DELPHI Y ORACLE

Diseño, programación e integración de sistemas de datos

Manual práctico de estudio

Primera entrega: introducción y capítulos 1-3

InformatOLI

Septiembre de 2026

Introducción

Este libro se plantea como un manual práctico de estudio sobre el desarrollo de aplicaciones de datos con Delphi y Oracle. El objetivo no se limita a aprender instrucciones aisladas de SQL, procedimientos PL/SQL o componentes de acceso a datos. Se estudia el recorrido completo de una operación: desde la necesidad funcional y el modelo relacional hasta la validación, la transacción, la concurrencia y, en las partes posteriores, la integración con una aplicación Delphi mediante FireDAC.

El hilo conductor es Atlas, un sistema didáctico de productos, ventas y existencias. El caso permite mantener un mismo contexto mientras se incorporan conceptos nuevos. Así, una regla sencilla -por ejemplo, impedir que el stock resulte negativo- se analiza primero como requisito, después como restricción de datos, más tarde como operación transaccional y finalmente como llamada desde la aplicación. Esta continuidad evita estudiar cada tecnología como un bloque independiente.

La metodología distingue tres niveles: comprender un concepto, implementarlo y demostrar que funciona. Una consulta que se ejecuta sin errores no se considera suficiente si no responde a la pregunta correcta; del mismo modo, una operación de escritura no se considera fiable si no se comprueba qué ocurre ante un fallo o ante dos usuarios que modifican el mismo dato. Oracle documenta la transacción como una unidad lógica de trabajo y PL/SQL como una extensión procedimental de SQL; esos principios sirven de base para razonar sobre atomicidad, excepciones y concurrencia (Oracle, s. f.-q; Oracle, s. f.-v).

El libro utiliza Oracle Database 19c como referencia principal para SQL y PL/SQL. En la integración de escritorio se emplea Delphi VCL con FireDAC, tecnología que se aborda de forma específica en la cuarta parte. Los ejemplos se mantienen deliberadamente pequeños para que cada resultado se pueda explicar, reproducir y verificar. No se pretende simular una plataforma empresarial completa, sino construir una base técnica transferible a proyectos reales.

Esta primera entrega reúne la introducción y los tres primeros capítulos. El primero organiza las responsabilidades de un sistema de datos; el segundo diseña el modelo relacional y las consultas SQL; el tercero incorpora PL/SQL, transacciones y concurrencia para proteger las reglas cuando varios procesos trabajan sobre la misma información. La estructura completa se plantea en cinco partes y continúa posteriormente con la conexión Delphi-Oracle y con la integración orientada a explotación analítica.

Cómo se organiza el libro

Parte	Pregunta que resuelve	Resultado de estudio
1. Arquitectura y fundamentos	¿Cómo se organiza un sistema completo?	Se asigna una responsabilidad a cada pieza.
2. Modelo relacional y SQL	¿Cómo se representa y consulta la información?	Se diseña un esquema y se interpretan sus consultas.
3. PL/SQL, transacciones y concurrencia	¿Cómo se mantienen las reglas con varios usuarios?	Se razona sobre atomicidad, validación y conflictos.

Parte	Pregunta que resuelve	Resultado de estudio
4. Aplicaciones Delphi con Oracle	¿Cómo se conecta la interfaz con la base de datos?	Se organiza una operación desde la pantalla hasta COMMIT.
5. Integración, Data Warehouse y operación	¿Cómo se convierten los datos en información fiable?	Se diseña una carga verificable y un modelo analítico.

Cómo se utiliza este manual

- Se lee cada apartado intentando anticipar el resultado de los ejemplos antes de ejecutarlos.
- Se distingue entre una regla de negocio, una validación de interfaz y una restricción de persistencia.
- Se documenta qué se espera que ocurra y qué evidencia demuestra que la operación funciona.
- Se utilizan los fragmentos de código como laboratorio: primero se comprende su propósito y después se adapta al entorno concreto.

Parte 1. Arquitectura y fundamentos de un sistema de datos

1.1. El punto de partida: describir una operación real

Antes de elegir componentes o escribir una consulta, se describe lo que ocurre en el negocio. En Atlas, una persona selecciona un cliente, añade productos y confirma una venta. El sistema conserva el documento, registra sus líneas y descuenta las unidades correspondientes. Otra persona consulta después las ventas del día.

Esa descripción contiene decisiones que la interfaz por sí sola no resuelve. ¿Se permite vender sin existencias? ¿Un precio admite cero? ¿Una venta confirmada permite cambiar sus líneas? ¿Una devolución elimina el registro original o crea otra operación? ¿La cantidad siempre representa unidades enteras?

Para mantener el ejemplo concreto, Atlas adopta estas reglas: las cantidades representan unidades enteras positivas; el precio admite cero, pero no valores negativos; una venta confirmada conserva sus condiciones económicas; las devoluciones se representan mediante operaciones específicas; y la confirmación exige existencias suficientes. Son decisiones del caso de estudio, no reglas universales de cualquier aplicación comercial.

La primera forma de documentación consiste en escribir esas reglas como frases comprobables. «La venta funciona bien» resulta ambiguo. «Una venta de tres unidades reduce las existencias de diez a siete» permite verificar una conducta. La precisión del requisito determina la precisión de la prueba.

