UNIVERSIDADE DA BEIRA INTERIOR
Faculdade de Engenharia

Sincronização e Reconciliação de Dados em Base de Dados

(Versão Definitiva Após Defesa Pública)

Orlando José Jamba Cawende

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor João Manuel da Silva Fernandes Muranho

Covilhã, Julho de 2018

Dedicatória

À minha esposa, Dulce Madalena e às minhas filhas Josimara, Analdina e Abiúde, por toda ajuda, apoio e paciência. Continuem sendo para sempre pessoas que trazem muita felicidade.

Aos meus pais, irmão e demais familiares e amigos, pelo constante incentivo.

Agradecimentos

Agradeço em primeiro lugar a Deus, dador da vida que nos tem dado ânimo para prosseguir com os estudos e outros afazeres. Agradeço-O, pela proteção ao longo dessa jornada, pois, serviu de pedra angular do conhecimento adquirido e lição de vida.

Agradeço a todos os professores do Departamento de Informática da UBI com os quais tive a oportunidade de trabalhar e aprender, especialmente ao meu orientador Professor Doutor João Muranho, pela inspiração, conselhos, encorajamento e apoio.

Agradeço a todos que, voluntariamente, deram o seu precioso tempo para ouvir as minhas ideias e fazer comentários sobre o trabalho que aqui apresento.

Agradeço ao Hermenegildo Simão, Ngaiele Fundão, Mário Pereira, Domingos Dionísio, Jacinto Comuleã, Tomais Hambili, Domingos de Oliveira, Alberto Segunda, pelo convívio, bem como pelo acompanhamento. Agradeço, igualmente, aos Diretores do Instituto Médio de Economia do Lubango, e Superior Politécnico da Huila. De igual modo, ao Secretário Provincial da Igreja Evangélica Congregacional em Angola pela oportunidade e confiança depositada.

O meu mais profundo agradecimento vai para toda a minha família.

A todos vós os meus eternos agradecimentos.

Resumo

O presente trabalho aborda as técnicas de sincronização e reconciliação de dados usadas na construção de aplicações em ambiente desconectado.

Apresenta-se uma breve resenha das principais técnicas de sincronização e reconciliação de dados existentes, bem como as ferramentas existentes nesta área.

Neste trabalho são propostas duas abordagens para o problema em estudo, uma baseada em transferências linha-a-linha, orientadas por um controlador de software, designado de *Sync Server*, e outra baseada num mecanismo das bases de dados distribuídas, o *Linked Server*. Foram desenvolvidas duas aplicações para lidar com o processo de sincronização e reconciliação de dados em ambiente desconectado, empregando as abordagens propostas, onde grandes quantidades de dados são transferidas de bases de dados locais para um servidor remoto e aí são reconciliados.

No contexto das aplicações em ambiente desconectado, o programador pode optar pela sincronização baseada em *Linked Server* para os dispositivos móveis mais apetrechados de recursos computacionais e por transferência linha-a-linha para os dispositivos mais modestos.

Palavras-chave

Aplicações em ambientes desconectado, replicação, sincronização de dados, reconciliação de dados.

Abstract

This work addresses the data synchronization and reconciliation techniques used in building applications in disconnected environment.

A brief review of the main data synchronization and reconciliation techniques, as well as existing tools, in this area is presented.

In this work, two approaches were proposed to the problem under study, one based on row-by-row transfers, controlled by a software piece, called *Sync Server*, and the other based on the *Linked Server* feature (a distributed database mechanism). Two applications were also developed to deal with the process of data synchronization and reconciliation in a disconnection environment, using the proposed approaches, where large amounts of data are transferred from local databases to a remote server and reconciled there.

In the context of applications in disconnected environment, the programmer can choose *Linked Server*-based synchronization for powerful mobile devices, and row-by-row transfer for the modest devices.

Keywords

Applications in offline environments, replication, data synchronization, data reconciliation
.

