SOEN 387: Web-Based Enterprise Application Design

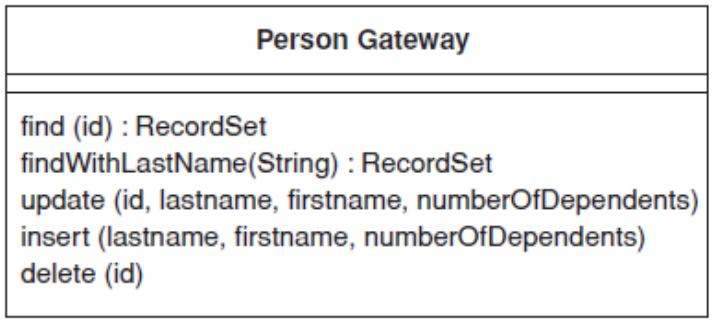
Chapter 10. DATA SOURCE ARCHITECTURAL PATTERNS

Chapter 9. DATA SOURCE ARCHITECTURAL PATTERNS

Table Data Gateway

Table Data Gateway is a*n object that acts as a Gateway to a database table. One instance handles all the rows in the table.*

Mixing SQL in application logic can cause several problems. Many developers aren't comfortable with SQL, and many who are comfortable may not write it well. Database administrators need to be able to find SQL easily so they can figure out how to tune and evolve the database.



A Table Data Gateway holds all the SQL for accessing a single table or view: selects, inserts, updates, and deletes. Other code calls its methods for all interaction with the database.

The **Table Data Gateway** is usually stateless, as its role is to push data back and forth.

How **Table Data Gateway** works

The trickiest thing about a *Table Data Gateway* is how it returns information from a query.

Even a simple find-by-ID query will return multiple data items.

In environments where you can return multiple items you can use that for a single row, but many languages give you only a single return value and many queries return multiple rows.

You can return data from query using:

1. Some simple data structure, such as a map. A map works, but it forces data to be copied out of the *Record Set* that comes from the database into the map.
2. Data Transfer Object . take all the data that the client needs and bundle it in Data Transfer Object
3. Return the *Record Set* that comes from the SQL query.

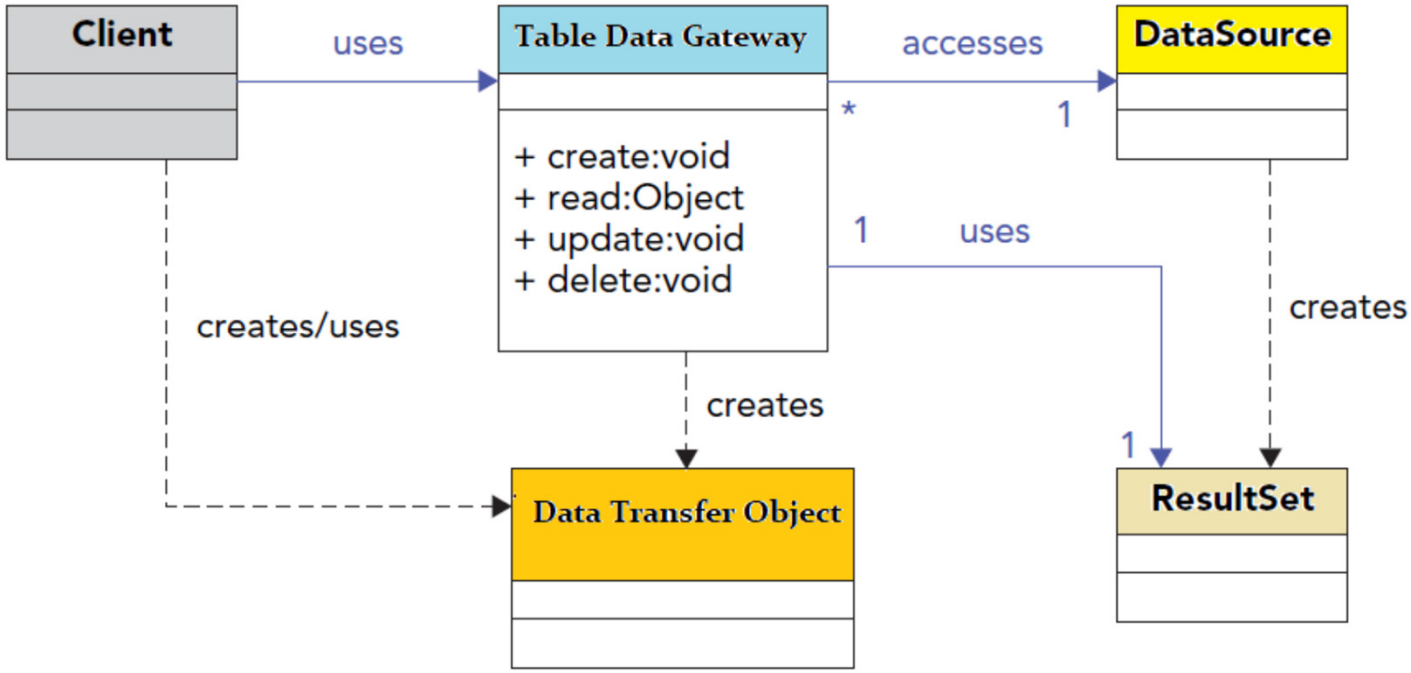
Most times when you use *Table Data Gateway*, you'll have one for each table in the database. For very simple cases, however, you can have a single *Table Data Gateway* that handles all methods for all tables.

