Fundamentos de Bases de Datos

Práctica- 2

Rodrigo Díaz-Regañón Ureña
Daniel Martínez Fernández

1. Introducción y objetivos

Esta práctica trata sobre el acceso a bases de datos desde un programa escrito en lenguaje C. El objetivo principal es implementar el acceso a la base de datos para dos funcionalidades: la primera es una búsqueda de vuelos (search) que, a partir de un origen, destino y una fecha, debe encontrar rutas (directas o con un transbordo, de menos de 24h) con asientos libres, mostrando detalles (como flight_id y aircraft_code) al seleccionarlas; y la segunda se encarga de la emisión de tarjetas de embarque a partir de un book_ref, asignando automáticamente el primer asiento disponible, actualizando la tabla boarding_passes y gestionando la concurrencia. Además se ha implementado una paginación para mostrar los resultados de una manera más eficiente. Por último se ha usado el valgrind para comprobar los errores, otorgando el valgrind.log.

2. Realización de búsquedas.

Antes de explicar la implementación en C se va a explicar la consulta de SQL. Esta consulta busca opciones de viaje entre dos aeropuertos (?) para una fecha específica (?). Existen dos tipos de vuelo, los vuelos directos y los que tienen escala (siempre que el viaje total sea menor a 24 horas). Para cada opción, calcula los asientos libres disponibles, asegurándose de que el resultado sea mayor que cero (asientos_libres > 0); en el caso de las escalas, utiliza el número LEAST (mínimo) de asientos libres entre los dos vuelos. Finalmente, la consulta externa filtra todos estos resultados por el origen, destino y fecha proporcionados, y los ordena por la duración total del viaje (last_arrival - first_departure ASC) para mostrar primero la opción más rápida.

```sql
SELECT flight_id_1, flight_id_2, aircraft_code_1, aircraft_code_2, first_departure, last_arrival, num_vuelos, asientos_libres
FROM (
  SELECT
    f.flight_id AS flight_id_1,
    NULL AS flight_id_2,
    f.aircraft_code AS aircraft_code_1,
    NULL AS aircraft_code_2,
    f.departure_airport,
    f.arrival_airport,
    f.scheduled_departure AS first_departure,
    f.scheduled_arrival AS last_arrival,
    1 AS num_vuelos,
    (
      SELECT COUNT(s.seat_no) - COUNT(bp.seat_no)
      FROM seats s
      LEFT JOIN boarding_passes bp
        ON bp.seat_no = s.seat_no AND bp.flight_id = f.flight_id
      WHERE s.aircraft_code = f.aircraft_code
    ) AS asientos_libres
  FROM flights f
  UNION ALL
```

Esta es la primera parte de la consulta que consigue vuelos directos.

```sql
  SELECT
    f1.flight_id AS flight_id_1,
    f2.flight_id AS flight_id_2,
    f1.aircraft_code AS aircraft_code_1,
    f2.aircraft_code AS aircraft_code_2,
    f1.departure_airport,
    f2.arrival_airport,
    f1.scheduled_departure AS first_departure,
    f2.scheduled_arrival AS last_arrival,
    2 AS num_vuelos,
    LEAST(
      (
        SELECT COUNT(s1.seat_no) - COUNT(bp1.seat_no)
        FROM seats s1
        LEFT JOIN boarding_passes bp1
          ON bp1.seat_no = s1.seat_no AND bp1.flight_id = f1.flight_id
        WHERE s1.aircraft_code = f1.aircraft_code
      ),
      (
        SELECT COUNT(s2.seat_no) - COUNT(bp2.seat_no)
        FROM seats s2
        LEFT JOIN boarding_passes bp2
          ON bp2.seat_no = s2.seat_no AND bp2.flight_id = f2.flight_id
        WHERE s2.aircraft_code = f2.aircraft_code
      )
    ) AS asientos_libres
  FROM flights f1
  JOIN flights f2
    ON f1.arrival_airport = f2.departure_airport
  WHERE f2.scheduled_departure > f1.scheduled_arrival
    AND (f2.scheduled_arrival - f1.scheduled_departure) <= INTERVAL '24 hours'
) AS vuelos
WHERE departure_airport = ?
  AND arrival_airport = ?
  AND DATE(first_departure) = ?
  AND asientos_libres > 0
ORDER BY last_arrival - first_departure ASC;
```

Esta es la segunda parte de la consulta, la cual une (con UNION ALL) los vuelos directos con los vuelos con trasbordo mientras se cumplen las condiciones requeridas.