Conteúdo

1 Introdução **1**

2 Estado do conhecimento **5**

2.1 Introdução . . . 5
2.2 Fundamentos da sincronização e reconciliação de dados . . . 5
2.3 Sincronização e reconciliação de dados . . . 5
2.4 Abordagens sobre a sincronização e reconciliação . . . 6
2.5 Replicação de base de dados . . . 8
2.5.1 Replicação em aplicações móveis . . . 10
2.5.2 Esquemas de replicação e categorização . . . 10
2.5.3 Semântica de replicação baseada em *logs* . . . 12
2.5.4 Protocolos de implementação das réplicas . . . 12
2.6 Modelo Relacional . . . 13
2.7 Reconciliação . . . 14
2.8 Transações em base de dados . . . 14
2.8.1 Propriedades das transações . . . 15
2.8.2 Gestão de transações . . . 15
2.9 Controlo de concorrência . . . 16
2.9.1 Atualização perdidas . . . 16
2.9.2 Dados não consolidados . . . 17
2.9.3 Leituras inconsistentes . . . 17
2.10 Escalonador . . . 17
2.11 Controlo de concorrência com métodos de bloqueios . . . 18
2.11.1 Tipos de trincos . . . 18
2.11.2 Bloqueio de duas fases . . . 19
2.11.3 Métodos de replicação otimista . . . 19
2.11.4 Métodos de *timestamp* . . . 19
2.12 Níveis de Isolamento . . . 20
2.12.1 Read uncommitted . . . 21
2.12.2 Read committed . . . 21
2.12.3 Repeatable read . . . 21
2.12.4 Snapshot . . . 21
2.12.5 Serializable . . . 22
2.12.6 Tipos de problemas de concorrência . . . 22
2.13 Arquitetura . . . 22
2.13.1 Cliente/Servidor . . . 23
2.13.2 *Peer-to-Peer* . . . 23
2.13.3 Agentes de replicação . . . 24
2.14 Sincronização em base de dados não estruturadas . . . 24

3 Implementação **27**
3.1 Introdução . . . 27
3.2 Especificação de requisitos da aplicação . . . 27
3.2.1 Requisitos funcionais . . . 27
3.2.2 Requisitos Não Funcionais . . . 28
3.3 Análise de requisitos . . . 28
3.3.1 Diagrama de Casos de Uso . . . 28
3.3.2 Diagrama de sequência . . . 32
3.3.3 Diagrama de classe . . . 33
3.3.4 Controlo e deteção de alterações . . . 34
3.3.5 Regras de negócio . . . 34
3.3.6 Visão geral da solução . . . 35
3.3.7 Estabelecimento da comunicação entre o servidor e aplicação cliente 35
3.4 Modelo de dados . . . 36
3.5 Tecnologias e Ferramentas Utilizadas . . . 37
3.5.1 NetBeans 8.2 . . . 37
3.5.2 SQL Server Management Studio 2012 . . . 38
3.6 Garantia da persistência de recursos. . . . 38
3.6.1 A operação de inserção . . . 38
3.6.2 A operação de exclusão . . . 38
3.6.3 Operação de atualização . . . 38
3.7 Transferência de dados . . . 39
3.7.1 Transferência linha-a-linha . . . 39
3.7.2 Transferência via *Linked Server* . . . 40
3.8 Sincronização . . . 41
3.8.1 *Linked servers* e bases de dados distribuídas . . . 41
3.8.2 Gestão de transparência . . . 41
3.8.3 Replicação com SQL Server . . . 42
3.8.4 Arquitetura . . . 42

4 Resultados **45**

5 Considerações finais **49**
5.1 Conclusões . . . 49
5.2 Trabalhos futuros . . . 50

Bibliografia **51**

A Anexos **57**
A.1 Método de obtenção de registo do cliente . . . 57
A.2 Método para enviar recuros ao servidor remoto . . . 57
A.3 Método para apagar registo na máquina local . . . 57
A.4 Método de receção de dados servidor remoto . . . 58
A.5 Painel principal da aplicação . . . 58

B Apêndice **59**
B.1 Dicionário de dados 59

Lista de Figuras

2.1 Exemplo de *deadlock* . . . 9
2.2 Extração e troca de Ontologia [1] . . . 12
2.3 Arquitetura Cliente/Servidor . . . 23
2.4 Arquitetura *Peer-to-Peer* . . . 23
2.5 Arquitetura agente de replicação . . . 24

3.1 Diagrama de caso de uso . . . 28
3.2 Diagrama de sequência para registo de cliente . . . 32
3.3 Diagrama de sequência para registar encomenda . . . 32
3.4 Diagrama de sequência para satisfazer encomenda . . . 33
3.5 Diagrama de sequência para registar stock . . . 33
3.6 Diagrama de sequência para registar produto . . . 33
3.7 Diagrama de classe . . . 34
3.8 Modelo abstrato do processo de sincronização . . . 35
3.9 Diagrama Entidade-Associação do exemplo . . . 36
3.10 Transferência de dados, linha-a-linha, entre o cliente e o *Sync Server* . . . 40
3.11 Transferência de dados por SGBD . . . 40

4.1 Transferência linha-a-linha . . . 46
4.2 Transferência com base no *Linked Server* . . . 46

A.1 Método de obtenção registos . . . 57
A.2 Método para enviar recursos ao servidor remoto . . . 57
A.3 Método para inserir dados no servidor . . . 57
A.4 Método para inserir dados no servidor . . . 58
A.5 Painel principal da aplicação . . . 58