Record Set* *is a*n in-memory representation of tabular data.**

When Table Data Gateway

Table Data Gateway works particularly well with *Table Module.* where it produces a record set data structure for the *Table Module* to work on.

Table Data Gateway is very suitable for *Transaction Scripts*.



A ResultSet object is a table of data representing a database result set, which is usually generated by executing a statement that queries the database.

Example: Person Gateway (C#)

Table Data Gateway is the usual form of database access in the windows world, so it makes sense to illustrate one with C#.

```
class PersonGateway...
public IDataReader FindAll() {
          String sql = "select * from person";
          return new OleDbCommand(sql, DB.Connection).ExecuteReader();
}
public IDataReader FindWithLastName(String lastName) {
          String sql = "SELECT * FROM person WHERE lastname = ?";
          IDbCommand comm = new OleDbCommand(sql, DB.Connection);
          comm.Parameters.Add(new OleDbParameter("lastname", lastName));
          return comm.ExecuteReader();
}
public IDataReader FindWhere(String whereClause) {
          String sql = String.Format("select * from person where {0}", whereClause);
          return new OleDbCommand(sql, DB.Connection).ExecuteReader();
}

public Object[] FindRow (long key) {
          String sql = "SELECT * FROM person WHERE id = ?";
          IDbCommand comm = new OleDbCommand(sql, DB.Connection);
          comm.Parameters.Add(new OleDbParameter("key",key));
          IDataReader reader = comm.ExecuteReader();
          reader.Read();
          Object [] result = new Object[reader.FieldCount];
          reader.GetValues(result);
          reader.Close();
          return result;
}
```

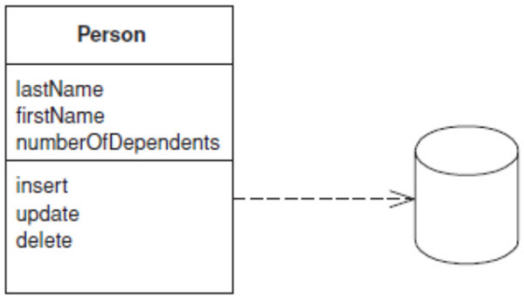
```
public void Update (long key, String lastname, String firstname, long numberOfDependents){
          String sql = @"
          UPDATE person
          SET lastname = ?, firstname = ?, numberOfDependents = ?
          WHERE id = ?";
          IDbCommand comm = new OleDbCommand(sql, DB.Connection);
          comm.Parameters.Add(new OleDbParameter ("last", lastname));
          comm.Parameters.Add(new OleDbParameter ("first", firstname));
          comm.Parameters.Add(new OleDbParameter ("numDep", numberOfDependents));
          comm.Parameters.Add(new OleDbParameter ("key", key));
          comm.ExecuteNonQuery();
}

public long Insert(String lastName, String firstName, long numberOfDependents) {
          String sql = "INSERT INTO person VALUES (?,?,?,?)";
          long key = GetNextID();
          IDbCommand comm = new OleDbCommand(sql, DB.Connection);
          comm.Parameters.Add(new OleDbParameter ("key", key));
          comm.Parameters.Add(new OleDbParameter ("last", lastName));
          comm.Parameters.Add(new OleDbParameter ("first", firstName));
          comm.Parameters.Add(new OleDbParameter ("numDep", numberOfDependents));
          comm.ExecuteNonQuery();
          return key;
}


public void Delete (long key) {
          String sql = "DELETE FROM person WHERE id = ?";
          IDbCommand comm = new OleDbCommand(sql, DB.Connection);
          comm.Parameters.Add(new OleDbParameter ("key", key));
          comm.ExecuteNonQuery();
}

}
```