Para implementar esta consulta, usamos la función ‘results_search’. Está primero valida que los parámetros de entrada (origen, destino y fecha) no sean nulos o vacíos. Si falta algún dato, se omite la conexión a la base de datos, se escribe un mensaje de error en la primera línea del array de resultados y se

finaliza la ejecución. Si los datos son válidos, se procede a establecer la conexión ODBC, gestionando también cualquier error que pueda ocurrir en este paso.

Una vez conectada, la función utiliza "prepared statements" (los ? en la consulta SQL). Estos actúan como marcadores de posición. Primero, se llama a SQLPrepare, que envía la plantilla de la consulta al servidor de la base de datos para que la compile. Después, se utiliza SQLBindParameter para vincular las variables C que contienen la entrada del usuario (from, to y date) a estos marcadores de posición. Luego, se prepara la recepción de los datos de vuelta. Para ello, se utiliza SQLBindCol para cada una de las ocho columnas que la consulta SELECT debe devolver. Esta función le indica a ODBC en qué variable local de C debe almacenar el dato de cada columna cuando se recupere una fila.

```c
/* Conexino a la base de datos */
ret = odbc_connect(&env, &dbc);
if (!SQL_SUCCEEDED(ret))
{
    fprintf(stderr, "Error conectando a la base de datos.\n");
    snprintf((*choices)[0], max_length, "Error: no se pudo conectar a la base de datos.");
    *n_choices = 1;
    return;
}

/* Reservar un handle para el statement */
SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

/*Preparamos la consulta*/
ret = SQLPrepare(stmt, (SQLCHAR *)query, SQL_NTS);
if (!SQL_SUCCEEDED(ret))
{
    snprintf((*choices)[0], max_length, "Error preparando la consulta SQL.");
    *n_choices = 1;
    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    odbc_disconnect(env, dbc);
    return;
}

date_len = (SQLLEN)strlen(date);
SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR, 3, 0, from, 0, &ind_from);
SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR, 3, 0, to, 0, &ind_to);
SQLBindParameter(stmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR, 10, 0, date, 0, &date_len);

/*Vinculamos la columna*/
SQLBindCol(stmt, 1, SQL_C_LONG, &flight_id1, sizeof(flight_id1), &ind_fl1);
SQLBindCol(stmt, 2, SQL_C_LONG, &flight_id2, sizeof(flight_id2), &ind_fl2);
SQLBindCol(stmt, 3, SQL_C_CHAR, aircraft_code_1, sizeof(aircraft_code_1), &ind_ac1);
SQLBindCol(stmt, 4, SQL_C_CHAR, aircraft_code_2, sizeof(aircraft_code_2), &ind_ac2);
SQLBindCol(stmt, 5, SQL_C_CHAR, first_departure, sizeof(first_departure), &ind_first);
SQLBindCol(stmt, 6, SQL_C_CHAR, last_arrival, sizeof(last_arrival), &ind_last);
SQLBindCol(stmt, 7, SQL_C_LONG, &num_vuelos, sizeof(num_vuelos), &ind_num);
SQLBindCol(stmt, 8, SQL_C_LONG, &asientos_libres, sizeof(asientos_libres), &ind_asientos);
```

Una vez que tanto los parámetros de entrada como los búferes de salida están vinculados, se llama a SQLExecute para que la base de datos ejecute la consulta preparada con los valores proporcionados. Si la ejecución es exitosa, el programa entra en un bucle que llama repetidamente a SQLFetch para procesar los resultados fila por fila, hasta el límite máximo permitido (max_rows). Con cada llamada, las variables C vinculadas con SQLBindCol se rellenan automáticamente. Por último, dentro del bucle el ‘snprintf’ formatea todas estas variables en una única cadena de texto. Esta cadena formateada se almacena en el array choices.

```
    ret = SQLExecute(stmt);
    if (!SQL_SUCCEEDED(ret))
    {
        snprintf((*choices)[0], max_length, "Error ejecutando la consulta SQL.");
        *n_choices = 1;
        SQLCloseCursor(stmt);
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        odbc_disconnect(env, dbc);
        return;
    }

    /* Recorrer todas las filas devueltas */
    i = 0;
    while (SQL_SUCCEEDED(ret = SQLFetch(stmt)) && i < max_rows)
    {
        snprintf((*choices)[i], max_length,
                 "Salida: %-19.19s | Llegada: %-19.19s | Vuelos: %d | Plazas: %4d"
                 " | INFO: %ld %s %ld %s",
                 (ind_first == SQL_NULL_DATA) ? "N/A" : (char *)first_departure,
                 (ind_last == SQL_NULL_DATA) ? "N/A" : (char *)last_arrival,
                 (ind_num == SQL_NULL_DATA) ? 0 : (int)num_vuelos,
                 (ind_asientos == SQL_NULL_DATA) ? 0 : (int)asientos_libres,
                 (ind_fl1 == SQL_NULL_DATA) ? -1 : (long)flight_id1,
                 (ind_ac1 == SQL_NULL_DATA) ? "N/A" : (char *)aircraft_code_1,
                 (ind_fl2 == SQL_NULL_DATA) ? -1 : (long)flight_id2,
                 (ind_ac2 == SQL_NULL_DATA) ? "N/A" : (char *)aircraft_code_2);
        i++;
    }
```