1.2. Dato, información y significado

El número 120 no explica por sí solo qué representa. Puede significar unidades, euros, segundos o registros. Un sistema útil conserva también el contexto: nombre del campo, unidad de medida, fecha, origen y regla de cálculo.

En Atlas, cantidad = 3 significa tres unidades de un producto dentro de una línea de venta. precio_unitario = 20 expresa el precio aplicado a cada unidad en esa operación. El importe de la línea es 60, según la regla didáctica que no incluye impuestos ni descuentos.

El precio actual del catálogo y el precio aplicado en una venta tienen nombres parecidos, pero representan hechos diferentes. El primero describe una condición vigente. El segundo conserva una condición de la operación. Recalcular una venta histórica con el precio actual cambia su significado.

Se propone un pequeño diccionario de datos antes de programar:

Campo	Significado en Atlas	Regla
codigo_producto	Identificador comercial del artículo	Es único dentro del catálogo.
precio_actual	Precio vigente de catálogo	Admite cero y dos decimales.
precio_unitario	Precio aplicado en una línea	Se conserva al confirmar la venta.
cantidad	Unidades de la línea	Es un entero positivo.

Campo	Significado en Atlas	Regla
fecha_venta	Momento de la operación comercial	Se distingue del momento de importación.

1.3. Aplicación, base de datos y servidor

La aplicación es el programa con el que interactúa el usuario. La base de datos conserva estructuras y datos. El servidor ejecuta servicios que atienden las solicitudes. Aunque un laboratorio aloja todo en un mismo ordenador, las responsabilidades siguen siendo distintas.

Oracle organiza objetos mediante esquemas. Una tabla contiene filas y columnas; otros objetos, como índices y vistas, cumplen funciones diferentes. El esquema representa una agrupación lógica de objetos, no una carpeta ordinaria que se abre con el explorador de archivos (Oracle, s. f.-u).

Cuando Delphi consulta un producto, envía una instrucción a través de una conexión. Oracle la procesa y devuelve un resultado. La pantalla interpreta ese resultado y lo muestra. Escribir un texto en un control no equivale a almacenarlo: entre ambos momentos intervienen validaciones, ejecución y confirmación.

Para investigar un fallo, Atlas distingue estas preguntas: ¿se produce el evento del botón?, ¿la aplicación utiliza el código correcto?, ¿la consulta se ejecuta?, ¿Oracle devuelve filas?, ¿se confirma la modificación?, ¿la pantalla refresca el resultado? Esta separación evita atribuir cualquier problema a «la base de datos».

1.4. Capas: organizar las responsabilidades

Se propone una arquitectura con cuatro responsabilidades. La **presentación** reúne formularios, controles y mensajes. La **lógica de aplicación** coordina operaciones. El **acceso a datos** gestiona consultas y parámetros. La **persistencia y las reglas centrales** residen en tablas, restricciones y procedimientos Oracle.

La separación es una decisión de diseño del manual. No exige cuatro servidores, cuatro ejecutables ni una estructura compleja de clases. En un ejercicio pequeño, un formulario y un módulo de datos ya permiten establecer límites útiles.

Responsabilidad	Pregunta que responde	Ejemplo de Atlas
Presentación	¿Qué necesita ver o introducir la persona?	Código, precio y mensaje de validación.
Aplicación	¿Qué pasos forman la acción solicitada?	Validar, ejecutar, confirmar y actualizar la vista.
Acceso a datos	¿Cómo se expresa la solicitud a Oracle?	Consulta parametrizada y llamada a PL/SQL.
Persistencia	¿Qué condiciones conservan los datos válidos?	Clave única, precio no negativo y control de versión.

Un error de organización aparece cuando el evento de un botón acumula toda la lógica: lee controles, construye SQL, calcula importes, abre conexiones y captura cualquier excepción. En Atlas se propone que ese evento coordine pocas acciones identificables. Así, una corrección de la regla del precio no exige recorrer todos los formularios que lo modifican.

1.5. Dos capas y tres capas

En el diseño de dos capas del laboratorio, Delphi se comunica directamente con Oracle. En una variante de tres capas, Delphi se comunica con un servicio y el servicio accede a Oracle. Se utiliza esta comparación como decisión arquitectónica, no como clasificación de calidad.

Aspecto del diseño	Acceso directo	Acceso mediante servicio
Recorrido	Delphi y Oracle.	Delphi, servicio y Oracle.
Ubicación propuesta de credenciales de base de datos	Cliente gestionado.	Servicio.
Contrato que utiliza el cliente	Consultas o procedimientos autorizados.	Operaciones expuestas por el servicio.
Elementos operativos que se mantienen	Aplicación, conectividad y base de datos.	También disponibilidad y versiones del servicio.

Atlas comienza con acceso directo para estudiar los conceptos. La decisión no implica exponer Oracle públicamente ni distribuir una cuenta administrativa. El entorno determina los mecanismos de autenticación, red y permisos. La aplicación necesita solo las operaciones que utiliza.

Un servicio también requiere reglas claras. Una operación como «confirmar venta» expresa una intención completa. Exponer numerosas operaciones diminutas que dejan al cliente coordinar estados intermedios traslada complejidad al exterior. En el ejercicio, la confirmación conserva una única responsabilidad identificable.

