# ITERACIÓN 3

Para el cumplimiento de los nuevos requerimientos presentados en esta iteración no fue necesaria la modificación o adición de nuevos elementos al esquema relacional propuesto en la iteración 2. De forma que el modelo conceptual de la iteración 2 sigue siento completamente valido para esta iteración.

***Requerimientos de consulta***

Para poder cumplir con los requerimientos de consulta lo primero fue modificar SQLUtil para realizar ahí la injección de las sentencias SQL que nos permitieran obtener los datos solicitados. Estos nuevos metodos son:

```
public List<Analisis> analizarInformacion(PersistenceManager pm, int i) {
    Query q;
    if(i == 0) {
        q = pm.newQuery(SQL, "SELECT fecha, COUNT(*), SUM(precio) FROM reserva WHERE TO_DATE(fecha, 'yyyy-mm-dd') >= ADD_MONTHS(SYSDATE, -1) GROUP BY fecha ORDER BY COUNT(*) DESC, SUM(precio) DESC");
    }
    else {
        q = pm.newQuery(SQL, "SELECT fecha, COUNT(*), SUM(precio) FROM reserva WHERE TRUNC(TO_DATE(fecha, 'yyyy-mm-dd'), 'IW') = TRUNC(SYSDATE, 'IW') GROUP BY fecha ORDER BY COUNT(*) DESC, SUM(precio) DESC");
    }
    q.setResultClass(Object[].class);
    List<Object[]> results = (List<Object[]>) q.executeList();

    List<Analisis> analisisList = new ArrayList<>();

    for (Object[] result : results) {
        String fecha = (String) result[0];
        int count = ((Number) result[1]).intValue();
        int sum = ((Number) result[2]).intValue();
        analisisList.add(new Analisis(fecha, count, sum));
    }

    return analisisList;
}
```

```
public List<Usuario> darClientesFrecuentes(PersistenceManager pm, String tipo, Long pid) {
    Query q = pm.newQuery(SQL, "SELECT usuario.* FROM usuario INNER JOIN reserva r ON usuario.id = r.idusuario INNER JOIN alohamiento a ON r.idalohamiento = a.id WHERE a."+ tipo +" = "+ pid
            +" GROUP BY usuario.id, usuario.nombre, usuario.cedula, usuario.edad, usuario.celular, usuario.vinculacion HAVING COUNT(*) >= 3");
    q.setResultClass(Object[].class);
    List<Object[]> results = (List<Object[]>) q.executeList();

    List<Usuario> usuarioList = new ArrayList<>();

    for (Object[] result : results) {
        Long id = ((Number) result[0]).longValue();
        String nombre = (String) result[1];
        String cedula = (String) result[2];
        int edad = ((Number) result[3]).intValue();
        String celular = (String) result[4];
        String vinculacion = (String) result[5];
        usuarioList.add(new Usuario(id, nombre, cedula, edad, celular, vinculacion));
    }

    return usuarioList;
}
```

```
public List<OfertasPopulares> darOfertasSinDemanda(PersistenceManager pm) {
    Query q = pm.newQuery(SQL, "SELECT alohamiento.id FROM alohamiento LEFT JOIN reserva ON alohamiento.id = reserva.id WHERE reserva.fecha IS NULL OR TO_DATE(fecha, 'yyyy-mm-dd') >= ADD_MONTHS(SYSDATE, -1)");
    q.setResultClass(Object[].class);
    List<Object[]> results = (List<Object[]>) q.executeList();

    List<OfertasPopulares> ofertasList = new ArrayList<>();

    for (Object[] result : results) {
        Long id = ((Number) result[0]).longValue();
        ofertasList.add(new OfertasPopulares(id));
    }

    return ofertasList;
}
```

```
public List<UsoAlohandes> darUsoAlohandes(PersistenceManager pm, Long id) {
    Query q = pm.newQuery(SQL, "SELECT COUNT(*), SUM(precio) FROM RESERVA WHERE idusuario = " + id);
    q.setResultClass(Object[].class);
    List<Object[]> results = (List<Object[]>) q.executeList();

    List<UsoAlohandes> usoAlohandesList = new ArrayList<>();

    for (Object[] result : results) {
        int count = ((Number) result[0]).intValue();
        int sum = ((Number) result[1]).intValue();
        usoAlohandesList.add(new UsoAlohandes(count, sum));
    }

    return usoAlohandesList;
}
```

