19955 - Fundamentos de Bases de Datos

Universidad Autónoma de Madrid

Grupo 1273 - Práctica 2

Profesora: Julia Díaz

Realizado por:

Carlos César Rodríguez García

Rodrigo Alejandro Yáñez García

15 de noviembre del 2023

1. Conexión a la Base de Datos

Se establece una única conexión a la base de datos que perdura a lo largo de todo el ciclo de vida de la aplicación. Esta conexión se gestiona en la función principal (*main*) del archivo `*menu.c*` y se utiliza para crear los *prepared statements*, los cuales son posteriormente compartidos con los diversos módulos del proyecto.

```
// menu.c
int main() {
     ...
      _PreparedStatements statements;

      SQLHENV env;
      SQLHDBC dbc;
      SQLRETURN ret; /* ODBC API return status */

     ...

      /* create connection to db */
      ret = odbc_connect(&env, &dbc);
      if (!SQL_SUCCEEDED(ret)) {
            write_error(windows.msg_win, "could not connect to
database");
      }

      init_statements(&statements, dbc);
      init_create_boarding_passes_function(dbc);

      /* process keyboard */
      loop(&windows, &menus, &forms, &panels, &statements);

     ...
}
```

1.1. Prepared statements

Los *prepared statements* ofrecen la posibilidad de crear consultas parametrizadas y dinámicas, al mismo tiempo que proporcionan una capa de seguridad contra las inyecciones de SQL. En nuestra aplicación, todas las consultas se realizan mediante este enfoque, permitiendo su reutilización para cada instancia de consulta. Para una gestión más eficiente, hemos creado una estructura denominada

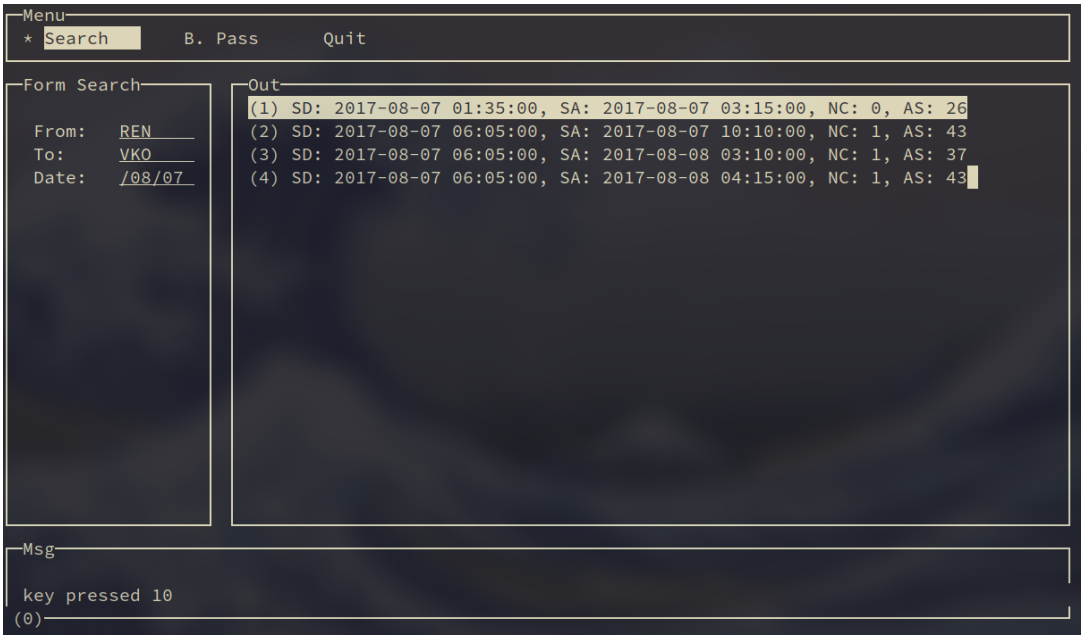
PreparedStatements, la cual almacena los statements asociados a las consultas utilizadas en cada caso de uso y se pasa como argumento a la función *loop* (donde se encuentra el manejo de la interacción con la interfaz).

```
// windows.h
typedef struct _PreparedStatements {
	SQLHSTMT flight_connections; /* gets flights between two airports
*/
	SQLHSTMT flights_details; /* retrieves details of a flight */
	SQLHSTMT booking_check; /* checks if a booking exists with a given
book_ref */
	SQLHSTMT created_boarding_passes; /* queries the boarding passes
that were just assigned */
} _PreparedStatements;
```