1.6. Conexión, sesión y transacción

Estos términos se relacionan, pero describen elementos diferentes. La conexión permite la comunicación del cliente con Oracle. La sesión representa el contexto de interacción con la base de datos. La transacción agrupa cambios como una unidad lógica (Oracle, s. f.-v).

Atlas puede mantener una conexión abierta mientras el usuario consulta varios productos. Eso no significa que deba mantener una transacción de escritura abierta durante toda la sesión de trabajo. La modificación comienza cerca del momento de guardar y termina con confirmación o deshacer.

Se evita una confusión frecuente: el botón «Guardar» no constituye una transacción por su nombre. Su implementación debe coordinar las instrucciones que pertenecen a esa acción. Si guarda la cabecera y falla al guardar las líneas, se necesita un límite de transacción que permita deshacer ambas cosas.

La parte 3 desarrolla esa unidad y muestra por qué la interacción de varios usuarios modifica el diseño, aunque cada instrucción parezca correcta cuando se prueba por separado.

1.7. Datos operativos y datos analíticos

Una consulta operativa localiza una venta o comprueba existencias. Una consulta analítica agrupa información para responder preguntas sobre periodos, productos o delegaciones. Un Data Warehouse se orienta al análisis y reúne habitualmente información histórica de varios orígenes (Oracle, s. f.-j).

En Atlas, el responsable del almacén necesita comprobar un producto ahora. La dirección necesita comparar meses. El segundo uso introduce definiciones adicionales: qué fecha cuenta, cómo se tratan devoluciones y cuándo se actualizan los datos. Una tabla que conserva ventas no garantiza por sí sola un indicador coherente.

La separación entre ambos usos se introduce de manera gradual. El laboratorio no necesita un almacén analítico completo para consultar diez ventas; sí necesita comprender qué problema resuelve antes de añadirlo.

1.8. Requisitos de calidad que cambian el diseño

Para Atlas se proponen cinco criterios medibles: exactitud, respuesta, recuperación, trazabilidad y mantenibilidad. La exactitud exige que cantidades e importes coincidan con las reglas. La respuesta se mide con un volumen y una operación concretos. La recuperación describe qué ocurre tras un fallo. La trazabilidad permite explicar un resultado. La mantenibilidad permite localizar y modificar una responsabilidad.

No se establece un tiempo «rápido» universal. Se define, por ejemplo, una consulta por código con un catálogo de prueba y se registra su duración. Si la aplicación tarda, se distingue el tiempo de Oracle, el transporte de datos y la actualización de la pantalla. Esta medición forma parte del método del ejercicio.

También se documenta un caso de fallo: se interrumpe una operación antes de confirmar. La expectativa no se limita a mostrar un mensaje; se comprueba que los datos no quedan a medio modificar.

1.9. Ejercicio de razonamiento

Se plantea una venta con dos líneas. La primera dispone de existencias y la segunda no. El lector identifica qué controles orientan al usuario, qué regla verifica Oracle y qué cambios deben deshacerse si la confirmación no resulta válida.

La respuesta propuesta asigna los mensajes a la interfaz, la coordinación a la operación de venta y la protección de los datos a la base de datos. La venta completa se rechaza según las reglas de Atlas. No se conserva la primera línea como una venta parcial sin una decisión explícita del modelo.

La evidencia de comprensión consiste en explicar el recorrido de esa operación sin comenzar por nombres de componentes: intención, datos, reglas, ejecución y resultado. Con ese mapa, la estructura relacional de la parte 2 adquiere un propósito concreto.

Parte 2. Modelo relacional y SQL: representar la información sin ambigüedades

2.1. Diseñar una tabla empieza por definir una fila

La pregunta principal no es cuántas columnas necesita una tabla, sino qué representa cada fila. En Atlas, una fila de PRODUCTOS representa un artículo del catálogo. Una fila de VENTAS representa una operación. Una fila de LINEAS_VENTA representa un producto incluido en esa operación.

Mezclar esos significados produce dificultades. Una tabla con columnas producto1, producto2 y producto3 limita artificialmente el número de artículos. Una tabla que repite todos los datos del cliente en cada línea dificulta mantener su información. El diseño propuesto separa entidades y establece relaciones entre ellas.

Se utilizan dos niveles de identificación: un identificador técnico, como id_producto, y un código comercial, como TEC-001. El primero permite relaciones estables; el segundo conserva un significado para la organización. Que ambos sean únicos no significa que sean intercambiables. En Atlas, un cambio del código comercial no obliga a cambiar todas las referencias internas.

2.2. Claves y restricciones

La clave primaria identifica una fila y no admite valores nulos. Una restricción UNIQUE protege una combinación que debe ser única. Una clave foránea exige una referencia válida según sus reglas. NOT NULL exige presencia y CHECK expresa una condición sobre los valores (Oracle, s. f.-d).

Para Atlas, esas herramientas concretan decisiones: no existen dos productos con el mismo código; cada línea pertenece a una venta; una línea referencia un producto del catálogo; el precio no es negativo; y la cantidad representa unidades enteras positivas.

Se combinan NOT NULL y CHECK cuando la regla exige ambos aspectos. Una comprobación precio >= 0 no sustituye la obligación de proporcionar precio. La restricción y el mensaje de la interfaz cumplen funciones complementarias: una protege el dato y el otro facilita corregir la entrada.