Cuando termina el bucle de SQLFetch, la función comprueba el contador de filas i. Si i es cero, la consulta no devolvió resultados, por lo que se devuelve un mensaje de que no hay datos para esos valores para el array de salida. Si i es mayor que cero, su valor se asigna a '*n_choices' para informar al bucle principal cuántas opciones de vuelo se encontraron. Finalmente, se liberan todos los recursos de ODBC (SQLCloseCursor, SQLFreeHandle) y se cierra la conexión (odbc_disconnect) para asegurar una gestión de memoria correcta y evitar fugas.

```
    /* Comprobar si se encontraron resultados */
    if (i == 0)
    {
        snprintf((*choices)[0], max_length, "Error no hay datos con estos valores (sin resultados).");
        *n_choices = 1;
    }
    else
    {
        *n_choices = i;
    }

    /* Liberar handles y desconectar */
    SQLCloseCursor(stmt);
    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    odbc_disconnect(env, dbc);
```

Es conveniente señalar que se ha tenido que modificar la propia función para añadir el parámetro de ‘date’ (se ha metido char *date) porque no estaba inicialmente. Esto implica que en menú.c también se ha añadido la opción de Date.

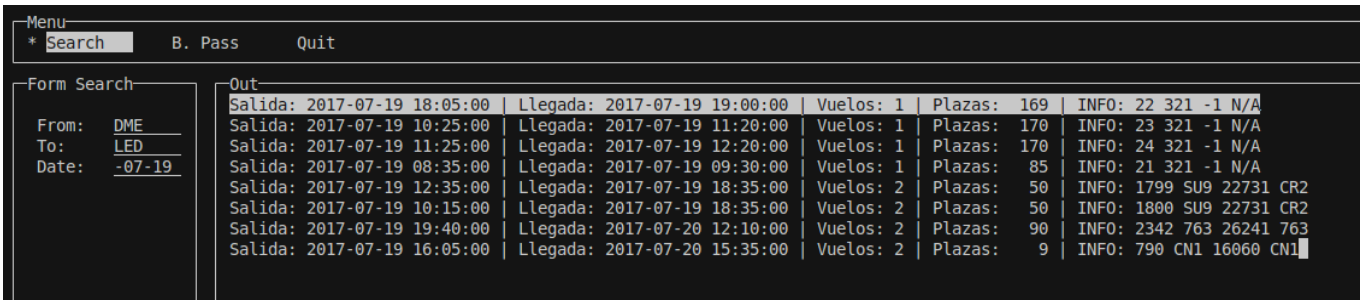
```
char *forms_search_choices[] = {
        "From: ",
        "To:   ",
        "Date: "
};
```

Otro aspecto importante es el hecho de que se ha borrado en el loop.c el caso 0x2D (case : 0x2D) porque para el ‘Date’ se usa el formato YYYY-MM-DD, por lo que si se usaba la tecla ‘-’ se realizaba una acción.

Para finalmente conectarnos a la base de datos y probar la función hemos cambiado el odbc.h y hemos metido la base de datos flight.sql dentro de nuestro directorio.

```
#define CONNECTION_PARS "DRIVER=PostgreSQL ANSI;DATABASE=flight;SERVER=localhost;PORT=5432;UID=alumnodb;PWD=alumnodb;"
```

Esta función nos devuelve en el menú todos los vuelos entre dos aeropuertos en una fecha determinada:



Para ver en detalle un vuelo le damos al enter y así nos sale en la zona inferior (windows->msg) la información completa de cada vuelo, tal y como pide el ejercicio.

