# Trabajo 2: PL/SQL - Gestión de Pedidos en Restaurante

Marzo 2025

## 1. Descripción de la práctica

Se presenta el caso de uso de un restaurante que gestiona los pedidos de los clientes. Cada pedido puede contener varios platos y se asigna a un miembro del personal de servicio que lo atiende. La gestión del pedido sigue estas reglas:

- Un cliente puede realizar múltiples pedidos.
- Cada pedido debe incluir al menos un plato (primer o segundo plato).
- El personal de servicio no puede gestionar más de 5 pedidos activos simultáneamente.
- Si un plato no está disponible, el pedido no se puede completar.
- Se registra la información del cliente, los platos solicitados y el personal de servicio asignado.

Las tablas involucradas en la gestión de pedidos son:

- **Clientes** (`id_cliente` (PK), nombre, apellido, teléfono)
- **Personal_servicio** (`id_personal` (PK), nombre, apellido, pedidos_activos)
- **Pedidos** (`id_pedido` (PK), `id_cliente` (FK → Clientes), `id_personal` (FK → Personal_servicio), fecha_pedido, total)
- **Platos** (`id_plato` (PK), nombre, precio, disponible (entero (0,1)))
- **Detalle_pedido** (`id_pedido` (FK → Pedidos), `id_plato` (FK → Platos), cantidad)

Además, existe la siguiente secuencia:

- **seq_pedidos**: alimenta los valores de la clave primaria de la tabla de pedidos (`id_pedido`).

Para facilitar la tarea se facilita el script `registrar_pedido_enun.sql` junto a este enunciado, que además de crear las tablas previamente descritas contiene:

1. El procedimiento `reset_seq(p_seq_name VARCHAR)` para resetear secuencias.
2. El procedimiento `inicializa_test` que reinicia la base de datos con datos de prueba.
3. Procedimiento vacío para la batería de pruebas `test_registrar_pedido`.

En este trabajo **se pide la implementación de un procedimiento que contenga una transacción que permita gestionar un nuevo pedido.** Además, se pide la **implementación de la batería de test** que comprueben la funcionalidad implementada.

```
CREATE OR REPLACE PROCEDURE registrar_pedido (
    arg_id_cliente INTEGER,
    arg_id_personal INTEGER,
    arg_id_primer_plato INTEGER,
    arg_id_segundo_plato INTEGER
) IS
BEGIN
    -- Implementación aquí
END;
/
```

Para proceder a gestionar un nuevo pedido la transacción debe:

1. Comprobar que el plato o platos solicitados están disponibles. En caso contrario devolverá la **excepción -20001, con el mensaje "Uno de los platos seleccionados no está disponible.'**
2. Si se cumple la comprobación anterior, y existe el plato, se comprobará que se puede cargar el pedido a esa persona del servicio. A lo largo de la transacción también pueden encontrarse errores que darán lugar a diferentes excepciones:

- En caso de que el pedido no contenga ningún plato, debe lanzarse la **excepción -20002, con el mensaje 'El pedido deber contener al menos un plato.'**
- En caso de que el personal de servicio al que se encarga el pedido haya cubierto su cupo máximo de pedidos (5 pedidos), se lanzará **excepción -20003, el personal de servicio tiene demasiados pedidos.**
- En caso de que uno de los platos seleccionado no exista, se lanzará la **excepción -20004, con el mensaje de error 'El primer plato seleccionado no existe' o 'El segundo plato seleccionado no existe'**.