B.1 Tabela representando o dicionário de dadosa . . . 59

Lista de Tabelas

2.1 Execução em série de transações 16
2.2 Atualizações perdidas 17
2.3 Dados não consolidados 17
2.4 Anomalias associadas aos níveis de isolamento 22

3.1 Requisitos funcionais 27
3.2 Requisitos Não funcionais 28
3.3 Registo de encomenda. 29
3.4 Registo de Cliente. 29
3.5 Editar Cliente. 29
3.6 Atualizar Encomenda. 30
3.7 Consultar cliente. 30
3.8 Consultar encomenda. 30
3.9 Registar stock. 30
3.10 Atualizar stock. 30
3.11 Registar produto. 31
3.12 Atualizar Produtos. 31
3.13 Inserir localização. 31
3.14 Satisfazer Encomenda. 31

Lista de Acrónimos

PC	Personal Computer
UML	Unified Modeling Language
SSMS	SQL Server Management Studio
PDA	Personal Digital Assistants
GRS	Group Replication Server
SQL	Strutured Querey Language
ESMS	Electronic student management system
SAMD	Synchronization Algorithms based on Message Digest
ISO	International Organization for Standardization
DBMS	Data Base Management System
SGBD	Sistema de Gestão de Base de Dados
ACID	Atomicity, Consistency, Isolation, Durability
ANSI	American National Standards Institute
NOSQL	Not Only SQL
SSD	Solid State Driver
AWS	Amazon Web Services
MTV	Music Television

Capítulo 1

Introdução

O desenvolvimento tecnológico tem impulsionado a utilização crescente de meios de comunicação, o que, aliado à miniaturização dos recursos computacionais, tem permitido aos utilizadores comunicar e aceder à informação de forma mais rápida e trabalhar com maior comodidade. Este desenvolvimento proporcionou o acesso aos dados em qualquer lugar, quer através de computador quer através dispositivos móveis [2].

No que concerne ao progresso na área de informática, o desenvolvimento tecnológico impulsionou o aparecimento de um novo ambiente de computação com recurso a dispositivos móveis, tais como, *smartphones, tablets, PDAs (Personal Digital Assistants)*[1]Computador de dimensão reduzida, dotada de grande capacidade computacional. e computadores portáteis. Estes dispositivos evoluíram rapidamente a partir de equipamentos simples, capazes de envio e receção de chamadas e mensagens, para dispositivos mais complexos, que podem ser usados como meios de gestão e armazenamento de dados pessoais e apoio empresarial.

Para que as aplicações destes dispositivos possam interagir entre si, é necessário um conjunto de padrões que possibilitem a integração de tecnologias bem como a gestão, processamento e manipulação de recursos. Neste âmbito surgiram modelos de desenvolvimento que dependem unicamente de tecnologias móveis, que implicam um novo paradigma.

A rede móvel não possui largura de banda, estabilidade, conexão persistente, segurança e fiabilidade suficientes que possam viabilizar o funcionamento regular do conjunto de instruções operadas [3]. O que torna a sincronização mais complexa, em relação ao modelo tradicional, pois os utilizadores podem realizar operações, tais como modificar, excluir e adicionar recursos, durante o tempo em que estiverem desconectados [4].

A temática da replicação total, ou parcial, da base de dados é uma das mais importantes em aplicações para ambientes móveis pois os dispositivos nem sempre possuem recursos estáveis, enfrentando, por exemplo, dificuldades na manutenção da comunicação.

As orientações para o desenvolvimento de aplicações envolvendo a utilização de mecanismos de replicação abrangem a periodicidade com que as réplicas são reconciliadas, e como e com que frequência, uma determinada réplica é disseminada, para que possa ocorrer a sincronização e a reconciliação de recursos entre a origem e o destino [5][6]. Por este motivo coloca-se a questão: como garantir a consistência dos dados quando um sistema possui várias réplicas de dados?

As réplicas de dados possibilitam a computação desconectada, oferecendo disponibilidade

[1]

e mobilidade. A utilização de réplicas flexibilizou a necessidade da comunicação contínua entre unidades móveis e fixas, permitindo desta forma, que clientes, empresas e organizações utilizem equipamentos de baixo custo.

Num ambiente cliente/servidor os dados são armazenados no servidor. Em ambiente desconectado os clientes replicam um subconjunto da base de dados. A administração e gestão das réplicas neste ambiente implicam muitos desafios. Diversas metodologias aplicadas em bases de dados distribuídas foram adaptadas para bases de dados móveis, permitindo, deste modo, que enquanto o cliente estiver conectado, cada transação local confirmada seja reconciliada com o servidor central, evitando a inconsistência de dados de todas as réplicas. Na eventualidade do cliente não conseguir conectar-se com o servidor central, as transações são armazenadas, no sentido de que as suas atualizações fiquem disponíveis para outras operações, assumindo o compromisso de as propagar para o servidor aquando da reintegração, evitando assim comprometer o estado da base de dados.