Los nombres de restricciones también forman parte de la legibilidad. ck_productos_precio permite relacionar un error con su regla. Un conjunto de nombres coherentes facilita investigar la estructura sin depender de la memoria de quien la crea.

2.3. Normalización mediante un problema concreto

Supóngase que cada línea de venta repite el nombre y la dirección del cliente. En Atlas, corregir su dirección exige localizar numerosas filas. También aparece una pregunta: ¿esas filas deben mostrar la dirección actual del cliente o la dirección que corresponde a cada documento?

La propuesta distingue ambas necesidades. CLIENTES conserva la información actual del cliente. Si una operación exige conservar una dirección histórica del documento, la guarda expresamente

como dato de esa operación. La repetición deliberada responde a una regla temporal; la repetición accidental carece de esa explicación.

La normalización organiza dependencias para reducir anomalías de inserción, actualización y eliminación. No se aplica como prohibición absoluta de repetir datos. Primero se identifica qué hecho representa cada atributo y de qué identificador depende. En Atlas, el nombre actual del producto depende del producto; el precio aplicado depende de la línea de venta.

Un ejercicio útil consiste en intentar eliminar la única venta de un cliente. El catálogo de clientes no debe desaparecer con ella. Esa prueba descubre si se mezclan entidades que necesitan ciclos de vida distintos.

2.4. Tipos de datos y precisión

Oracle ofrece tipos numéricos, de texto y temporales. NUMBER permite especificar precisión y escala; VARCHAR2 almacena texto variable; DATE incluye fecha y hora hasta segundos; TIMESTAMP permite mayor precisión temporal. Las variantes con zona horaria responden a necesidades adicionales (Oracle, s. f.-f).

Atlas define importes con dos decimales y unidades enteras. No representa fechas como cadenas de texto ni presupone que el formato visible equivale al almacenamiento. Una pantalla puede mostrar 14/09/2026, pero la consulta utiliza un valor temporal o un literal explícito.

La elección de NUMBER(12,2) establece hasta diez dígitos enteros y dos decimales. Si el dominio exige más rango, se modifica el diseño de forma consciente. Para cantidades fraccionarias, como kilogramos, Atlas necesita otra regla; no basta con eliminar la validación visual.

Una escala de cero define cómo se almacena el número, pero no garantiza que una entrada fraccionaria se rechace antes de convertirse: Oracle puede redondearla. Por eso el procedimiento de existencias de la parte 3 comprueba la cantidad recibida antes del UPDATE. La interfaz y los procesos de importación también necesitan respetar ese contrato.

Se fija también un criterio de redondeo. En el ejercicio, cada importe de línea utiliza cantidad por precio unitario y se redondea a dos decimales. La suma del documento parte de esos importes. Un cálculo que redondea solo al final puede diferir de otro que redondea cada línea; el modelo necesita una convención única.

2.5. Un esquema mínimo coherente

El siguiente esquema constituye un laboratorio nuevo. Los objetos se crean una vez en un esquema de prácticas. No se ejecuta sobre tablas con el mismo nombre y datos que se necesitan conservar. La creación de objetos pertenece a la preparación, separada de las transacciones de negocio.

```
CREATE TABLE productos (
   id_producto NUMBER(10) CONSTRAINT pk_productos PRIMARY KEY,
   codigo VARCHAR2(30 CHAR) NOT NULL,
   nombre VARCHAR2(120 CHAR) NOT NULL,
   precio NUMBER(12,2) NOT NULL,
   stock NUMBER(10) DEFAULT 0 NOT NULL,
   version_fila NUMBER(10) DEFAULT 0 NOT NULL,
   CONSTRAINT uq_productos_codigo UNIQUE (codigo),
   CONSTRAINT ck_productos_precio CHECK (precio >= 0),
   CONSTRAINT ck_productos_stock CHECK (stock >= 0),
   CONSTRAINT ck_productos_version CHECK (version_fila >= 0)
);

CREATE TABLE ventas (
   id_venta NUMBER(10) CONSTRAINT pk_ventas PRIMARY KEY,
   fecha_venta DATE NOT NULL,
   estado VARCHAR2(12 CHAR) DEFAULT 'BORRADOR' NOT NULL,
   CONSTRAINT ck_ventas_estado
      CHECK (estado IN ('BORRADOR', 'CONFIRMADA', 'ANULADA'))
);

CREATE TABLE lineas_venta (
   id_venta NUMBER(10) NOT NULL,
   numero_linea NUMBER(5) NOT NULL,
   id_producto NUMBER(10) NOT NULL,
   cantidad NUMBER(10) NOT NULL,
   precio_unitario NUMBER(12,2) NOT NULL,
   CONSTRAINT pk_lineas_venta PRIMARY KEY (id_venta, numero_linea),
   CONSTRAINT fk_lineas_venta FOREIGN KEY (id_venta)
      REFERENCES ventas (id_venta),
   CONSTRAINT fk_lineas_producto FOREIGN KEY (id_producto)
      REFERENCES productos (id_producto),
   CONSTRAINT ck_lineas_numero CHECK (numero_linea > 0),
   CONSTRAINT ck_lineas_cantidad CHECK (cantidad > 0),
   CONSTRAINT ck_lineas_precio CHECK (precio_unitario >= 0)
);
```

Este núcleo omite clientes y delegaciones para mantener manejable la práctica. No representa todavía todo el sistema conceptual. La existencia de estados tampoco implementa por sí sola sus transiciones: la lógica necesita impedir, por ejemplo, que una venta confirmada vuelva a borrador sin una operación autorizada.

version_fila permite detectar cambios entre la lectura y la escritura. Su uso aparece en las partes 3 y 4. El stock se guarda por producto únicamente en el laboratorio; un sistema con varios almacenes necesita existencias por producto y ubicación.