Para cumplir con este requisito de poder "ver en detalle los vuelos" se implementan cambios en el bucle principal de loop.c. Cuando el programa detecta que se ha pulsado la tecla Enter (case 10:) mientras el foco está en la ventana de resultados (focus == FOCUS_RIGHT) y la chioce es SEARCH, se inicia este proceso. Primero, el código obtiene la cadena de texto completa de la línea que el usuario tiene

seleccionada (menus->out_win_choices[out_highlight]). El loop.c utiliza sscanf con un formato para ir extrayendo los identificadores flight_id1, flight_id2 y los códigos aircraft_code_1 y aircraft_code_2. Con estos datos, se formatea una nueva cadena de "DETALLES:" que incluye la fecha, el flight_id y el aircraft_code y se utiliza la función write_msg para mostrar esta información detallada en la message_window inferior, misma ventana que también se usa para reportar errores. Además se diferencia entre si es vuelo directo o vuelo con trasbordo.

```c
/*SEARCH desde out window */
else if (choice == SEARCH && focus == FOCUS_RIGHT)
{
    if (n_out_choices == 0)
    {
        choice = -1;
        enterKey = FALSE;
        continue;
    }
    linea = menus->out_win_choices[out_highlight];
    if (out_highlight == 0 &&
        (strncmp(linea, "Error", 5) == 0))
    {
        write_msg(msg_win, linea, -1, -1, windows->msg_title);
    }
    else
    {
        /* sscanf que COINCIDE con el snprintf de search.c */
        int n = sscanf(linea,
                       "Salida: %29[^|] | Llegada: %29[^|] | Vuelos: %d | Plazas: %d | INFO: %ld %9s %ld %9s",
                       salida_str,
                       llegada_str,
                       &num_vuelos,
                       &num_plazas,
                       &flight_id1,
                       ac1_str,
                       &flight_id2,
                       ac2_str);

        /* comprueba si leyó 8 elementos */
        if (n < 8)
        {
            snprintf(buffer, sizeof(buffer), "Error: no se pudo leer la línea. (n=%d)", n);
            write_msg(msg_win, buffer, -1, -1, windows->msg_title);
        }
        else
        {
            /* muestra los datos */
            if (flight_id2 == -1)
            {
                /* Vuelo directo */
                snprintf(buffer, sizeof(buffer),
                         "DETALLES: Vuelo ID %ld (Aircraft_code %s). Salida: %s. Llegada: %s",
                         flight_id1, ac1_str, salida_str, llegada_str);
            }
            else
            {
                /* Vuelo con escala */
                snprintf(buffer, sizeof(buffer),
                         "DETALLES: Vuelo1(ID %ld, Aircraft_code %s) -> Vuelo2(ID %ld, Aircraft_code %s). Salida: %s. Llegada: %s",
                         flight_id1, ac1_str, flight_id2, ac2_str, salida_str, llegada_str);
            }
            write_msg(msg_win, buffer, -1, -1, windows->msg_title);
        }
    }
}
```

3. Emisión de tarjetas de embarque

Antes de explicar la implementación en C (es muy parecida a la de los vuelos) se va a explicar la consulta de SQL. Para hacerlo de una forma más sencilla en vez de hacerlo en 1 consulta se ha dividido en 4 subconsultas.

La primera subconsulta busca todos los billetes (ticket_flights) asociados a un código de reserva (book_ref = ?) que todavía no tienen una tarjeta de embarque emitida. Para ello se utiliza un LEFT JOIN con boarding_passes y la condición WHERE bp.flight_id IS NULL.

```
SELECT tf.flight_id, tf.ticket_no, t.passenger_name, f.aircraft_code, f.scheduled_departure
FROM ticket_flights AS tf
JOIN tickets AS t ON tf.ticket_no = t.ticket_no
JOIN flights AS f ON tf.flight_id = f.flight_id
LEFT JOIN boarding_passes AS bp ON tf.ticket_no = bp.ticket_no AND tf.flight_id = bp.flight_id
WHERE t.book_ref = ? AND bp.flight_id IS NULL
ORDER BY tf.ticket_no ASC
```

En la segunda subconsulta, para cada billete encontrado en la consulta 1, encuentra un asiento disponible y lo bloquea para evitar que dos usuarios lo cojan a la vez. Para ello, busca un asiento (seat_no) en el avión correcto (aircraft_code = ?) que no esté en la lista de boarding_passes para ese vuelo (flight_id = ?). Con el ORDER BY s.seat_no ASC LIMIT 1 selecciona el primer asiento disponible (ej: '1A'). FOR UPDATE SKIP LOCKED es la parte más importante para la concurrencia ya que bloquea la fila del asiento que ha elegido (FOR UPDATE) para que nadie más pueda usarlo durante la transacción. Si un asiento ya está bloqueado por otro usuario, lo salta (SKIP LOCKED) y prueba con el siguiente.

```
SELECT s.seat_no FROM seats AS s
WHERE s.aircraft_code = ? AND s.seat_no NOT IN (
  SELECT bp.seat_no FROM boarding_passes AS bp WHERE bp.flight_id = ?
) ORDER BY s.seat_no ASC LIMIT 1
FOR UPDATE SKIP LOCKED
```