2. Realización de búsquedas.

2.1. Función “results_search”

Esta función contiene la lógica para encontrar todos los vuelos entre dos aeropuertos dada una fecha. Estos vuelos pueden tener una conexión como máximo y el tiempo total de viaje no puede superar 24 hrs.



a. **Preparación y validación de parámetros:**

- Los parámetros *from*, *to*, y *date* se truncan para eliminar espacios adicionales al final.
- Se verifica si alguno de los campos requeridos está vacío. En caso afirmativo, se muestra un mensaje de error que indica los campos faltantes.

```
char result[512], missing_fields[124] = "the following fields are
missing: ";

trim_trailing(from);
trim_trailing(to);
trim_trailing(date);

/* Check if any field is missing */
if (strlen(from) == 0 || strlen(to) == 0 || strlen(date) == 0) {
    if (strlen(from) == 0) {
        strcat(missing_fields, " `from`");
    }

    if (strlen(to) == 0) {
        strcat(missing_fields, " `to`");
    }

    if (strlen(date) == 0) {
        strcat(missing_fields, " `date`");
        }

    write_error(msg_win, missing_fields);
    return;
}
```

b. **Vinculación de parámetros y ejecución de consulta:**

- Los parámetros se vinculan a la consulta reutilizada SQL (*stmt*) utilizando la función *SQLBindParameter*.
- Se ejecuta la consulta SQL utilizando *SQLExecute*.

```
/* Bind parameters */
    SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
```

```
sizeof(from), 0, from, sizeof(from), NULL);
    SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
sizeof(to), 0, to, sizeof(to), NULL);
    SQLBindParameter(stmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
sizeof(date), 0, date, sizeof(date), NULL);
    SQLBindParameter(stmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
sizeof(from), 0, from, sizeof(from), NULL);
    SQLBindParameter(stmt, 5, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
sizeof(to), 0, to, sizeof(to), NULL);
    SQLBindParameter(stmt, 6, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
sizeof(date), 0, date, sizeof(date), NULL);

    if (SQL_SUCCEEDED(SQLExecute(stmt))) {
        SQLBindCol(stmt, 1, SQL_C_CHAR, scheduled_departure,
sizeof(scheduled_departure), NULL);
        SQLBindCol(stmt, 2, SQL_C_CHAR, scheduled_arrival,
sizeof(scheduled_arrival), NULL);
        SQLBindCol(stmt, 3, SQL_INTEGER, &n_connections,
sizeof(n_connections), NULL);
        SQLBindCol(stmt, 4, SQL_INTEGER, &n_availabe_seats,
sizeof(n_availabe_seats), NULL);
        SQLBindCol(stmt, 5, SQL_INTEGER, &flight_id_1,
sizeof(flight_id_1), NULL);
        SQLBindCol(stmt, 6, SQL_INTEGER, &flight_id_2,
sizeof(flight_id_2), NULL);

...
```

c. Vinculación de columnas y procesamiento de resultados:

- Se vinculan las columnas de la consulta a variables C para procesar los resultados.
- Se recorren los resultados utilizando *SQLFetch* y se almacenan en un arreglo de opciones *choices*, cada una formateada según las columnas seleccionadas.

```
/* Fetch and process the results */
  *n_choices = 0;
  while (SQL_SUCCEEDED(SQLFetch(stmt))) {
        if (*n_choices < max_rows) {
            /* Allocate memory for the current choice */
            (*choices)[*n_choices] = (char*)malloc(max_length *
```

```
sizeof(char));
            search_flight_ids_1[*n_choices] =
(char*)malloc(sizeof(flight_id_1));

            /* Capture and format result */
            sprintf(result, "(%d) SD: %s, SA: %s, NC: %d, AS: %d",
(*n_choices)+1, scheduled_departure, scheduled_arrival, n_connections,
n_availabe_seats);

...
```

d. Almacenamiento de IDs de vuelo:

- A través de las columnas vinculadas, se almacenan los IDs de vuelo en los arreglos *search_flight_ids_1* y, opcionalmente, *search_flight_ids_2* (para aquellos vuelos con conexión) con el objetivo de desplegarlos y manipularlos en el futuro.

```
...

/* Allocate memory for the current choice */
(*choices)[*n_choices] = (char*)malloc(max_length * sizeof(char));
search_flight_ids_1[*n_choices] = (char*)malloc(sizeof(flight_id_1));

/* Capture and format result */
sprintf(result, "(%d) SD: %s, SA: %s, NC: %d, AS: %d", (*n_choices)+1,
scheduled_departure, scheduled_arrival, n_connections,
n_availabe_seats);

/* Use proper indexing and dereferencing for choices */
write_choice(result, choices, (*n_choices), max_length);
sprintf(search_flight_ids_1[*n_choices], "%d", flight_id_1);

if (flight_id_2 != SQL_NULL_DATA) {
search_flight_ids_2[*n_choices] = (char*)malloc(sizeof(flight_id_2));
sprintf(search_flight_ids_2[*n_choices], "%d", flight_id_2);

...
```

e. **Manejo de errores y mensajes:**

Se gestionan tanto los escenarios de éxito como los de error de la consulta, proporcionando mensajes pertinentes para el usuario a través de la ventana `message window`. Los errores considerados incluyen:

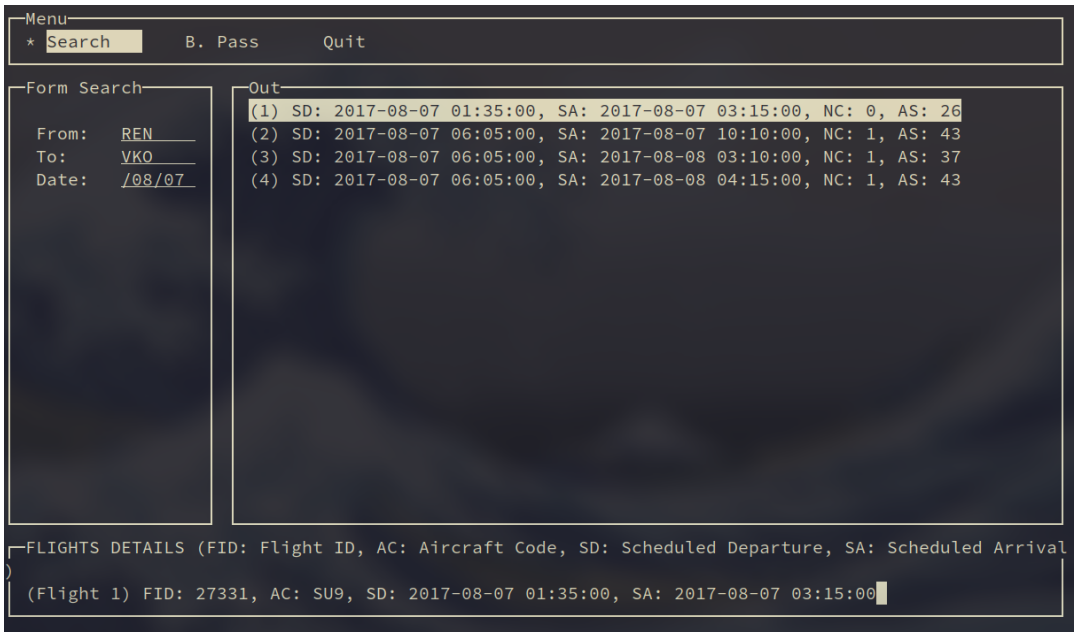
- Ausencia de alguno de los campos (“the following fields are missing”).

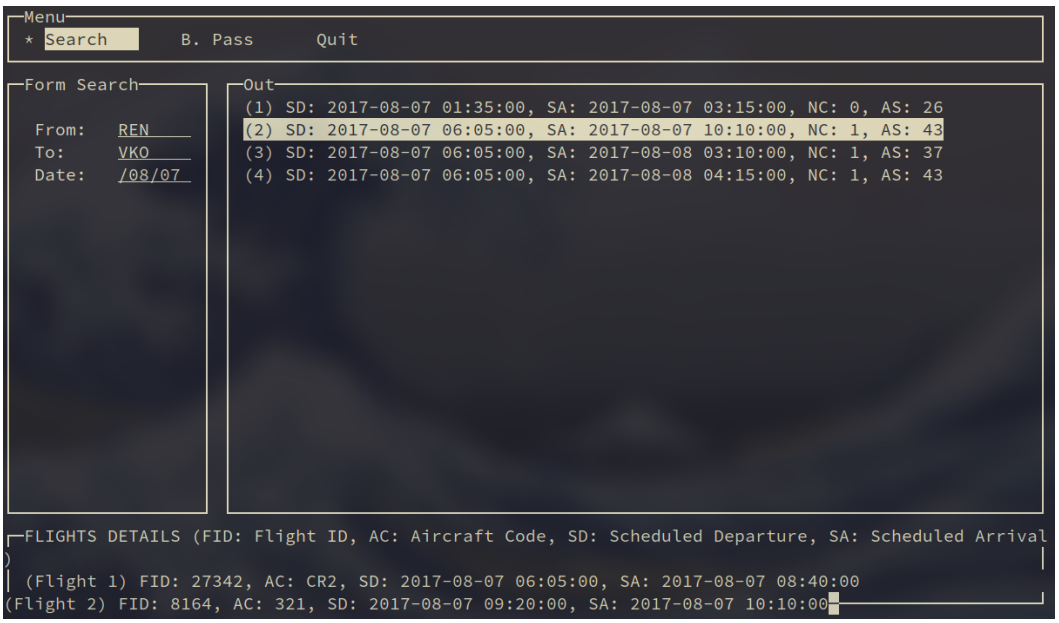
- No hay vuelos con los datos proporcionados (“no flights found with the given data”).