2.6. Datos de prueba pequeños y explicables

Se utilizan identificadores manuales para que los ejemplos resulten fáciles de seguir. Los números no representan una estrategia de generación para una aplicación multiusuario.

```
INSERT INTO productos (id_producto, codigo, nombre, precio, stock)
VALUES (1, 'TEC-001', 'Teclado USB', 20.00, 10);

INSERT INTO productos (id_producto, codigo, nombre, precio, stock)
VALUES (2, 'RAT-001', 'Raton USB', 15.00, 8);

INSERT INTO ventas (id_venta, fecha_venta, estado)
VALUES (100, DATE '2026-09-14', 'BORRADOR');

INSERT INTO lineas_venta
   (id_venta, numero_linea, id_producto, cantidad, precio_unitario)
VALUES (100, 1, 1, 2, 20.00);

INSERT INTO lineas_venta
   (id_venta, numero_linea, id_producto, cantidad, precio_unitario)
VALUES (100, 2, 2, 1, 15.00);

COMMIT;
```

La venta en borrador contiene dos teclados y un ratón. Su importe es 55. El stock no se reduce porque la confirmación de una venta no se implementa en estos INSERT. Esta distinción evita atribuir a la inserción de líneas un efecto que el script no contiene.

Para generar identificadores se puede utilizar una secuencia Oracle. Sus valores avanzan independientemente de que la transacción se confirme o se deshaga; por ello, una secuencia admite huecos. No se utiliza MAX(id) + 1 como generador multiusuario (Oracle, s. f.-b).

2.7. Leer SQL como una operación sobre conjuntos

Una consulta expresa qué filas y columnas se necesitan. SELECT define la proyección, FROM identifica los orígenes y WHERE filtra. ORDER BY establece un orden explícito; sin él, no se presupone una ordenación estable (Oracle, s. f.-s).

```
SELECT codigo, nombre, precio
FROM productos
WHERE precio >= 15
ORDER BY precio DESC, id_producto;
```

El segundo criterio resuelve empates de precio. En el conjunto inicial, la consulta muestra primero el teclado y después el ratón. La expectativa se conoce antes de ejecutarla: esa costumbre ayuda a detectar errores sin confiar únicamente en que Oracle acepta la sintaxis.

La lógica de conjuntos también cambia la forma de actualizar. Si una regla afecta a numerosos registros, se estudia primero si se expresa mediante una sola instrucción SQL. Leer cada fila en Delphi y enviar una modificación individual introduce coordinación y viajes de red que pueden resultar innecesarios.

2.8. JOIN: unir datos sin multiplicarlos accidentalmente

Un JOIN combina filas según una condición. Una relación uno a varios produce varias filas por cada elemento del lado uno cuando existen varias coincidencias. Un LEFT JOIN conserva además las filas de la izquierda sin correspondencia (Oracle, s. f.-k).

```
SELECT l.numero_linea,
       p.codigo,
       p.nombre,
       l.cantidad,
       l.precio_unitario,
       ROUND(l.cantidad * l.precio_unitario, 2) AS importe
FROM lineas_venta l
JOIN productos p ON p.id_producto = l.id_producto
WHERE l.id_venta = 100
ORDER BY l.numero_linea;
```

El precio procede de la línea y el nombre procede del catálogo. Esa elección tiene una consecuencia: un cambio del nombre actual aparece en esta consulta histórica. Si el documento necesita conservar también la descripción original, Atlas requiere un campo específico en la línea.

La revisión de una unión incluye una pregunta cuantitativa: ¿cuántas filas se esperan por cada clave? Si se une una venta con sus líneas y, simultáneamente, con varios pagos, puede aparecer una combinación de líneas por pagos. Sumar entonces los importes de línea los multiplica. En ese caso se agregan los conjuntos al nivel adecuado antes de combinarlos.

2.9. Agrupar y filtrar periodos

GROUP BY reúne filas para calcular agregados. WHERE filtra filas antes de la agrupación y HAVING filtra grupos. La siguiente consulta mantiene un propósito didáctico: calcula el importe de cada venta sin decidir si ese importe debe formar parte de un indicador comercial (Oracle, s. f.-s).

```
SELECT id_venta,
       SUM(ROUND(cantidad * precio_unitario, 2)) AS total
FROM lineas_venta
GROUP BY id_venta
HAVING SUM(ROUND(cantidad * precio_unitario, 2)) > 0;
```

Para seleccionar septiembre se propone un intervalo con inicio incluido y fin excluido:

```
SELECT id_venta, fecha_venta, estado
FROM ventas
WHERE fecha_venta >= DATE '2026-09-01'
  AND fecha_venta <  DATE '2026-10-01';
```

Así, una operación del último día con hora también pertenece al intervalo. La aplicación envía fechas mediante parámetros temporales, no mediante concatenaciones dependientes del formato regional.