Row Data Gateway

A ***Row Data Gateway*** acts as an object that exactly mimics a single record, such as one database row. In it each column in the database becomes one field.

A ***Row Data Gateway*** gives you objects that can be accessed with the regular mechanisms of your programming language.

Data Gateway directly. The gateway acts as a good interface for each row of data. This approach works particularly well for *Transaction Scripts*



Example: A Person Record (Java)

Here's an example for *Row Data Gateway*. It's a simple person table.

```
create table people (ID int primary key, lastname varchar, firstname varchar, number_of_dependents int)
```

PersonGateway is a gateway for the table. It starts with data fields and accessors.

```
class PersonGateway...
        private String lastName;
        private String firstName;
        private int numberOfDependents;
        public String getLastName() {
        return lastName;
}
public void setLastName(String lastName) {
        this.lastName = lastName;
}
public String getFirstName() {
        return firstName;
}
public void setFirstName(String firstName) {
        this.firstName = firstName;
}
public int getNumberOfDependents() {
        return numberOfDependents;
}
public void setNumberOfDependents(int numberOfDependents) {
        this.numberOfDependents = numberOfDependents;
}
```

```
private static final String updateStatementString =
  "UPDATE people " + " set lastname = ?, firstname = ?, number_of_dependents = ? " + " where id = ?";

public void update() {
PreparedStatement updateStatement = null;
try {
 updateStatement = DB.prepare(updateStatementString);
 updateStatement.setString(1, lastName);
 updateStatement.setString(2, firstName);
 updateStatement.setInt(3, numberOfDependents);
 updateStatement.setInt(4, getID().intValue());
 updateStatement.execute();
} catch (Exception e) {
throw new ApplicationException(e);
} finally {DB.cleanUp(updateStatement);
}
}
private static final String insertStatementString =
 "INSERT INTO people VALUES (?, ?, ?, ?)";
public Long insert() {
  PreparedStatement insertStatement = null;
try {
  insertStatement = DB.prepare(insertStatementString);
  setID(findNextDatabaseId());
  insertStatement.setInt(1, getID().intValue());
  insertStatement.setString(2, lastName);
  insertStatement.setString(3, firstName);
  insertStatement.setInt(4, numberOfDependents);
  insertStatement.execute();
  Registry.addPerson(this);
 return getID();
} catch (SQLException e) {
```

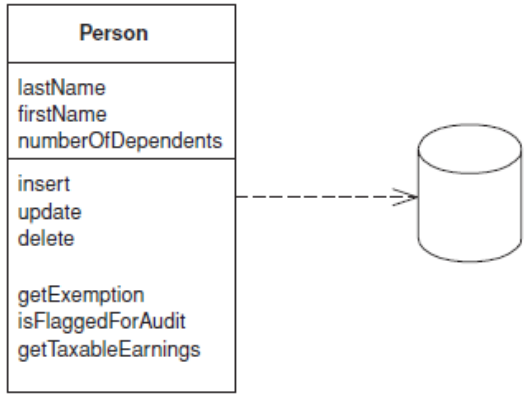
To find more than one person according to some criteria we can provide a suitable finder method.

```
class PersonFinder...
private static final String findResponsibleStatement =
 "SELECT id, lastname, firstname, number_of_dependents " + " from people " + " WHERE number_of_dependents > 0";
public List findResponsibles() {
 List result = new ArrayList();
 PreparedStatement stmt = null;
 ResultSet rs = null;
try {
 stmt = DB.prepare(findResponsibleStatement);
 rs = stmt.executeQuery();
while (rs.next()) {
 result.add(PersonGateway.load(rs));
}
return result;
} catch (SQLException e) {
 throw new ApplicationException(e);
} finally {DB.cleanUp(stmt, rs);
}
}
```

Active Record

An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data.