- Error ejecutando la consulta (“error executing query”).

2.2. Función “flight_details”

Se encarga de desplegar el detalle de los vuelos encontrados al dar enter sobre una de sus filas. Despliega ya sea para uno o dos vuelos (si hay conexión), su código de identificación, el código de la aeronave, el tiempo de despegue y aterrizaje.





a. Ejecución de la consulta de detalles de vuelo:

- Se preparan y ejecutan consultas SQL para obtener detalles específicos de los vuelos identificados por *flight_id_1* y, opcionalmente, *flight_id_2*.

```
SQLCHAR aircraft_code[8], scheduled_departure[64],
scheduled_arrival[64];
char result[1024], temp[512];
int i = 0;

SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 0, 0,
flight_id_1, 0, NULL);
SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 0, 0,
(flight_id_2[0] != '\0' ? flight_id_2 : "-1"), 0, NULL);

SQLExecute(stmt);
```

b. Vinculación de columnas y construcción de resultados:

- Se vinculan las columnas de la consulta a variables C para procesar los detalles del vuelo.
- Se construye una cadena de resultados formateada con detalles de los vuelos encontrados.

```
SQLBindCol(stmt, 1, SQL_C_CHAR, aircraft_code, sizeof(aircraft_code),
NULL);
SQLBindCol(stmt, 2, SQL_C_CHAR, scheduled_departure,
sizeof(scheduled_departure), NULL);
SQLBindCol(stmt, 3, SQL_C_CHAR, scheduled_arrival,
sizeof(scheduled_arrival), NULL);
```

c. Manejo de resultados y visualización:

- Se manejan los casos de éxito y se visualizan los detalles de los vuelos en la ventana *msg_win*.

```
result[0] = '\0';
while (SQL_SUCCEEDED(SQLFetch(stmt))) {
    sprintf(temp, "(Flight %d) FID: %s, AC: %s, SD: %s, SA: %s\n", i +
1, (i == 0) ? flight_id_1 : flight_id_2, aircraft_code,
scheduled_departure, scheduled_arrival);
    strcat(result, temp);
    i++;
}

if (result[0] != '\0') {
    result[strlen(result) - 1] = '\0';
    write_msg(msg_win, result, -1, -1, "FLIGHTS DETAILS (FID: Flight ID,
AC: Aircraft Code, SD: Scheduled Departure, SA: Scheduled Arrival)");
} else {
    write_error(msg_win, "no results found");
}

SQLCloseCursor(stmt);
```

Observaciones generales:

- Se utiliza la biblioteca ODBC para interactuar con la base de datos SQL.
- Se realiza una validación inicial de los campos obligatorios antes de ejecutar la consulta.
- La función *write_choice* se utiliza para formatear y almacenar resultados en la interfaz gráfica.
- Se utiliza la función *write_msg* para mostrar mensajes en la ventana *msg_win*.
- El código utiliza la función *SQLCloseCursor* para cerrar el cursor después de procesar los resultados.

Comandos SQL utilizados:

a. statements -> flight_connections

```
SELECT * FROM
(
    -- Búsqueda Directa
    SELECT
        f.scheduled_departure AS scheduled_departure,
        f.scheduled_arrival AS scheduled_arrival,
        0 AS no_connections,
        COUNT(s.seat_no) AS seats_available,
        f.flight_id AS flight_id_1,
        -1 AS flight_id_2
    FROM
        flights f
    JOIN seats s ON f.aircraft_code = s.aircraft_code
    LEFT JOIN boarding_passes bp ON f.flight_id = bp.flight_id AND
s.seat_no = bp.seat_no
    WHERE
        f.departure_airport = ?
        AND f.arrival_airport = ?
        AND DATE(f.scheduled_departure) = ?
        AND bp.ticket_no IS NULL
    GROUP BY f.flight_id

    UNION

    -- Búsqueda con Conexiones
    SELECT
        f1.scheduled_departure AS scheduled_departure,
        f2.scheduled_arrival AS scheduled_arrival,
        1 AS no_connections,
        LEAST(COUNT(DISTINCT s1.seat_no), COUNT(DISTINCT s2.seat_no)) AS
seats_available,
        f1.flight_id AS flight_id_1,
        f2.flight_id AS flight_id_2
    FROM
        flights f1
    JOIN flights f2 ON f1.arrival_airport = f2.departure_airport
    JOIN seats s1 ON f1.aircraft_code = s1.aircraft_code
    JOIN seats s2 ON f2.aircraft_code = s2.aircraft_code
    WHERE
        s1.seat_no NOT IN (
```

```
            SELECT seat_no
            FROM boarding_passes
            WHERE flight_id = f1.flight_id
        )
        AND s2.seat_no NOT IN (
            SELECT seat_no
            FROM boarding_passes
            WHERE flight_id = f2.flight_id
        )
        AND f1.scheduled_arrival < f2.scheduled_departure
        AND EXTRACT(EPOCH FROM (f2.scheduled_departure -
f1.scheduled_arrival)) / 3600 <= 24
        AND f1.departure_airport = ?
        AND f2.arrival_airport = ?
        AND DATE(f1.scheduled_departure) = ?
    GROUP BY f1.departure_airport, f2.arrival_airport,
f1.scheduled_departure, f2.scheduled_arrival
) AS combined_result

WHERE seats_available > 0
ORDER BY scheduled_arrival - scheduled_departure ASC;
```

La lógica implementada en este código está diseñada para buscar vuelos disponibles en función de ciertos criterios, como el aeropuerto de salida, el aeropuerto de llegada y la fecha de salida. La búsqueda se realiza en dos partes:

1. **Búsqueda Directa (sin conexiones).**
    - Selecciona vuelos directos que coincidan con los criterios de salida, llegada y fecha.
    - Se cuenta el número de asientos disponibles en esos vuelos.
    - Se utiliza una subconsulta *LEFT JOIN* para verificar si hay pases de abordar (*boarding_passes*) asociados a esos asientos y vuelos. Si el pase de abordar es *null*, entonces el asiento está disponible.
    - Los resultados se agrupan por el ID del vuelo (*flight_id*).

2. **Búsqueda de Conexiones.**
    - Selecciona vuelos que tienen una conexión intermedia. Es decir, un vuelo que llega a un aeropuerto y otro que sale de ese mismo aeropuerto en una fecha posterior.
    - Se verifica la disponibilidad de asientos en ambos vuelos, asegurándose de que los asientos no estén ocupados según los pases de abordar.
    - Se limita la diferencia de tiempo entre la llegada del primer vuelo y la salida del segundo a un máximo de 24 horas.
    - Los resultados se agrupan por los aeropuertos de salida y llegada del primer y segundo vuelo, así como por las fechas de salida y llegada respectivas.

Después de realizar estas dos búsquedas, los resultados se combinan utilizando *UNION* y se filtran para mostrar solo aquellos con asientos disponibles *(seats_available > 0)*. Finalmente, los resultados se ordenan de manera ascendente por la diferencia entre la hora de llegada programada y la hora de salida programada.

b. statements -> flight_details

```
SELECT aircraft_code, scheduled_departure, scheduled_arrival
FROM flights
WHERE flight_id = ? OR flight_id = ?
ORDER BY scheduled_departure ASC;
```

Este código sigue una lógica muy simple, y está implementado con el único objetivo de extraer los detalles solicitados para los itinerarios de vuelos seleccionados; todo bajo la siguiente estructura:

1. **Selección de columnas.**

La consulta selecciona tres columnas de la tabla *flights*: *aircraft_code*, *scheduled_departure*, y *scheduled_arrival*.

2. **Filtrado de filas.**

Se seleccionan las filas donde el valor de la columna *flight_id* es igual al valor proporcionado en los parámetros de la consulta. La consulta utiliza el operador OR para permitir que coincida con los valores de *flight_id* tanto para el vuelo 1 como para el vuelo 2 (en los casos donde existe conexión). Los valores de *flight_id* son parámetros de la consulta, indicados por los marcadores de posición ?.