2.10. NULL, cero y ausencia de filas

NULL expresa ausencia de valor y se comprueba mediante IS NULL o IS NOT NULL. No equivale a cero. En Oracle, la cadena de longitud cero se trata actualmente como nula; conviene no utilizarla como un estado de negocio independiente (Oracle, s. f.-n).

También se diferencia un resultado sin filas de una fila cuyo importe es nulo. COUNT(*) cuenta filas; COUNT(columna) cuenta valores no nulos. Si Atlas necesita «cero ventas», define explícitamente cómo transforma la ausencia de datos en ese indicador.

Un error frecuente consiste en sustituir cualquier NULL por cero sin examinar su significado. Un coste desconocido no representa necesariamente un coste gratuito. Esa sustitución puede convertir datos incompletos en márgenes aparentemente válidos.

2.11. Ejercicios y criterio de dominio

Se proponen cuatro comprobaciones. Primero, se intenta repetir TEC-001 y se identifica la restricción que lo impide. Segundo, se intenta insertar una línea con producto inexistente. Tercero, se calcula el total esperado de 55 antes de consultar. Cuarto, se cambia el precio actual del teclado y se comprueba que el precio de su línea conserva 20.

El criterio de dominio consiste en explicar por qué cada resultado aparece. La corrección de una consulta incluye sintaxis, relaciones, nivel de detalle y significado. Una consulta que ejecuta sin errores puede seguir respondiendo a una pregunta distinta de la que se necesita resolver.

Parte 3. PL/SQL, transacciones y concurrencia: proteger la lógica del sistema

3.1. Qué aporta PL/SQL

SQL describe operaciones sobre datos. PL/SQL añade una estructura procedimental con variables, condiciones, subprogramas y tratamiento de excepciones. Un bloque puede organizar varias instrucciones y expresar decisiones alrededor de ellas (Oracle, s. f.-q).

En Atlas, el objetivo no consiste en trasladar cualquier cálculo a Oracle. Se centralizan las reglas que deben mantenerse cuando diferentes clientes modifican los mismos datos. Cambiar un precio exige comprobar su validez; descontar existencias exige verificar que la cantidad disponible resulta suficiente.

La interfaz también valida, pero su comprobación no sustituye a la regla central. Una importación o un segundo programa puede realizar la misma acción. El diseño necesita que todos encuentren una conducta coherente, con independencia de la pantalla que utilizan.

3.2. Bloques, procedimientos, funciones y paquetes

Un bloque anónimo permite ejecutar una unidad de código sin crear un procedimiento permanente. Un procedimiento representa una operación con parámetros; una función devuelve un valor. La decisión entre ambos responde al propósito de la interfaz del código, no al deseo de reducir líneas.

Un paquete agrupa elementos relacionados. Su especificación declara lo que otros componentes utilizan; el cuerpo contiene la implementación. Oracle compila y conserva estos objetos en la base de datos (Oracle, s. f.-x).

Atlas propone pkg_catalogo para operaciones del catálogo. La pantalla conoce el nombre de la operación y sus parámetros. No necesita reproducir el detalle de su UPDATE. Si la regla cambia, el cuerpo concentra esa modificación siempre que el contrato se conserva.

Se evita que el paquete se convierta en un contenedor de cualquier función del sistema. Ventas, catálogo y cargas representan responsabilidades distintas. Los nombres expresan acciones y conceptos del dominio, no solamente instrucciones técnicas como «ejecutar SQL».

3.3. Parámetros y contratos

Un contrato indica qué recibe una operación, qué resultado produce y qué errores representa. En Atlas, cambiar un precio recibe identificador, nuevo importe y versión que muestra la pantalla. Si esa versión ya no coincide, el sistema rechaza la actualización para evitar sobrescribir un cambio ajeno.

Los parámetros IN aportan entradas. OUT permite devolver resultados e IN OUT combina ambas direcciones. %TYPE vincula una declaración al tipo de una columna, lo que facilita mantener

consistencia de tipos. Esa declaración no equivale a heredar automáticamente todas las restricciones de negocio de la columna.

Por eso el procedimiento comprueba explícitamente nulos, rango y precisión admitida. En el caso de Atlas, 24,999 no constituye un precio válido de dos decimales. El contrato rechaza ese valor antes de que una conversión de almacenamiento lo redondee sin una decisión visible.

3.4. Atomicidad: definir qué se confirma junto

Una transacción agrupa instrucciones como una unidad. COMMIT confirma los cambios y ROLLBACK los deshace. En Oracle, las instrucciones DDL introducen confirmaciones implícitas en los casos que describe su documentación; no se mezclan como si fueran simples modificaciones de negocio reversibles (Oracle, s. f.-v).

La confirmación de una venta de Atlas incluye cabecera, líneas y existencias. La política elegida exige que se conserven todos esos cambios o ninguno. Un COMMIT dentro de cada procedimiento auxiliar rompe esa unidad: una parte puede quedar confirmada aunque otra falle.

Se propone que el coordinador de la operación controle la transacción. Un procedimiento de catálogo realiza su acción y deja la decisión final al llamador. Cuando se diseña una API PL/SQL que controla transacciones por sí misma, esa responsabilidad también necesita documentación explícita; no se presupone a partir de su nombre.