The data structure of the *Active Record* should exactly match that of the database: one field in the class for each column in the table.

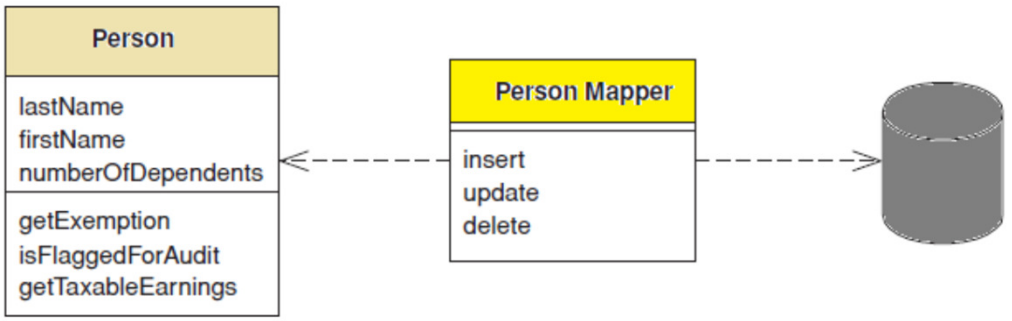


Active Record is very similar to *Row Data Gateway*. The principal difference is that a *Row Data Gateway* contains only database access while an *Active Record* contains both data source and domain logic.

Data Mapper

A layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself.

To load a person from the database, a client would call a find method on the mapper . The mapper uses an Identity Map to see if the person is already loaded; if not, it loads it.



A simple *Data Mapper* would just map a database table to an equivalent in memory class

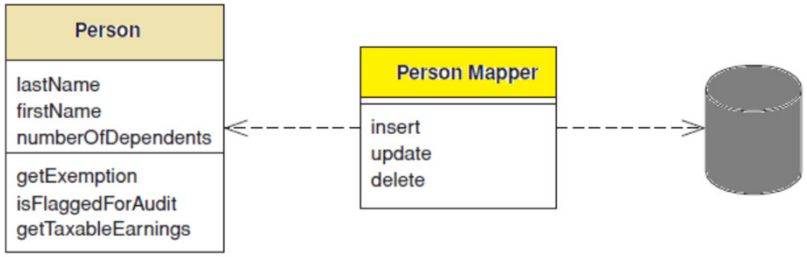
When it comes to inserts and updates, the database mapping layer needs to understand what objects have changed, which new ones have been created, and which ones have been destroyed. It also has to fit the whole workload into a transactional framework. The *Unit of Work* pattern is a good way to organize this.

Unit of Work pattern maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems.

When to use Data Mapper

The primary occasion for using *Data Mapper* is when you want the database schema and the object model to evolve independently. The most common case for this is with a *Domain Model* .

Data Mapper's primary benefit is that when working on the domain model you can ignore the database, both in design and in the build and testing process. The domain objects have no idea what the database structure is, because all the correspondence is done by the mappers.



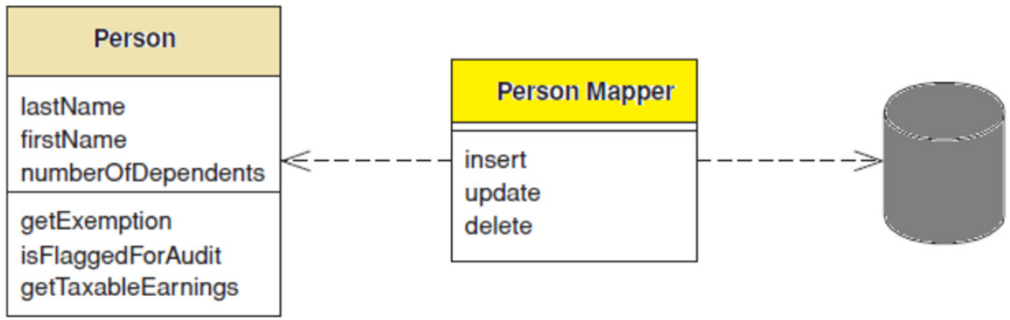
Example: A Simple Database Mapper (Java)

