) RETURN VARCHAR2
AS
  v_Numero_Factura facturas_venta.numero_factura%type;
  v_CIF_Pescaderia facturas_venta.cif_pescaderia%type;
  v_Fecha_VTO facturas_venta.fecha_vto%type;
  existe int := 0;

  CURSOR factura_cursor IS SELECT fv.Numero_Factura, fv.CIF_Pescaderia, fv.Fecha_VTO
  FROM Facturas_Venta fv
    JOIN Pescaderias p ON fv.CIF_Pescaderia = p.CIF
    WHERE p.PAGA = 'credito' AND fv.CIF_Pescaderia = user AND EXTRACT(YEAR FROM fv.Fecha_VTO) = año;

BEGIN
  OPEN factura_cursor;
  LOOP
    FETCH factura_cursor INTO v_Numero_Factura, v_CIF_Pescaderia, v_Fecha_VTO;
    EXIT WHEN factura_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Numero Factura: ' || v_Numero_Factura || ', CIF Pescaderia: ' || v_CIF_Pescaderia || ', Fecha VTO: ' || v_Fecha_VTO);
    existe := existe + 1;
  END LOOP;
  CLOSE factura_cursor;

  IF existe = 0 THEN
    RETURN 'No se encontraron facturas a credito para la fecha proporcionada.';
  ELSE
```

```
    RETURN 'Consulta completada.';
  END IF;
END;
```

--Función para trabajador1Lonja. Función que le das el código de lote y te da la pescadería que lo ha adquirido. Consulta útil por motivos sanitarios.

```
CREATE OR REPLACE FUNCTION Codigo_Lote (codigo ventas.codigo_lote%type) return varchar2
AS
  v_pescaderia pescaderias.nombre%type;

BEGIN
  SELECT p.nombre into v_pescaderia FROM Pescaderias p JOIN Ventas v ON p.cif = v.cif_pescaderia where codigo_lote = codigo;

  return v_pescaderia;
end;
```

PROCEDIMIENTOS

--Procedimiento descrito en la sección Trigger Mutante. Es llamado para realizar una inserción en una tabla auxiliar y evitar problemas de mutación.

```
CREATE OR REPLACE PROCEDURE Insertar_Registro_Insercion
(p_Tipo in registro_inserciones.tipo%type,
p_Numero_Factura IN registro_inserciones.numero_factura%type)
AS
  v_contador int;
BEGIN

  SELECT COUNT(*) INTO v_contador FROM Registro_Inserciones
  WHERE Numero_Factura = p_Numero_Factura AND Tipo = p_Tipo;

  IF v_contador = 0 THEN
    INSERT INTO Registro_Inserciones (ID, Tipo, Numero_Factura)
    VALUES (SEQ_REGISTRO_INS.NEXTVAL, p_Tipo, p_Numero_Factura);
  END IF;
END;
```

--Procedimiento descrito en la sección Trigger Mutante. Es llamado por un jobs, para hacer la inserción en la tabla **Facturas_Compra_Pagada** y salvar la mutación.

```
CREATE OR REPLACE PROCEDURE Procesar_Registro_Inserciones AS
  v_Numero_Factura Registro_Inserciones.Numero_Factura%TYPE;
```

```
    v_Tipo Registro_Inserciones.Tipo%TYPE;
    CURSOR c_registro IS SELECT * FROM Registro_Inserciones;
BEGIN
    OPEN c_registro;
    LOOP
        FETCH c_registro INTO v_Tipo, v_Numero_Factura;
        EXIT WHEN c_registro%NOTFOUND;

        IF v_Tipo = 'Facturas_Compra' THEN
            INSERT INTO Facturas_Compra_Pagada (Numero_Factura) VALUES (v_Numero_Factura);
        END IF;

        DELETE FROM Registro_Inserciones WHERE Tipo = v_Tipo AND Numero_Factura = v_Numero_Factura;

    END LOOP;
    CLOSE c_registro;
END;
```

--Procedimiento que solo lo pueden ejecutar trabajadores de la Lonja que muestra la facturación por cliente anualmente. Sirve para el modelo 347 del propio cliente.