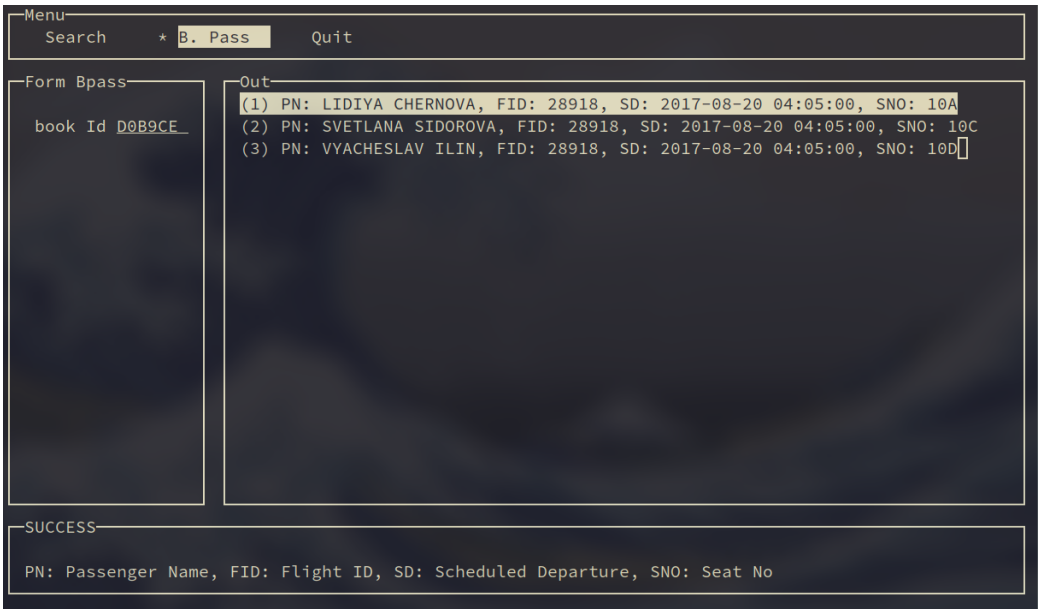
3. **Ordenación de resultados.**

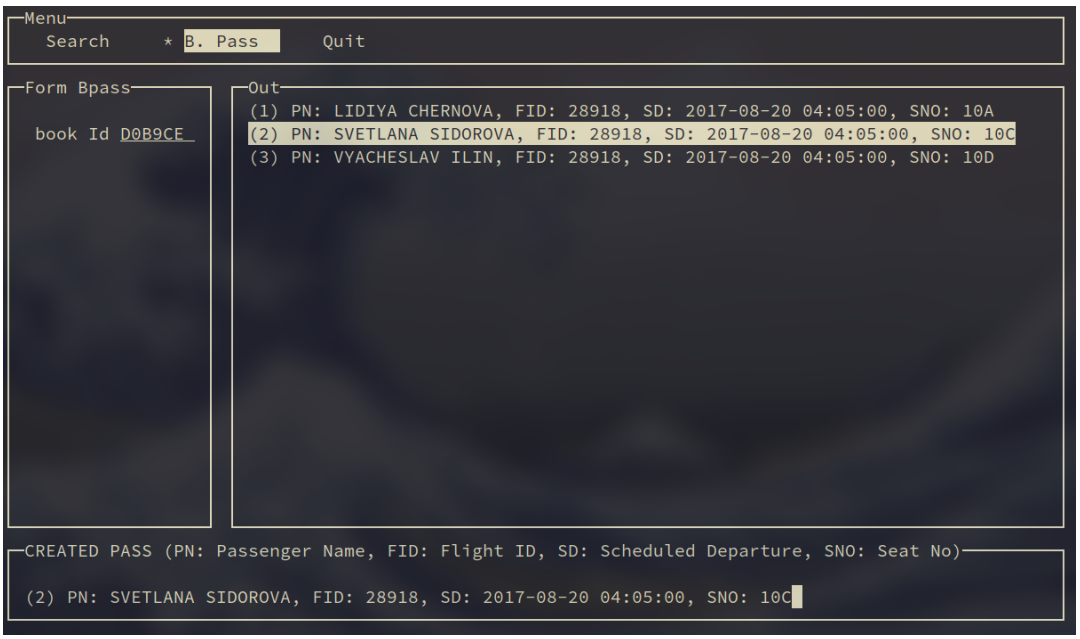
Finalmente se ordenan los resultados según la columna *scheduled_departure* en orden ascendente.