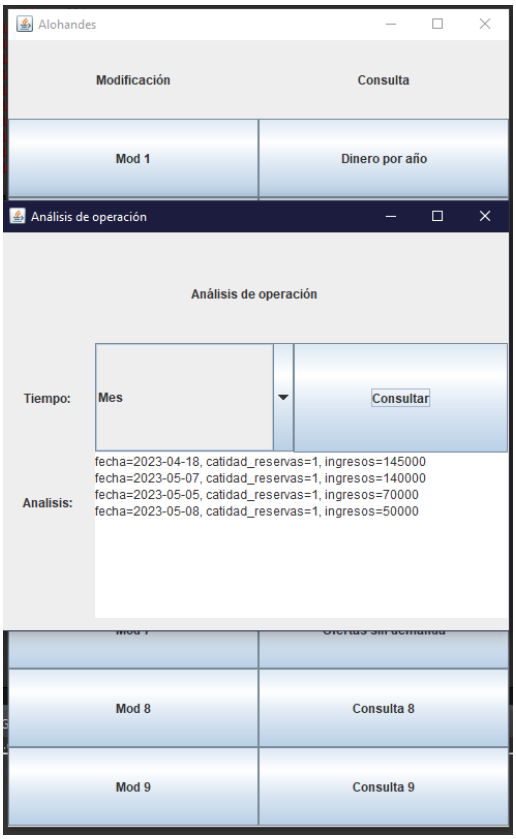
Luego de implementar estos método, lo siguiente fue crear un método en la clase PersistenciaAlohandes por cada requerimiento nuevo, este devuelve el resultado de las sentencias como una cadena de texto para su posterior vizualicación. Estos métodos son de la forma:

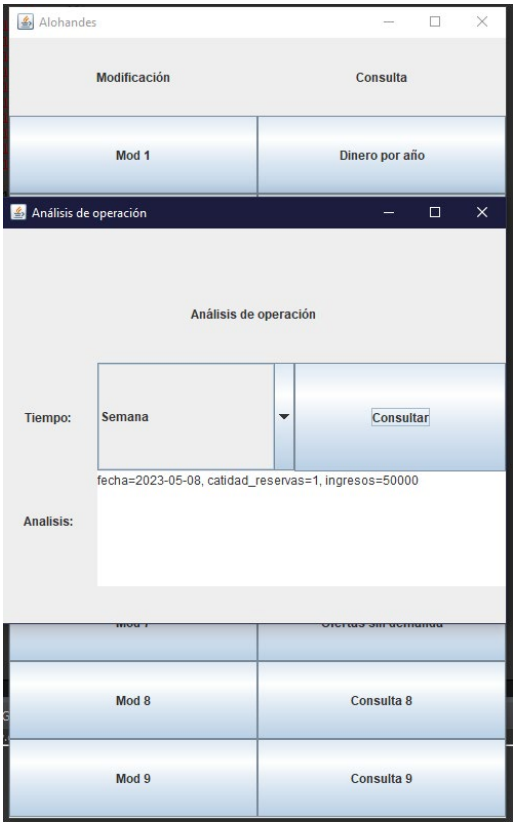
```
public String darClientesFrecuentes(String tipo, Long id) {
    PersistenceManager pm = pmf.getPersistenceManager();
    Transaction tx=pm.currentTransaction();
    try
    {
        tx.begin();
        List<Usuario> resp = sqlUtil.darClientesFrecuentes(pm, tipo, id);
        tx.commit();
        String str = "";
        for (Usuario element : resp) {
            str += element.toString()+ "\n";
        }
        return str;
    }
    catch (Exception e)
    {
        e.printStackTrace();
        log.error ("Exception : " + e.getMessage() + "\n" + darDetalleException(e));
        return "-1";
    }
    finally
    {
        if (tx.isActive())
        {
            tx.rollback();
        }
        pm.close();
    }
}
```