3. Para formalizar el pedido, y habiéndose cumplido lo anterior, se deben realizar tres acciones (y nunca una sin las otras):
   - Añadir el pedido a la tabla pedidos.
   - Añadir los detalles de pedido a la tabla detalle_pedido.
   - Actualizar la tabla de personal_servicio para añadir un pedido al personal encargado de este pedido.
4. Incluye en los comentarios del código la respuesta a las siguientes preguntas (indicando la referencia a la pregunta en cada caso P4.1, P4.2, P4.3):
   - **P4.1** ¿Cómo garantizas en tu código que un miembro del personal de servicio no supere el límite de pedidos activos?
   - **P4.2** ¿Cómo evitas que dos transacciones concurrentes asignen un pedido al mismo personal de servicio cuyos pedidos activos estan a punto de superar el límite?
   - **P4.3** Una vez hechas las comprobaciones en los pasos 1 y 2, ¿podrías asegurar que el pedido se puede realizar de manera correcta en el paso 4 y no se generan inconsistencias? ¿Por qué? Recuerda que trabajamos en entornos con conexiones .concurrentes.
   - **P4.4** Si modificásemos la tabla de `personal_servicio` añadiendo `CHECK (pedido_activos` $\leq$ `5)`, ¿Qué implicaciones tendría en tu código? ¿Cómo afectaría en la gestión de excepciones? Describe en detalle las modificaciones que deberías hacer en tu código para mejorar tu solución ante esta situación (puedes añadir pseudocódigo).

- **P4.5** ¿Qué tipo de estrategia de programación has utilizado? ¿Cómo puede verse en tu código?

5. **BATERÍA DE TEST**: Comprobará al menos que se cumple que:
   - Se realiza un pedido con un primer y/o segundo plato disponibles y personal con capacidad.
   - Si se realiza un pedido vacío (sin platos) devuelve el error -20002.
   - Si se realiza un pedido con un plato que no existe devuelve en error -20004.
   - Si se realiza un pedido que incluye un plato que no está ya disponible devuelve el error -20001.
   - Personal de servicio ya tiene 5 pedidos activos y se le asigna otro pedido devuelve el error -20003

# 2. Normas de Entrega

La práctica se realizará **EN GRUPOS DE TRES PERSONAS**
Formato de entrega:

- Se creará un repositorio en **GitHub** para este trabajo, valorándose la contribución de cada persona del grupo en el resultado final, así como la progresividad en el avance (se penalizará entregas con commits sólo en el último momento). Por si no estáis muy familiarizad@s aún con Git, aquí tenéis una guía básica de comandos.
- Incluir en el README del proyecto git información de las personas autoras.
- Para poder evaluarlo deberéis invitar a la profesora (asmamolar@ubu.es) al repositorio en el momento en el que lo creéis. En la cabecera del script que entregaréis, tenéis que añadir también el enlace al repositorio.
- Se enviará un fichero `.zip` a través de la plataforma **UBUVirtual** completando la tarea:

  **[ABD] Trabajo 2 - PLSQL-1C -24_25**

- El fichero comprimido (`.zip` o `.rar`) seguirá el siguiente formato de nombre, sin utilizar tildes:
  `Nombre_1erApellido_Nombre_1erApellido_Nombre_1erApellido.zip`

- El fichero comprimido contendrá todos los scripts **PL/SQL** (formato `.sql`) generados para resolver el enunciado y que incluya todas las cuestiones planteadas (recuerda que las respuestas a la preguntas deben insertarse como comentarios en el propio códido).
- **El código entregado debe compilar correctamente para poder ser evaluado**, en caso contrario la evaluación será un **0**.
- Se valorará la documentación del código. El código debe estar bien documentado, indicando las decisiones tomadas en cada caso, así como cualquier explicación que demuestre que se conocen los conceptos vistos en la asignatura.
- En caso de alguna irregularidad, podría solicitarse una **defensa del trabajo** en horario de tutorías, a fijar con las profesoras responsables de la asignatura. Una exagerada diferencia entre el resultado de este trabajo y el del examen del tema 2 supone razón suficiente para solicitar esa defensa. En caso de no ser capaz de defender el trabajo, la calificación será modificada acorde.
- **Fecha máxima de entrega: Jueves 2 de Abril a las 23.59h.**
- Aunque la entrega sea grupal, deben entregarlo todas las personas que formen el grupo. **La persona que no lo entregue en UBUVirtual en tiempo, no podrá ser evaluada.**
- **NO SE PERMITEN ENTREGAS DESPUÉS DE LA FECHA LÍMITE.**
- **LAS DUDAS SE RESOLVERÁN A TRAVÉS DEL FORO.**