3. Emisión de tarjetas de embarque.

3.1. Función results_bpass

Dada la referencia a una reservación, crea todos los pases de embarque que hagan falta entre alguno de sus vuelos.





a. Entrada de datos:

- Se espera el *bookID* como un parámetro de formulario. El código verifica que este valor no esté vacío y lo limpia de espacios en blanco al final.

```
trim_trailing(bookID); /* Remove white spaces from `bookID` */

if (strlen(bookID) == 0) {
    write_error(msg_win, "`book Id` cannot be empty");
    return;
}
```

b. Verificación de existencia de reserva:

- Utiliza una consulta SQL preparada (*booking_stmt*) para verificar si existe una reserva con el ID proporcionado (*bookID*). La existencia se determina al contar las filas afectadas por la ejecución de la consulta.

```
SQLBindParameter(booking_stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, sizeof(bookID), 0, bookID, sizeof(bookID), NULL);
SQLExecute(booking_stmt);

SQLRowCount(booking_stmt, &row_count);
```

```
SQLCloseCursor(booking_stmt);
```

c. **Recuperación de pases de abordar creados:**

- Si se encuentra una reserva, utiliza otra consulta SQL preparada (*created_boarding_passes_stmt*) para recuperar los detalles de los pases de abordar creados asociados a esa reserva.

```
SQLBindParameter(created_boarding_passes_stmt, 1, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_VARCHAR, sizeof(bookID), 0, bookID,
sizeof(bookID), NULL);
SQLExecute(created_boarding_passes_stmt);

SQLBindCol(created_boarding_passes_stmt, 1, SQL_C_CHAR,
passenger_name, sizeof(passenger_name), NULL);
SQLBindCol(created_boarding_passes_stmt, 2, SQL_C_CHAR, flight_id,
sizeof(flight_id), NULL);
SQLBindCol(created_boarding_passes_stmt, 3, SQL_C_CHAR,
scheduled_departure, sizeof(scheduled_departure), NULL);
SQLBindCol(created_boarding_passes_stmt, 4, SQL_C_CHAR, seat_no,
sizeof(seat_no), NULL);
```

d. **Procesamiento de resultados:**

- Itera a través de los resultados y almacena la información relevante en el arreglo *choices* para presentarla posteriormente.
- Formatea los resultados, incluyendo el nombre del pasajero truncado, el ID del vuelo, la hora de salida programada y el número de asiento.

```
while (SQL_SUCCEEDED(SQLFetch(created_boarding_passes_stmt))) {
   if (*n_choices < max_rows) {
      /* Allocate memory for the current choice */
      (*choices)[*n_choices] = (char*)malloc(max_length *
sizeof(char));

      passenger_name[20] = '\0'; /* truncate the passenger name */

      /* Capture and format result */
      sprintf(result, "(%d) PN: %s, FID: %s, SD: %s, SNO: %s",
```

```
(*n_choices)+1, passenger_name, flight_id, scheduled_departure,
seat_no);

      /* Use proper indexing and dereferencing for choices */
       write_choice(result, choices, (*n_choices), max_length);
       (*n_choices)++;
    }
}
```

e. Manejo de errores y mensajes:

Se gestionan tanto los escenarios de éxito como los de error de la consulta, proporcionando mensajes pertinentes para el usuario a través de la ventana `message window`. Los errores considerados incluyen:

- Book_ID vacío (“book id cannot be empty”).

- No hay una reserva con el `book_ID` proporcionado.

- La reservación no tiene pases de abordar pendientes (“booking already has all boarding passes”).

Comandos SQL utilizados:

a. **statements -> create_boardin_passes_function**

```
CREATE OR REPLACE FUNCTION create_boarding_passes(book_ref_param TEXT)
RETURNS TABLE (
    passenger_name TEXT,
    flight_id INT,
    scheduled_departure TIMESTAMP WITH TIME ZONE,
    seat_no CHAR VARYING(4)
) AS $$
DECLARE
    ticket_flight_without_boarding_pass RECORD;
    available_seat_no CHAR VARYING(4);
    last_boarding_no INT;
BEGIN
    -- drop the results table if it exists
     IF EXISTS (SELECT FROM information_schema.tables WHERE table_name =
'results') THEN
    DROP TABLE results;
    END IF;

    -- temporary table to store the data of created boarding passes
    CREATE TEMPORARY TABLE results (
        passenger_name TEXT,
        flight_id INT,
        scheduled_departure TIMESTAMP WITH TIME ZONE,
        seat_no CHAR VARYING(4)
    );

    -- main iteration
    FOR ticket_flight_without_boarding_pass IN (
        -- query to get all flight tickets that do
        -- not have an associated boarding pass
        SELECT
            tf.flight_id,
            tf.ticket_no,
            t.passenger_name
        FROM
            ticket_flights tf
            NATURAL JOIN tickets t
            LEFT JOIN boarding_passes bp
                    ON bp.flight_id = tf.flight_id AND bp.ticket_no =
tf.ticket_no
```

```
        WHERE
             bp.flight_id IS NULL AND bp.ticket_no IS NULL AND t.book_ref
= book_ref_param
        ORDER BY tf.ticket_no ASC -- order according to instructions
    ) LOOP
        -- query to get the first available seat on the flight
        SELECT s.seat_no
        FROM seats s
        WHERE s.aircraft_code IN (
            SELECT f.aircraft_code
            FROM flights f
                                                WHERE    f.flight_id    =
ticket_flight_without_boarding_pass.flight_id
        )
        AND NOT EXISTS(
            -- exclude all seats that are already assigned
            SELECT 1
            FROM boarding_passes bp
                                               WHERE    bp.flight_id    =
ticket_flight_without_boarding_pass.flight_id
            AND bp.seat_no = s.seat_no
        )
        ORDER BY s.seat_no ASC, s.aircraft_code ASC
        LIMIT 1 INTO available_seat_no;

        -- query to get the last boarding pass number
        -- associated with the flight
        SELECT COALESCE(MAX(boarding_no), 0)
        INTO last_boarding_no
        FROM boarding_passes bp
                                             WHERE      bp.flight_id      =
ticket_flight_without_boarding_pass.flight_id;

        -- create new boarding pass assigning the found
        -- available seat to the ticket flight
         INSERT INTO boarding_passes (ticket_no, flight_id, boarding_no,
seat_no)
        VALUES (
            ticket_flight_without_boarding_pass.ticket_no,
            ticket_flight_without_boarding_pass.flight_id,
            last_boarding_no + 1,
            available_seat_no
        );

        -- save created boarding pass into results table
                    INSERT   INTO   results   (passenger_name,   flight_id,
```

```
scheduled_departure, seat_no)
            SELECT
                t.passenger_name,
                tf.flight_id,
                f.scheduled_departure,
                bp.seat_no
            FROM
                tickets t
                NATURAL JOIN ticket_flights tf
                NATURAL JOIN flights f
                NATURAL JOIN boarding_passes bp
            WHERE
                                                        tf.ticket_no    =
ticket_flight_without_boarding_pass.ticket_no    AND    tf.flight_id    =
ticket_flight_without_boarding_pass.flight_id
            LIMIT 1;
    END LOOP;

    -- return the contents of the results table
    RETURN QUERY SELECT * FROM results;
END;

$$ LANGUAGE plpgsql;
```

Esta función almacenada tiene el nombre *create_boarding_passes* y toma un parámetro *book_ref_param* de tipo texto. Su función es retornar una tabla con las columnas *passenger_name*, *flight_id*, *scheduled_departure*, y *seat_no*. Cabe señalar que los bloques DECLARE y BEGIN incluyen la lógica principal de esta función almacenada, iterando sobre los vuelos asociados a una reserva que aún no tienen una tarjeta de embarque, asignando asientos disponibles y almacenando la información de las tarjetas de embarque generadas en una tabla temporal llamada *results*, la cual regresa como resultado al final de la operación. El algoritmo que sigue dicha consulta es el siguiente:

1. **Comprobación de la existencia de 'results':**

Se verifica si existe una tabla temporal llamada *'results'*. En caso afirmativo, se elimina.

2. **Creación de la tabla temporal 'results':**

Si no existe la tabla temporal buscada en el paso anterior, entonces se crea una nueva de nombre *'results'* para almacenar temporalmente la información de las tarjetas de embarque creadas.

3. **Iteración principal:**

Para cada *ticket_flight_without_boarding_pass* (vuelo sin tarjeta de embarque):

- Se obtiene un asiento disponible para el vuelo.
- Se obtiene el último número de embarque asociado al vuelo.
- Se crea una nueva tarjeta de embarque asignando el asiento disponible al vuelo.
- La tarjeta de embarque creada se guarda en la tabla 'results'.

4. **Devolución de resultados:**

Se devuelve el contenido de la tabla 'results'.

b. statements -> booking_check

```
SELECT 1 FROM bookings WHERE book_ref = ? LIMIT 1;
```

Esta pequeña consulta SQL verifica si existe, o no, una reserva en la tabla *bookings* con el *book_ref* proporcionado. Se utiliza el valor de ? como un parámetro que se vinculará con el input del usuario.

c. statements -> created_boarding_passes

```
SELECT * FROM create_boarding_passes(?);
```

Para esta consulta SQL se utiliza la función almacenada *create_boarding_passes* con el objetivo de recuperar las tarjetas de embarque generadas para una reserva específica. De igual forma que la consulta anterior, se utiliza el valor de ? como un parámetro que se vinculará con el input del usuario (*book_ref*).