La tercera subconsulta genera un nuevo número de embarque (boarding_no) secuencial para ese vuelo. Esto se consigue encontrando el número más alto (MAX) ya asignado en ese vuelo y le suma 1.

```
SELECT COALESCE(MAX(boarding_no), 0) + 1 FROM boarding_passes WHERE flight_id = ?
```

La cuarta subconsulta crea la tarjeta de embarque insertando un nuevo registro en boarding_passes utilizando todos los datos recopilados en los pasos anteriores: el ticket_no (de la Consulta 1), el flight_id (Consulta 1), el boarding_no (Consulta 3) y el seat_no (Consulta 2).

```
INSERT INTO boarding_passes (ticket_no, flight_id, boarding_no, seat_no) VALUES (?, ?, ?, ?)
```

Aunque la implementación en C de results_bpass y results_search es parecida (se siguen usando las funciones de SQLExecute, SQLPrepare… de una misma manera), la de results_bpass es más compleja ya que se abandona el modo SQL_AUTOCOMMIT por defecto y se gestiona una transacción manualmente. La función comienza desactivando SQL_AUTOCOMMIT. Esto permite que el programa ejecute múltiples operaciones en la base de datos (encontrar un billete, encontrar un asiento, insertar la tarjeta) como un único "bloque". Si algún paso de este bloque falla (por ejemplo, el vuelo está lleno), toda la transacción puede ser revertida (ROLLBACK), asegurando que no se generen tarjetas de embarque para una misma reserva. Se utiliza una variable flag para rastrear el éxito de la transacción.

```
/* Desactivamos autocommit para controlar la transacción manualmente */
ret = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);
if (!SQL_SUCCEEDED(ret))
{
    fprintf(stderr, "Error desactivando autocommit.\n");
    odbc_disconnect(env, dbc);
    return;
}
```

Una vez conectada, ejecuta la primera consulta (stmt_tickets) para obtener la lista de billetes. Luego entra en un bucle while que procesa cada billete individualmente. Dentro del bucle, se ejecuta la segunda consulta (stmt_seat) para encontrar y bloquear un asiento. Si esta consulta devuelve SQL_NO_DATA (indicando que el vuelo está lleno), se establece flag = 0, se guarda un mensaje de error y el bucle se rompe. Si se encuentra un asiento, se procede a ejecutar stmt_bno y stmt_insert. Si cualquiera de estas operaciones falla, el flag también se pone a 0. Si el INSERT es exitoso, se formatea la línea de resultado (con el nombre, vuelo y asiento) y se almacena en el array choices. Una vez que el bucle termina, se comprueba el flag de la transacción. Si flag es 1 (éxito), se llama a SQLEndTran con SQL_COMMIT para hacer permanentes todos los cambios en la base de datos. Si flag es 0 (error), se llama a SQLEndTran con SQL_ROLLBACK, revirtiendo todas las inserciones que se hayan podido realizar. Finalmente, se liberan los cuatro handles de sentencias, se reactiva SQL_AUTOCOMMIT y se cierra la conexión.

```
while (flag && SQL_SUCCEEDED(ret = SQLFetch(stmt_tickets)) && i < max_rows)
{
    /* Consulta 2 (Encontrar asiento) */
    SQLBindParameter(stmt_seat, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR, 3, 0, aircraft_code, 0, &ind_ac);
    SQLBindParameter(stmt_seat, 2, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, &flight_id, 0, &ind_fid);
    SQLBindCol(stmt_seat, 1, SQL_C_CHAR, seat_no, sizeof(seat_no), &ind_seat);

    ret = SQLExecute(stmt_seat);
    if (!SQL_SUCCEEDED(ret))
    {
        snprintf((*choices)[0], max_length, "Error BD al ejecutar búsqueda de asiento");
        flag = 0;
        SQLCloseCursor(stmt_seat);
        break;
    }
    ret = SQLFetch(stmt_seat);
    if (ret == SQL_NO_DATA)
    {
        snprintf((*choices)[0], max_length, "Error: Vuelo lleno (ID: %d)", (int)flight_id);
        flag = 0;
        SQLCloseCursor(stmt_seat);
        break;
    }
    if (!SQL_SUCCEEDED(ret))
    {
        snprintf((*choices)[0], max_length, "Error BD al buscar asiento (fetch)");
        flag = 0;
        SQLCloseCursor(stmt_seat);
        break;
    }
    SQLCloseCursor(stmt_seat);

    /* Consulta 3 (Generar boarding_no) */
    SQLBindParameter(stmt_bno, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, &flight_id, 0, &ind_fid);
    SQLBindCol(stmt_bno, 1, SQL_C_LONG, &new_boarding_no, sizeof(new_boarding_no), &ind_bno);

    ret = SQLExecute(stmt_bno);
    if (!SQL_SUCCEEDED(ret))
    {
        snprintf((*choices)[0], max_length, "Error BD al generar boarding_no");
        flag = 0;
        SQLCloseCursor(stmt_bno);
        break;
    }
    ret = SQLFetch(stmt_bno);
    if (!SQL_SUCCEEDED(ret))
    {
        snprintf((*choices)[0], max_length, "Error BD al generar boarding_no");
        flag = 0;
        SQLCloseCursor(stmt_bno);
        break;
    }
    SQLCloseCursor(stmt_bno);
```

