18 Structural: Proxy Pattern for Secure & Efficient Database Access

**Scenario** Your SaaS platform relies on a heavy RealDatabase object to run queries.  
 Requirements:

* **Lazy loading** – open the real connection only when the very first query or connect is requested.
* **Access control** – deny connections to callers without the proper role.
* **Logging** – record every operation.
* **Optional caching** – add-on proxy layer that remembers SELECT results.

You will design a layered proxy chain that meets these needs while leaving RealDatabase untouched.

#### **Tasks**

1 — analysis/proxy\_drivers.md  
 • Explain why creating RealDatabase eagerly wastes resources.  
 • List two security risks solved by an access-control proxy.

2 — Core contracts (src/main/java/proxy/)  
 • Database – the **Subject** interface with connect() & executeQuery(String).  
 • RealDatabase – prints “Connecting…” and “Executing query…”. **Do not** add logging, caching, or checks here.

3 — Proxies

| **File** | **Responsibility** |
| --- | --- |
| LazyDatabaseProxy.java | wraps a Supplier<RealDatabase>; instantiates on first call |
| AccessControlProxy.java | checks a boolean hasAccess flag; blocks if false |
| LoggingProxy.java | prints messages before delegating |
| CachingProxy.java | caches SELECT results in a Map<String,String> (fake result text) |

Each implements Database and holds a reference to the **next** link in the chain.

4 — Proxy wiring ProxyDemo.java

* Build chain **Logging → Access Control → Lazy** (no caching) and run one permitted query.
* Build chain **Logging → Access Control (denied)** and show failure.
* Build chain **Logging → Access Control → Lazy → Caching** and execute the same SELECT twice, proving the second call hits the cache.

5 — Unit tests (src/test/java/)

* LazyInstantiationTest – assert RealDatabase is allocated only after first method call (use a static counter).
* AccessDeniedTest – verify no “Connecting…” output when access = false.
* CachingHitTest – ensure second identical query prints “Cache hit”.

6 — Reflection (reflection.md)

* Summarise how each proxy addresses one cross-cutting concern.
* Discuss performance overhead vs. benefits.
* Outline steps to add a **RateLimitingProxy** tomorrow without editing existing classes.

#### **Deliverables**

analysis/proxy\_drivers.md

src/main/java/proxy/Database.java

src/main/java/proxy/RealDatabase.java

src/main/java/proxy/LazyDatabaseProxy.java

src/main/java/proxy/AccessControlProxy.java

src/main/java/proxy/LoggingProxy.java

src/main/java/proxy/CachingProxy.java

src/main/java/proxy/ProxyDemo.java

src/test/java/LazyInstantiationTest.java

src/test/java/AccessDeniedTest.java

src/test/java/CachingHitTest.java

reflection.md

README.md

## **Detailed Solution**

### **1 Subject & RealSubject**

/\* Database.java \*/

package proxy;

public interface Database {

void connect();

void executeQuery(String query);

}

/\* RealDatabase.java \*/

package proxy;

public class RealDatabase implements Database {

public static int instances = 0; // for tests

public RealDatabase(){ instances++; }

@Override public void connect(){

System.out.println("Connecting to the database...");

}

@Override public void executeQuery(String q){

System.out.println("Executing query: "+q);

}

}

### **2 Proxy chain links**

/\* LazyDatabaseProxy.java \*/

package proxy;

import java.util.function.Supplier;

public class LazyDatabaseProxy implements Database {

private final Supplier<RealDatabase> supplier;

private RealDatabase real;

public LazyDatabaseProxy(Supplier<RealDatabase> s){ this.supplier = s; }

private RealDatabase db(){

if(real==null) real = supplier.get();

return real;

}

public void connect(){ db().connect(); }

public void executeQuery(String q){ db().executeQuery(q); }

}

/\* AccessControlProxy.java \*/

package proxy;

public class AccessControlProxy implements Database {

private final Database next;

private final boolean allowed;

public AccessControlProxy(Database next, boolean allowed){

this.next=next; this.allowed=allowed;

}

public void connect(){

if(!allowed){ deny("connect"); return; }

next.connect();

}

public void executeQuery(String q){

if(!allowed){ deny("execute queries"); return; }

next.executeQuery(q);