Por último, se incluyó cada uno de los nuevos requerimientos en un botón de la interfaz y se daba su resultado en una nueva ventana Jframe. Esto tiene la forma:

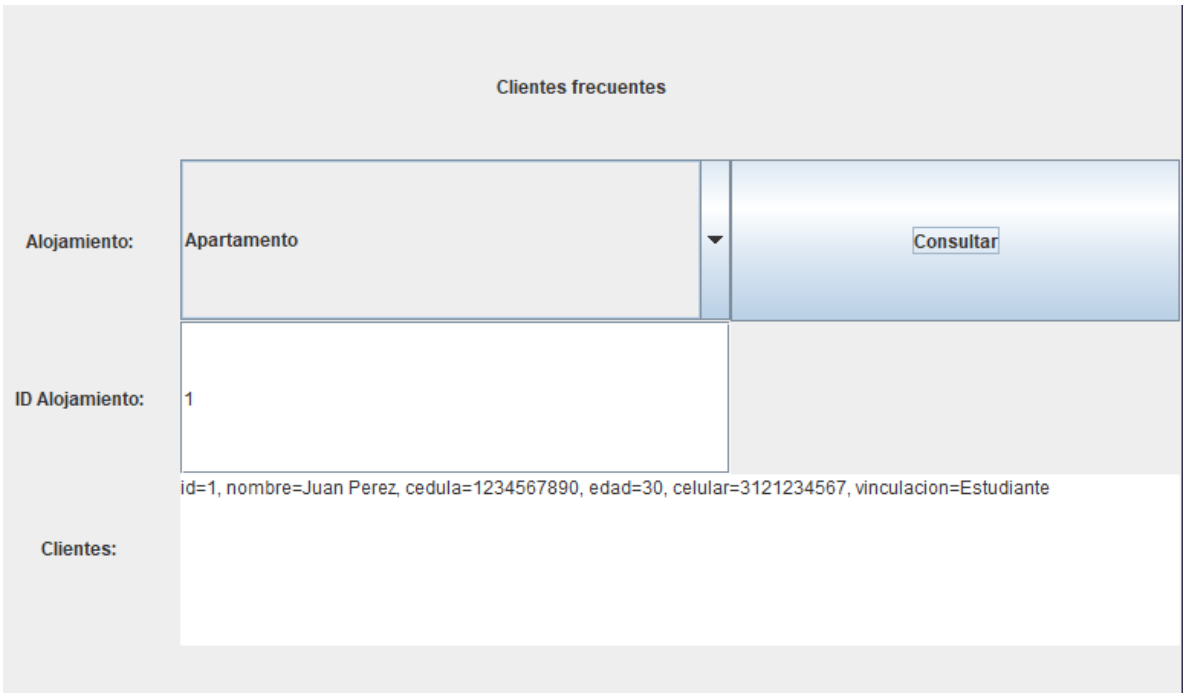
```
private void darOfertasSinDemanda() {
    JFrame ventana = new JFrame("Ofertas sin demanda");
    ventana.setSize(350, 600);
    ventana.setLocationRelativeTo(this);
    ventana.setVisible(true);
    ventana.setLayout(new GridBagLayout());
    JLabel titulo = new JLabel("Ofertas sin demanda", SwingConstants.CENTER);
    JTextArea resp = new JTextArea();
    resp.setEditable(false);

    GridBagConstraints constraints = new GridBagConstraints();
    constraints.weightx = 1.0;
    constraints.weighty = 0.0;
    constraints.fill = GridBagConstraints.HORIZONTAL;

    constraints.gridx = 0;
    constraints.gridy = 0;
    constraints.gridwidth = 1;
    constraints.gridheight = 1;
    constraints.ipady = 50;
    ventana.add(titulo, constraints);

    constraints.gridx = 0;
    constraints.gridy = 1;
    constraints.gridwidth = 2;
    constraints.gridheight = 30;
    resp.setText(pa.darOfertasSinDemandas());
    ventana.add(resp, constraints);
}
```

*Analizar la operación de Alohandes*



*Encontrar los clientes frecuentes*



*Encontrar las ofertas de alojamiento que no tienen mucha demanda*