```
CREATE OR REPLACE PROCEDURE Actualizar_Ventas_Anuales AS

    v_Año Resumen_Ventas_Anuales.Año%TYPE;
    v_CIF_Pescaderia Resumen_Ventas_Anuales.CIF_Pescaderia%TYPE;
    v_Nombre_Pescaderia Resumen_Ventas_Anuales.Nombre_Pescaderia%TYPE;
    v_Importe_Total Resumen_Ventas_Anuales.Importe_Total%TYPE;

    CURSOR c_ventas_anuales IS SELECT EXTRACT(YEAR FROM f.Fecha) AS Año, fv.CIF_Pescaderia, p.Nombre AS Nombre_Pescaderia, SUM(v.Total_Venta) AS Importe_Total FROM Facturas_Venta fv
        JOIN Facturas f on f.numero = fv.numero_factura
        JOIN Ventas v ON fv.Codigo_Lote = v.Codigo_Lote
        JOIN Pescaderias p ON fv.CIF_Pescaderia = p.CIF
        GROUP BY EXTRACT(YEAR FROM f.Fecha), fv.CIF_Pescaderia, p.Nombre;

BEGIN

    DELETE FROM Resumen_Ventas_Anuales;

    OPEN c_ventas_anuales;

    LOOP
        FETCH c_ventas_anuales INTO v_Año, v_CIF_Pescaderia, v_Nombre_Pescaderia, v_Importe_Total;
        EXIT WHEN c_ventas_anuales%NOTFOUND;

        INSERT INTO Resumen_Ventas_Anuales (Año, CIF_Pescaderia, Nombre_Pescaderia, Importe_Total)
        VALUES (v_Año, v_CIF_Pescaderia, v_Nombre_Pescaderia, v_Importe_Total);
```

```
    END LOOP;
    CLOSE c_ventas_anuales;

END;
--Procedimiento de borrado del proyecto.


CREATE OR REPLACE PROCEDURE Borrar_Proyecto AS
BEGIN
    -- Eliminamos las secuencias
    EXECUTE IMMEDIATE 'DROP SEQUENCE sec_numero_factura';
    EXECUTE IMMEDIATE 'DROP SEQUENCE SEQ_REGISTRO_INS';


       --Eliminación de vistas
    EXECUTE IMMEDIATE 'DROP VIEW FACTURASCLIENTE';
    EXECUTE IMMEDIATE 'DROP VIEW FACTURAS_PENDIENTES_PAGO';
    EXECUTE IMMEDIATE 'DROP VIEW HISTORIAL_VENTAS_POR_AÑO';


    --Eliminación de procedimientos
    EXECUTE IMMEDIATE 'DROP PROCEDURE Insertar_Registro_Insercion';
    EXECUTE IMMEDIATE 'DROP PROCEDURE Procesar_Registro_Inserciones';
    EXECUTE IMMEDIATE 'DROP PROCEDURE Actualizar_Ventas_Anuales';


       -- Eliminamos las tablas.
    EXECUTE IMMEDIATE 'DROP TABLE Resumen_Ventas_Anuales';
    EXECUTE IMMEDIATE 'DROP TABLE Facturas_Venta_Pagadas';
    EXECUTE IMMEDIATE 'DROP TABLE Facturas_Venta';
    EXECUTE IMMEDIATE 'DROP TABLE Ventas';
    EXECUTE IMMEDIATE 'DROP TABLE Facturas_Compra_Pagada';
    EXECUTE IMMEDIATE 'DROP TABLE Facturas_Compra';
    EXECUTE IMMEDIATE 'DROP TABLE Facturas';
    EXECUTE IMMEDIATE 'DROP TABLE Capturas';
    EXECUTE IMMEDIATE 'DROP TABLE Lotes';
    EXECUTE IMMEDIATE 'DROP TABLE Especies';
    EXECUTE IMMEDIATE 'DROP TABLE Caladeros';
    EXECUTE IMMEDIATE 'DROP TABLE Barcos';
    EXECUTE IMMEDIATE 'DROP TABLE Armador';
    EXECUTE IMMEDIATE 'DROP TABLE Pescaderias_Telefonos';
    EXECUTE IMMEDIATE 'DROP TABLE Pescaderias';
    EXECUTE IMMEDIATE 'DROP TABLE Registro_Inserciones';



   -- Eliminación de usuarios.
    EXECUTE IMMEDIATE 'DROP USER CIF001 cascade';
    EXECUTE IMMEDIATE 'DROP USER CIF002 cascade';
    EXECUTE IMMEDIATE 'DROP USER ADMINLONJA cascade';
    EXECUTE IMMEDIATE 'DROP USER TRABAJADOR1LONJA cascade';
```

```
  -- Eliminamos el rol creado.
   EXECUTE IMMEDIATE 'DROP ROLE ' || select_all_tables;

