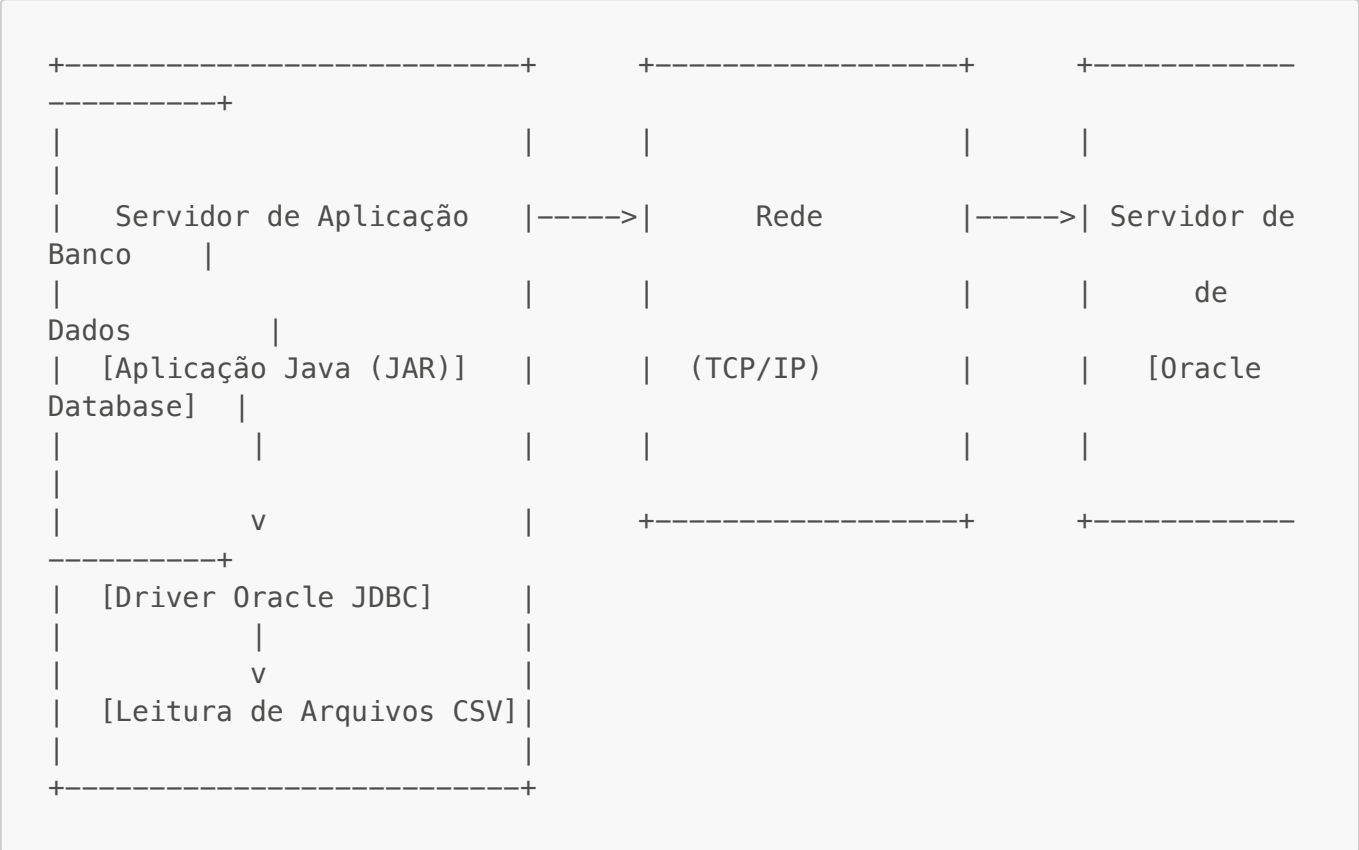
Proposta de Solução: Processo de Reconciliação com Java e JDBC

Esta proposta descreve uma aplicação Java independente, criada para substituir a rotina atual que usa stored procedures e scripts shell. O objetivo é garantir que tudo esteja dentro das políticas de segurança do cliente.