La transacción se mantiene breve. Atlas no bloquea una fila mientras la persona decide durante varios minutos qué precio introduce. La edición ocurre en la interfaz y el control de conflicto se ejecuta al guardar.

3.5. Concurrencia: dos usuarios sobre el mismo producto

Se plantea un caso con dos sesiones. Ambas leen precio 20 y versión 0. La primera cambia el precio a 22. La segunda intenta guardarlo a 19 utilizando todavía la versión 0. Sin una comprobación adicional, el segundo cambio puede sobrescribir el primero aunque el usuario no lo conoce.

Oracle gestiona bloqueos y consistencia de lectura, pero la aplicación necesita expresar qué conflicto de negocio desea detectar. Un UPDATE que incluye los valores originales o una versión permite comprobar esa condición (Oracle, s. f.-c).

Atlas utiliza control optimista: la lectura no mantiene un bloqueo de edición y la escritura exige que la versión coincida. Cada cambio admitido incrementa la versión. Si ninguna fila cumple la condición, se solicita recargar el dato.

La alternativa pesimista reserva la fila mediante una operación de bloqueo, como SELECT FOR UPDATE, dentro de una transacción. Su aplicación exige considerar espera, duración y orden de acceso. El laboratorio utiliza la opción optimista para la edición del catálogo y una modificación condicional para existencias.

3.6. Un paquete pequeño con reglas reales

El siguiente código utiliza PRODUCTOS de la parte 2. Se crea después de la tabla. La barra que aparece tras cada definición es un separador para clientes que ejecutan scripts; no se incorpora al texto PL/SQL que Delphi envía como llamada.

```
CREATE OR REPLACE PACKAGE pkg_catalogo AS
   PROCEDURE cambiar_precio (
      p_id IN productos.id_producto%TYPE,
      p_precio IN productos.precio%TYPE,
      p_version IN productos.version_fila%TYPE
   );

   PROCEDURE descontar_stock (
      p_id IN productos.id_producto%TYPE,
      p_cantidad IN productos.stock%TYPE
   );
END pkg_catalogo;
/
```

El cuerpo implementa las comprobaciones y las operaciones. Ningún procedimiento confirma la transacción.

```
CREATE OR REPLACE PACKAGE BODY pkg_catalogo AS

  PROCEDURE cambiar_precio (
    p_id IN productos.id_producto%TYPE,
    p_precio IN productos.precio%TYPE,
    p_version IN productos.version_fila%TYPE
  ) IS
  BEGIN
    IF p_precio IS NULL
      OR p_precio < 0
      OR p_precio > 9999999999.99
      OR p_precio <> ROUND(p_precio, 2) THEN
       RAISE_APPLICATION_ERROR(-20001, 'Precio no valido');
    END IF;

    IF p_version IS NULL OR p_version < 0
      OR p_version <> TRUNC(p_version) THEN
       RAISE_APPLICATION_ERROR(-20002, 'Version no valida');
    END IF;

    UPDATE productos
      SET precio = p_precio,
          version_fila = version_fila + 1
     WHERE id_producto = p_id
      AND version_fila = p_version;

    IF SQL%ROWCOUNT = 0 THEN
       RAISE_APPLICATION_ERROR(
          -20003, 'Producto ausente o modificado; recargue'
       );
    END IF;
  END cambiar_precio;

  PROCEDURE descontar_stock (
    p_id IN productos.id_producto%TYPE,
    p_cantidad IN productos.stock%TYPE
  ) IS
  BEGIN
    IF p_cantidad IS NULL OR p_cantidad <= 0
      OR p_cantidad <> TRUNC(p_cantidad) THEN
       RAISE_APPLICATION_ERROR(-20004, 'Cantidad no valida');
    END IF;

    UPDATE productos
      SET stock = stock - p_cantidad,
          version_fila = version_fila + 1
     WHERE id_producto = p_id
      AND stock >= p_cantidad;

    IF SQL%ROWCOUNT = 0 THEN
```

La operación de stock comprueba disponibilidad y descuenta en una misma instrucción. No lee primero el stock en Delphi para calcular y escribir después un valor absoluto. Ese patrón separado puede utilizar un dato obsoleto.

La versión de Atlas protege toda la fila. Por tanto, un movimiento de stock también invalida una edición de precio que conserva una versión anterior. Es una política conservadora. Un diseño que necesita separar conflictos por campos define otra estrategia y sus pruebas; no elimina el control sin estudiar el efecto.

3.7. Excepciones: distinguir rechazo y fallo

Una excepción interrumpe el flujo ordinario y permite tratar una condición anómala. Oracle dispone de excepciones predefinidas y mecanismos para declarar y propagar errores de aplicación (Oracle, s. f.-p).

En Atlas se distinguen tres categorías: entrada inválida, conflicto de negocio y fallo técnico. Precio negativo pertenece a la primera. Una versión antigua pertenece a la segunda. Una conexión interrumpida pertenece a la tercera. Las tres requieren respuestas diferentes.

El mensaje de la pantalla explica qué puede hacer la persona. El registro técnico conserva información suficiente para investigar. No se transforma cualquier excepción en «Guardado correctamente», ni se utiliza WHEN OTHERS THEN NULL para ocultarla.