}

private void deny(String op){

System.out.println("Access denied: Not permitted to "+op+".");

}

}

/\* LoggingProxy.java \*/

package proxy;

public class LoggingProxy implements Database {

private final Database next;

public LoggingProxy(Database next){ this.next=next; }

public void connect(){

System.out.println("Log: connect requested.");

next.connect();

}

public void executeQuery(String q){

System.out.println("Log: query -> "+q);

next.executeQuery(q);

}

}

/\* CachingProxy.java \*/

package proxy;

import java.util.\*;

public class CachingProxy implements Database {

private final Database next;

private final Map<String,String> cache = new HashMap<>();

public CachingProxy(Database next){ this.next=next; }

public void connect(){ next.connect(); }

public void executeQuery(String q){

if(q.trim().toUpperCase().startsWith("SELECT") && cache.containsKey(q)){

System.out.println("Cache hit: "+cache.get(q));

return;

}

next.executeQuery(q);

if(q.trim().toUpperCase().startsWith("SELECT")){

cache.put(q,"Result of "+q);

}

}

}

### **3 Demo**

/\* ProxyDemo.java \*/

package proxy;

public class ProxyDemo {

public static void main(String[] args){

// chain: Logging -> Access (allow) -> Lazy

Database db = new LoggingProxy(

new AccessControlProxy(

new LazyDatabaseProxy(RealDatabase::new), true));

db.connect();

db.executeQuery("SELECT \* FROM users");

System.out.println("\n--- Access denied demo ---");

Database dbDenied = new LoggingProxy(

new AccessControlProxy(

new LazyDatabaseProxy(RealDatabase::new), false));

dbDenied.executeQuery("UPDATE users SET role='ADMIN'");

System.out.println("\n--- Caching demo ---");

Database cached = new LoggingProxy(

new AccessControlProxy(

new CachingProxy(

new LazyDatabaseProxy(RealDatabase::new)), true));

cached.executeQuery("SELECT \* FROM products");

cached.executeQuery("SELECT \* FROM products"); // second time hits cache

}

}

**Sample console**

Log: connect requested.

Connecting to the database...

Log: query -> SELECT \* FROM users

Executing query: SELECT \* FROM users

--- Access denied demo ---

Log: query -> UPDATE users SET role='ADMIN'

Access denied: Not permitted to execute queries.

--- Caching demo ---

Log: query -> SELECT \* FROM products

Connecting to the database...

Executing query: SELECT \* FROM products

Log: query -> SELECT \* FROM products

Cache hit: Result of SELECT \* FROM products

### **4 Tests (outline)**

/\* LazyInstantiationTest \*/

RealDatabase.instances = 0;

Database lazy = new LazyDatabaseProxy(RealDatabase::new);

assertEquals(0, RealDatabase.instances);

lazy.connect();

assertEquals(1, RealDatabase.instances);

/\* AccessDeniedTest \*/

String out = capture(() ->

new AccessControlProxy(new LazyDatabaseProxy(RealDatabase::new), false)

.executeQuery("SELECT 1"));

assertTrue(out.contains("Access denied"));

assertFalse(out.contains("Connecting"));

/\* CachingHitTest \*/

String s = capture(() -> {

Database c = new CachingProxy(new LazyDatabaseProxy(RealDatabase::new));

c.connect(); c.executeQuery("SELECT 1"); c.executeQuery("SELECT 1");

});

assertTrue(s.contains("Cache hit"));

*(Helper capture(Runnable) redirects System.out to a string.)*

### **Reflection (key points)**

| **Concern** | **Proxy that handles it** | **Outcome** |
| --- | --- | --- |
| Lazy loading | LazyDatabaseProxy | Avoids connection cost until first use |
| Access control | AccessControlProxy | Centralised permission check |
| Logging | LoggingProxy | Cross-cutting monitoring without touching RealDatabase |
| Caching | CachingProxy | Faster repeat SELECTs |

*Adding RateLimitingProxy*: create a new class implementing Database, track timestamps, then insert it anywhere in the chain—no changes to existing proxies or RealDatabase.

This layered Proxy solution enforces security, boosts performance, and collects telemetry, all while keeping the real database code pristine.