Aquí se encuentra la primera parte de bucle.

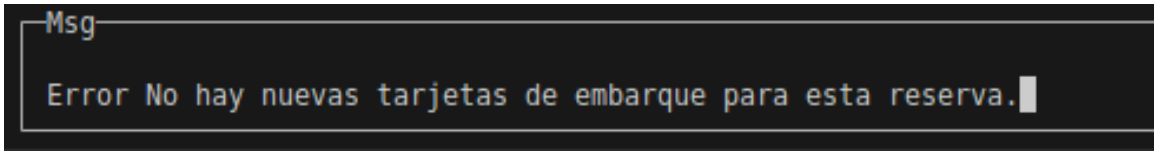
```
    /* Ejecución Consulta 4 (Insertar) */
    SQLBindParameter(stmt_insert, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR, 13, 0, ticket_no, 0, &ind_tk);
    SQLBindParameter(stmt_insert, 2, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, &flight_id, 0, &ind_fid);
    SQLBindParameter(stmt_insert, 3, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, &new_boarding_no, 0, &ind_bno);
    SQLBindParameter(stmt_insert, 4, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR, 4, 0, seat_no, 0, &ind_seat);

    ret = SQLExecute(stmt_insert);
    if (!SQL_SUCCEEDED(ret))
    {
        snprintf((*choices)[0], max_length, "Error BD al insertar la tarjeta embarque");
        flag = 0;
        break;
    }

    /* Éxito para este billete: Formatear salida */
    snprintf((*choices)[i], max_length, "Nombre: %-20.20s | Vuelo: %-6d | Salida: %-19s | Asiento: %-4s",
             (char *)passenger_name,
             (int)flight_id,
             (char *)scheduled_departure,
             (char *)seat_no);
    i++;
}

if (flag == 1)
{
    /* Éxito: Confirmar transacción */
    SQLEndTran(SQL_HANDLE_DBC, dbc, SQL_COMMIT);
    if (i == 0)
    {
        snprintf((*choices)[0], max_length, "Error No hay nuevas tarjetas de embarque para esta reserva.");
        *n_choices = 1;
    }
    else
    {
        *n_choices = i;
    }
}
else
{
    /* Error: Revertir transacción */
    SQLEndTran(SQL_HANDLE_DBC, dbc, SQL_ROLLBACK);
    *n_choices = 1;
}
```

```
/* Reactivar Autocommit */
SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_ON, 0);
odbc_disconnect(env, dbc);
```

Aquí vemos la segunda parte del bucle y la verificación del flag para ver si si hace commit o rollback. Por último se reactiva el autocommit.

Esta función encuentra todos los billetes sin asignar y crea con todas las tarjetas de embarque correspondientes, actualizando la base de datos.

Se puede ver que se actualiza correctamente la base de datos cuando se vuelve a buscar el mismo book_Id porque todos los asientos de los vuelos con ese book_Id tendrían ya su tarjeta de embarque, y por tanto no se devolvería ningún asiento.



3. Paginación

Para este apartado se ha tenido que cambiar el loop.c para integrar la lógica de pasar página y así mostrar en cada página un número fijo (en este caso 8) en cada página. La implementación se centra en el manejo de nuevas variables dentro de loop.c: current_page (para rastrear la página visible), total_pages (número total de páginas) y out_highlight (la posición del cursor en la lista total de resultados).