END;
```

TRIGGERS

```
--Todos descritos en la sección Proceso y funcionalidad

CREATE OR REPLACE TRIGGER tr_facturas_numero_1
BEFORE INSERT ON Facturas
FOR EACH ROW

BEGIN

   IF :NEW.Tipo = 'c' THEN
      SELECT 'FC' || sec_numero_factura.NEXTVAL INTO :NEW.Numero FROM dual;

   ELSIF :NEW.Tipo = 'v' THEN
      SELECT 'FV' || sec_numero_factura.NEXTVAL INTO :NEW.Numero FROM dual;

   END IF;
END;



CREATE OR REPLACE TRIGGER tr_importe_factura_compras
BEFORE INSERT ON Facturas_Compra
FOR EACH ROW

BEGIN

        SELECT l.Kilos_totales * l.Precio_Kilo_Salida INTO :NEW.Importe FROM Lotes l WHERE
        l.Codigo = :NEW.Codigo_Lote;

END;



CREATE OR REPLACE TRIGGER tr_importe_factura_venta
BEFORE INSERT ON Ventas
FOR EACH ROW

BEGIN

        SELECT l.Kilos_totales * :NEW.Precio_Kilo_Venta INTO :NEW.Total_venta FROM Lotes l
        WHERE l.Codigo = :NEW.Codigo_Lote;
END;
```

```
CREATE OR REPLACE TRIGGER tr_actualizar_fecha_vto
BEFORE INSERT ON Facturas_Venta
FOR EACH ROW

  DECLARE
    v_dias_vto INT;
  BEGIN
        SELECT Fecha_VTO INTO v_dias_vto FROM Pescaderias WHERE CIF =
        :NEW.CIF_Pescaderia;

        :NEW.Fecha_VTO := SYSDATE + v_dias_vto;
END;
CREATE OR REPLACE TRIGGER tr_calcular_importe_total_v
BEFORE INSERT ON Facturas_Venta_Pagadas
FOR EACH ROW

  DECLARE

    v_importe_total DECIMAL(10,2);
    v_cif varchar2(10);
    v_fecha DATE:=NULL;
    v_paga VARCHAR2(10);

  BEGIN

    SELECT cif_pescaderia INTO v_cif FROM facturas_venta WHERE numero_factura =
:new.numero_factura AND ROWNUM = 1;

    SELECT SUM(v.Total_Venta) INTO v_importe_total FROM Ventas v WHERE v.Codigo_Lote
IN (
      SELECT fv.Codigo_Lote
      FROM Facturas_Venta fv
      WHERE fv.Numero_Factura = :NEW.Numero_Factura
    );

    :NEW.Importe_Total := v_importe_total;

    SELECT paga INTO v_paga FROM pescaderias WHERE cif = v_cif;

    IF v_paga = 'contado' THEN
      SELECT fecha INTO v_fecha FROM facturas WHERE numero = :new.numero_factura;
    END IF;

    :NEW.Fecha_pago := v_fecha;

END;



CREATE OR REPLACE TRIGGER tr_calcular_importe_total_c
BEFORE INSERT ON Facturas_Compra_Pagada
FOR EACH ROW
```

```
  DECLARE
    v_importe_total DECIMAL(10,2);

  BEGIN

    SELECT SUM(Importe) INTO v_importe_total FROM Facturas_Compra WHERE
Numero_Factura = :NEW.Numero_Factura;

    :NEW.Importe_Total := v_importe_total;

END;
CREATE OR REPLACE TRIGGER tr_factura_compras_pago
AFTER INSERT ON Facturas_Compra
FOR EACH ROW
BEGIN

  Insertar_Registro_Insercion('Facturas_Compra', :NEW.Numero_Factura);

END;
```

JOBS

```
-- Crear un job programado para verificar la tabla de registro y habilitar el job

BEGIN
    DBMS_SCHEDULER.CREATE_JOB (
      job_name       => 'VERIFICAR_JOB',
      job_type       => 'PLSQL_BLOCK',
      job_action     => 'BEGIN
                FOR linea IN (SELECT * FROM Registro_Inserciones WHERE accion =
                ''ENABLE_VERIFICAR_JOB'')
                    LOOP
                      Procesar_Registro_Inserciones;
                      DELETE FROM Registro_Inserciones WHERE id = linea.id;

                    END LOOP;
                END;',
      start_date     => SYSTIMESTAMP,
      repeat_interval => 'FREQ=SECONDLY; INTERVAL=20',
      enabled        => TRUE
    );
  END;
```