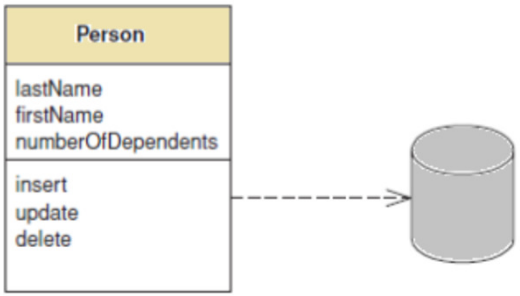
The database schema looks like this:

create table people (ID int primary key, lastname varchar, firstname varchar, number_of_dependents int)

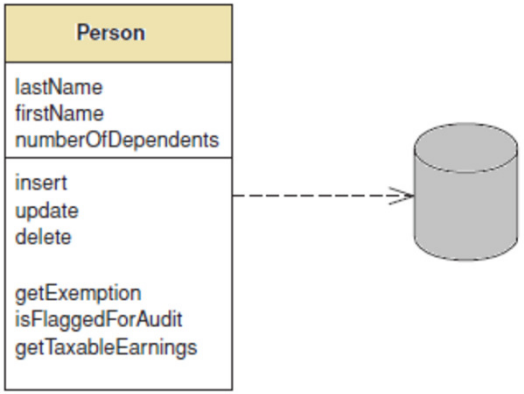
```
class PersonMapper...
protected String findStatement() {
return "SELECT " + COLUMNS + " FROM people" + " WHERE id = ?";
}
public static final String COLUMNS = " id, lastname, firstname, number_of_dependents
";
public Person find(Long id) {
        return (Person) abstractFind(id);
}
public Person find(long id) {
        return find(new Long(id));
}
```

```
class PersonMapper...
private static final String updateStatementString =
"UPDATE people " +
" SET lastname = ?, firstname = ?, number_of_dependents = ? " + " WHERE id = ?";

public void update(Person subject) {
PreparedStatement updateStatement = null;
try {
updateStatement = DB.prepare(updateStatementString);
updateStatement.setString(1, subject.getLastName());
updateStatement.setString(2, subject.getFirstName());
updateStatement.setInt(3, subject.getNumberOfDependents());
updateStatement.setInt(4, subject.getID().intValue());
updateStatement.execute();
} catch (Exception e) {
throw new ApplicationException(e);
} finally {
DB.cleanUp(updateStatement);
}
}
```

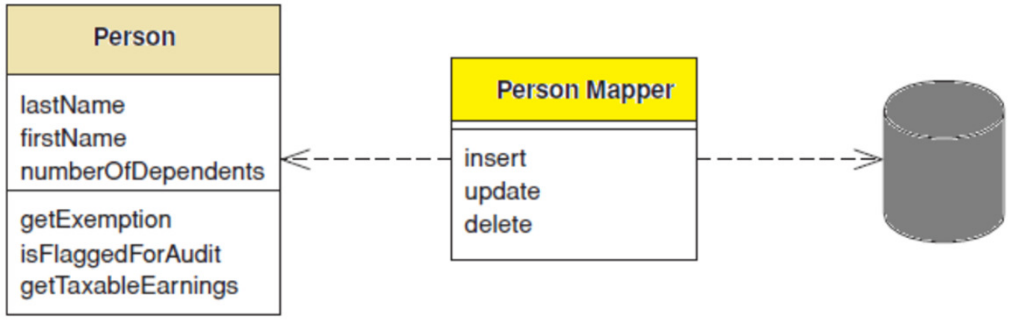




Row Data Gateway



Active Record



Data Mapper