Cuando una función de búsqueda (results_search o results_bpass) devuelve un número de resultados (n_out_choices), la lógica calcula inmediatamente el total_pages, (n_out_choices + tam - 1) / tam. Despues, se inicializa current_page = 0 y se muestra la primera página de resultados en la out_window usando print_out.

El switch principal de loop.c también se ha modificado para usar las teclas KEY_NPAGE (Avance Página) y KEY_PPAGE (Retroceso Página). Cuando se detecta una de estas teclas (y el foco está en FOCUS_RIGHT), el código comprueba si el movimiento es válido. Si lo es, actualiza el current_page, recalcula los índices de inicio (start) y fin (end) del array de resultados, y redibuja la ventana out_window (print_out) mostrando únicamente el subconjunto de resultados que corresponde a la nueva página.

Por ejemplo, el KEY_PPAGE:

```
case KEY_NPAGE:
    if (focus == FOCUS_RIGHT && total_pages > 1 && current_page < total_pages - 1)
    {
        current_page++;
        start = current_page * N_ROWS_PAGE;
        end = MIN(start + N_ROWS_PAGE, n_out_choices);
        if (out_highlight < start)
            out_highlight = start;
        else if (out_highlight >= end)
            out_highlight = end - 1;
        wclear(out_win);
        print_out(out_win, &menus->out_win_choices[start], end - start, out_highlight - start, windows->out_title);
        snprintf(buffer, sizeof(buffer), "Página %d/%d", current_page + 1, total_pages);
        write_msg(msg_win, buffer, -1, -1, windows->msg_title);
    }
    break;
```

Además, la lógica de KEY_UP y KEY_DOWN se ha integrado con este sistema. Si el usuario mueve el cursor (out_highlight) más allá del límite superior o inferior de la página actualmente visible, current_page se actualiza automáticamente, forzando el redibujado de la página anterior o siguiente. En cada cambio de página, la ventana de mensajes (msg_win) se actualiza para informar al usuario del estado actual (ej: "Página 2/5"). Esta implementación la observamos aquí:

```
    else if (focus == FOCUS_RIGHT && n_out_choices > 0)
    {
        out_highlight = MIN(out_highlight + 1, n_out_choices - 1);

        /* Si el cursor se sale por abajo de la página actual, avanza página */
        end = MIN((current_page * N_ROWS_PAGE) + N_ROWS_PAGE, n_out_choices);
        if (out_highlight >= end && current_page < total_pages - 1)
        {
            current_page++;
        }

        start = current_page * N_ROWS_PAGE;
        end = MIN(start + N_ROWS_PAGE, n_out_choices);
        wclear(out_win);
        print_out(out_win, &menus->out_win_choices[start], end - start, out_highlight - start, windows->out_title);
        snprintf(buffer, sizeof(buffer), "Página %d/%d", current_page + 1, total_pages);
        write_msg(msg_win, buffer, -1, -1, windows->msg_title);
    }
    break;
```

La segunda parte del bucle principal del loop.c gestiona el estado de la aplicación. Parte del bloque if (choice != -1 && enterKey). Esta sección es la encargada de ejecutar las acciones y solo se activa cuando el switch anterior detecta que el usuario ha pulsado la tecla 'Enter' (case 10:). La lógica aquí se divide en cuatro escenarios principales, que dependen de la combinación de dos variables de estado: choice (SEARCH o BPASS) y focus (la ventana activa, FOCUS_LEFT para el formulario o FOCUS_RIGHT para los resultados).

Si el choice es SEARCH y el focus está en el formulario (FOCUS_LEFT), se llama a la función results_search para ejecutar la consulta SQL y rellenar la ventana de resultados. Por otro lado, si el foco está en los resultados (FOCUS_RIGHT), se activa la funcionalidad de "ver detalles": el código toma la línea seleccionada (menus->out_win_choices[out_highlight]), utiliza sscanf para extraer los datos "ocultos" (flight_id, aircraft_code) que search.c guardó tras el literal | INFO:, y muestra estos detalles en la message_window inferior.

Si el choice es BPASS y el focus está en el formulario (FOCUS_LEFT), se llama a la función results_bpass para ejecutar la transacción que genera las tarjetas de embarque. Finalmente, si el foco está en los resultados de BPASS (FOCUS_RIGHT), la lógica simplemente toma la cadena de la tarjeta de embarque seleccionada y la muestra en la message_window.