---

1. Visão Geral da Arquitetura

A solução é uma aplicação Java que roda em um servidor de aplicação ou em uma máquina dedicada — não precisa ser instalada no servidor do banco de dados. A comunicação acontece pela rede de forma segura.



---

2. Componentes Necessários

- **Java Development Kit (JDK):** Versão 8 ou superior para compilar e rodar o programa.
- **Oracle JDBC Driver:** O arquivo `.jar` oficial da Oracle (exemplo: `ojdbc8.jar`) para conectar a aplicação ao banco Oracle. Ele será incluído no pacote da aplicação.
- **Biblioteca para leitura de CSV:** Para facilitar a leitura e evitar erros, vamos usar uma biblioteca pronta como OpenCSV ou Apache Commons CSV.

---

3. Detalhamento do Processo

Passo 1: Conexão com o Banco de Dados

A aplicação começa conectando ao banco. Usamos a classe `DriverManager` do JDBC para abrir essa conexão.

- **URL de conexão:** Contém endereço, porta e serviço (SID) do Oracle, por exemplo:

```
jdbc:oracle:thin:@//endereco-do-servidor:1521/NOME_DO_SERVICO
```

- **Credenciais:** Usuário e senha ficam guardados de forma segura, em arquivo `.properties` ou variáveis de ambiente, nunca no código.

Exemplo conceitual de código:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;

String url = "jdbc:oracle:thin:@//host:port/service";
Properties props = new Properties();
props.setProperty("user", "meu_usuario");
props.setProperty("password", "minha_senha_segura");

Connection conn = DriverManager.getConnection(url, props);
System.out.println("Conexão com o Oracle estabelecida com sucesso!");
```

Passo 2: Leitura e Processamento dos Arquivos CSV

- A aplicação procura arquivos `.csv` em uma pasta pré-definida.
- Para cada arquivo encontrado:
  - Usa a biblioteca CSV para ler linha a linha.
  - Cada linha vira um objeto Java (POJO), como uma `NotaFiscal`, para facilitar o trabalho com os dados.

Passo 3: Lógica de Reconciliação e Execução de SQL

Esse é o coração da aplicação. Para cada `NotaFiscal`:

- **Verifica se já existe no banco** com um `SELECT` usando a chave primária (ex: número da nota).
- **Se não existe:** insere a nota com `INSERT`.
- **Se existe:** compara os dados do CSV com o banco.
  - Se diferente, atualiza com `UPDATE`.
  - Se igual, não faz nada (evita duplicidade e processamento desnecessário).

Usamos `PreparedStatement` para executar os comandos SQL porque:

- **Segurança:** Evita ataques de SQL Injection.
- **Performance:** O banco pré-compila as instruções, acelerando execuções repetidas.

Exemplo conceitual para `INSERT`:

```
String sql = "INSERT INTO NOTAS_FISCAIS (ID, VALOR, DATA_EMISSAO) VALUES
(?, ?, ?)";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
    pstmt.setInt(1, notaFiscal.getId());
    pstmt.setDouble(2, notaFiscal.getValor());
    pstmt.setDate(3, new
java.sql.Date(notaFiscal.getDataEmissao().getTime()));

    pstmt.executeUpdate(); // Executa a inserção
}
```

Passo 4: Gerenciamento de Transações

Para garantir que os dados fiquem sempre consistentes:

- Desligamos o auto-commit: `conn.setAutoCommit(false);`
- Após processar todas as linhas do arquivo com sucesso, damos um `conn.commit();`
- Se algum erro acontecer (arquivo mal formatado, falha na conexão), damos um `conn.rollback();` para desfazer as mudanças feitas pelo arquivo.
- No final, fechamos todos os recursos (`Connection`, `Statement`) em um bloco `finally` para evitar vazamento de memória.

---

4. Vantagens da Solução Proposta

- **Segurança:** O código roda fora do banco, respeitando as regras do cliente.
- **Portabilidade:** É Java puro, roda em qualquer sistema com JVM (Windows, Linux, etc).
- **Facilidade de manutenção:** Código Java é mais simples de manter e evoluir que stored procedures e scripts shell misturados.
- **Flexibilidade:** Podemos ampliar a solução para enviar e-mails, gerar logs, integrar com outras APIs, etc.