Tampoco se asume que lanzar una excepción deshace siempre toda la operación lógica que rodea al procedimiento. El coordinador conserva la responsabilidad de finalizar la transacción de acuerdo con su contrato. La parte 4 muestra ese control desde Delphi.

3.8. Probar la atomicidad sin alterar el estado de referencia

El siguiente bloque se ejecuta con autocommit desactivado y sin cambios pendientes ajenos al ejercicio. Se utilizan un SAVEPOINT y una reversión explícita. La segunda cantidad supera las existencias iniciales, por lo que se espera un error y se deshace también el primer descuento.

```
SAVEPOINT inicio_prueba;

BEGIN
   pkg_catalogo.descontar_stock(1, 2);
   pkg_catalogo.descontar_stock(2, 9999999999);
EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK TO inicio_prueba;
      RAISE;
END;
/
```

Después se consultan las existencias y se comparan con el valor anterior a la prueba. La evidencia correcta consiste en que ambos productos conservan ese valor, no solamente en que aparece una excepción.

Una prueba de cambio de precio utiliza también un SAVEPOINT, invoca el procedimiento con la versión que devuelve la consulta, verifica el precio dentro de la sesión y ejecuta ROLLBACK TO. Si se usa una versión fija, se comprueba antes que coincide con el estado actual.

3.9. Conjuntos, bucles y rendimiento

La existencia de bucles en PL/SQL no significa que cada transformación necesite recorrer fila a fila. Atlas estudia primero una instrucción SQL que expresa el conjunto completo. Cuando existe una necesidad procedimental, se analiza el volumen y la frecuencia de interacción entre SQL y PL/SQL.

Las técnicas de procesamiento masivo, como BULK COLLECT y FORALL, permiten reducir determinadas interacciones cuando se utiliza lógica procedimental sobre colecciones. Su elección exige estudiar memoria, tamaño de lote y tratamiento de errores (Oracle, s. f.-a).

Para el laboratorio de dos productos no se añade procesamiento masivo: no existe un volumen que justifique esa complejidad. El aprendizaje consiste en reconocer la diferencia entre una solución correcta para el caso y una optimización que responde a una medición.

3.10. Pruebas de contrato

Situación	Conducta que Atlas exige
Precio válido y versión vigente	Se modifica una fila y se incrementa la versión.
Precio negativo o con precisión no admitida	Se rechaza sin modificar la fila.
Versión antigua	Se rechaza y se requiere recargar.
Producto inexistente	No se comunica una actualización correcta.
Cantidad nula, negativa o fraccionaria	Se rechaza el descuento.
Existencias insuficientes	Se conserva el stock.
Dos descuentos concurrentes que superan juntos el disponible	No se permite un stock negativo.

La última prueba requiere dos sesiones reales. Una simulación secuencial no demuestra concurrencia. El criterio de estudio incluye justificar el resultado esperado y observar bloqueos o esperas cuando corresponden.

Con estas reglas, la interfaz dispone de operaciones con un contrato explícito. La parte 4 se centra en invocarlas correctamente, presentar sus resultados y mantener separada la edición visual de la persistencia.

Referencias

Las referencias se presentan según APA 7 y proceden de documentación oficial de Oracle. Se utiliza «s. f.» cuando la página consultada no presenta una fecha editorial identificable.

Oracle. (s. f.-a). *Bulk SQL and bulk binding*. Oracle Database 19c PL/SQL Language Reference. https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/bulk-sql-and-bulk-binding.html

Oracle. (s. f.-b). *CREATE SEQUENCE*. Oracle Database 19c SQL Language Reference. https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-SEQUENCE.html

Oracle. (s. f.-c). *Data concurrency and consistency*. Oracle Database 19c Database Concepts. https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/data-concurrency-and-consistency.html

Oracle. (s. f.-d). *Data integrity*. Oracle Database 19c Database Concepts. https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/data-integrity.html

Oracle. (s. f.-f). *Data types*. Oracle Database 19c SQL Language Reference. https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Data-Types.html

Oracle. (s. f.-j). *Introduction to data warehousing concepts*. Oracle Database 19c Data Warehousing Guide. https://docs.oracle.com/en/database/oracle/oracle-database/19/dwhsg/introduction-data-warehouse-concepts.html

Oracle. (s. f.-k). *Joins*. Oracle Database 19c SQL Language Reference. https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Joins.html

Oracle. (s. f.-n). *Nulls*. Oracle Database 19c SQL Language Reference. https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Nulls.html

Oracle. (s. f.-p). *Overview of exception handling*. Oracle Database 19c PL/SQL Language Reference. https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/overview-exception-handling.html

Oracle. (s. f.-q). *Overview of PL/SQL*. Oracle Database 19c PL/SQL Language Reference. https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/overview.html

Oracle. (s. f.-s). *SELECT*. Oracle Database 19c SQL Language Reference. https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/SELECT.html

Oracle. (s. f.-u). *Tables and table clusters*. Oracle Database 18c Database Concepts. https://docs.oracle.com/en/database/oracle/oracle-database/18/cncpt/tables-and-table-clusters.html

Oracle. (s. f.-v). *Transactions*. Oracle Database 19c Database Concepts. https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/transactions.html

Oracle. (s. f.-x). *What is a package?*. Oracle Database 19c PL/SQL Language Reference. https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/what-is-package.html