Por ejemplo, para la funcion SEARCH:

```
if (choice != -1 && enterKey)
{
    /*Variables para gestión de memoria*/
    initial_capacity = windows->rows_out_win;
    line_length = windows->cols_out_win - 4;

    if (choice == QUIT)
        break;

    /* SEARCH desde form window */
    if (choice == SEARCH && focus == FOCUS_LEFT)
    {
        if (menus->out_win_choices != NULL)
        {
            for (i = 0; i < out_win_capacity; i++)
            {
                if ((menus->out_win_choices)[i] != NULL)
                {
                    free((menus->out_win_choices)[i]);
                }
            }
            free(menus->out_win_choices);
        }
        menus->out_win_choices = (char **)calloc(initial_capacity, sizeof(char *));
        if (menus->out_win_choices == NULL)
            break;
        for (i = 0; i < initial_capacity; i++)
        {
            (menus->out_win_choices)[i] = (char *)calloc(line_length, sizeof(char));
            if ((menus->out_win_choices)[i] == NULL)
                break;
        }
        out_win_capacity = initial_capacity;
        n_out_choices = 0;
        out_highlight = 0;
        wclear(out_win);
        form_driver(forms->search_form, REQ_VALIDATION);
        tmpStr1 = field_buffer((forms->search_form_items)[1], 0);
        results_search(tmpStr1,
                       field_buffer((forms->search_form_items)[3], 0),
                       field_buffer((forms->search_form_items)[5], 0),
                       &n_out_choices, &(menus->out_win_choices),
                       line_length, initial_capacity);

        if (n_out_choices > 0 &&
            (strncmp(menus->out_win_choices[0], "Error", 5) == 0))
        {
            write_msg(msg_win, menus->out_win_choices[0], -1, -1, windows->msg_title);
            n_out_choices = 0;
        }

        if (n_out_choices > 0)
        {
            total_pages = (n_out_choices + N_ROWS_PAGE - 1) / N_ROWS_PAGE;
            current_page = 0;
            out_highlight = 0;
            start = current_page * N_ROWS_PAGE;
            end = MIN(start + N_ROWS_PAGE, n_out_choices);
            wclear(out_win);
            print_out(out_win, &menus->out_win_choices[start], end - start, out_highlight - start, windows->out_title);
        }

        if (n_out_choices > out_win_capacity)
            out_win_capacity = n_out_choices;
    }
```

```c
/*SEARCH desde out window */
else if (choice == SEARCH && focus == FOCUS_RIGHT)
{
    if (n_out_choices == 0)
    {
        choice = -1;
        enterKey = FALSE;
        continue;
    }
    linea = menus->out_win_choices[out_highlight];
    if (out_highlight == 0 &&
        (strncmp(linea, "Error", 5) == 0))
    {
        write_msg(msg_win, linea, -1, -1, windows->msg_title);
    }
    else
    {
        /* sscanf que COINCIDE con el snprintf de search.c */
        int n = sscanf(linea,
                       "Salida: %29[^|] | Llegada: %29[^|] | Vuelos: %d | Plazas: %d | INFO: %ld %9s %ld %9s",
                       salida_str,
                       llegada_str,
                       &num_vuelos,
                       &num_plazas,
                       &flight_id1,
                       ac1_str,
                       &flight_id2,
                       ac2_str);
        /* comprueba si leyó 8 elementos */
        if (n < 8)
        {
            snprintf(buffer, sizeof(buffer), "Error: no se pudo leer la línea. (n=%d)", n);
            write_msg(msg_win, buffer, -1, -1, windows->msg_title);
        }
        else
        {
            /* muestra los datos */
            if (flight_id2 == -1)
            {
                /* Vuelo directo */
                snprintf(buffer, sizeof(buffer),
                         "DETALLES: Vuelo ID %ld (Aircraft_code %s). Salida: %s. Llegada: %s",
                         flight_id1, ac1_str, salida_str, llegada_str);
            }
            else
            {
                /* Vuelo con escala */
                snprintf(buffer, sizeof(buffer),
                         "DETALLES: Vuelo1(ID %ld, Aircraft_code %s) -> Vuelo2(ID %ld, Aircraft_code %s). Salida: %s. Llegada: %s",
                         flight_id1, ac1_str, flight_id2, ac2_str, salida_str, llegada_str);
            }
            write_msg(msg_win, buffer, -1, -1, windows->msg_title);
        }
    }
}
```

Por último hemos añadido el valgrind.log para ver los errores de la práctica. Se puede observar que tal como dice el enunciado, existen errores pero no atribuibles al trabajo propio ya que no hay ningún byte definitely lost e indirectly lost.