Para que a replicação possa ser efetivada é necessário sincronizar os dispositivos. No âmbito da computação, a sincronização é uma atividade de transferência de cópias de estrutura e dados entre duas entidades, sendo seus recursos rastreados através de identificação de versões, quer por aplicação de uma função de *hash*, quer por utilização de selos temporais (*timestamps*). Esta verificação é feita comparando os recursos existentes na réplica com os existentes no servidor [5][7].

A decisão da escolha de um dispositivo, *smartphone, tablet* ou PC portátil, assenta sempre na avaliação das especificações que o dispositivo oferece. Os dispositivos móveis nos últimos anos tornaram-se mais potentes, com armazenamento local e capacidade de efetuar cálculos complexos, podendo, assim, assumir novos papéis. Por exemplo, um conjunto de recursos, pode ser transferido da base de dados do servidor remoto para a (base de dados) réplica do cliente, permitindo que este realize as alterações que precisa enquanto se encontra desconectado. Posteriormente, quando o cliente se reconetar, as alterações efetuadas são propagadas da base de dados local para o servidor.

O principal objetivo da presente dissertação consiste na exploração de técnicas de sincronização e reconciliação de dados para a construção de aplicações desconectadas.

Para que se alcance o objetivo é necessário a realização de um conjunto de tarefas específicas, tais como:

- Realizar o levantamento das técnicas de sincronização e reconciliação de dados existentes;
- Apresentar e comparar as ferramentas existentes no âmbito da temática e estudar as vantagens relativas das técnicas de sincronização, e reconciliação de dados;
- Aferir a sua aplicabilidade em ambientes de computação cliente/servidor, multinível e nuvem, usando diferentes tipos de bases de dados (estruturadas e não-estruturadas);

- Estabelecer recomendações de uso e, eventualmente, propor novas abordagens para aplicações futuras;
- Criar uma aplicação que possa lidar com o processo de sincronização e reconciliação de dados em ambiente desconectado, transferindo e sincronizando grandes quantidades de dados da base de dados local para o servidor.

A presente dissertação está organizada do seguinte modo: após uma breve introdução, em que se descreve a caracterização da utilização da sincronização e reconciliação de dados, é apresentada, no Capítulo 2, uma visão geral sobre as abordagens referentes à sincronização e reconciliação de dados, assim como os tipos e métodos de replicação, os níveis de isolamento, as arquiteturas envolvidas no processo de sincronização e políticas de resolução de conflitos empregues. É, também, apresentado, de forma resumida, o processo de sincronização das bases de dados não estruturadas.

No Capítulo 3, apresenta-se a implementação de uma solução para o processo de sincronização e reconciliação de dados. É descrita a estrutura implementada para assegurar que a aplicação responde de forma adequada e que funciona corretamente.

No Capítulo 4, apresentam-se os resultados obtidos.

No Capítulo 5, apresentam-se principais conclusões e perspetivas de trabalhos futuros.

Capítulo 2

Estado do conhecimento

2.1 Introdução

No presente capítulo apresenta-se uma abordagem sobre os processos de sincronização e reconciliação utilizados na atualidade. De salientar que a problemática da sincronização e reconciliação de dados está fortemente ligada às alterações dos recursos das diferentes máquinas no ambiente do negócio da organização.

2.2 Fundamentos da sincronização e reconciliação de dados

Na revisão da literatura sobre engenharia de software, Kitchenham [8] sugere que os engenheiros de software devem abordar as evidências disponíveis para que sínteses de estudos e revisão sistémica de literatura científica correlacionada respondam a uma determinada temática.

Nesta perspetiva, houve a necessidade de se recorrer às fontes de informação que sustentam a sincronização e reconciliação de dados. Na análise, recorreram-se às fontes de documentação tais como: teses, dissertações, artigos, jornais, revista e livros.

Christiaan Heygens é apontada como a primeira cientista que observou e descreveu o processo de sincronização, no século XVII [9]. Tal descoberta teve um impacto no desenvolvimento tecnológico e científico, o que levou a um aumento significativo da exatidão da medição do tempo. Heygens ao descobrir e revelar a "simpatia de dois relógios", não deu somente uma descrição exata, mas também uma caracterização qualitativa brilhante da sincronização. Ela entendeu que a conformidade dos ritmos de dois relógios foi causada por um impercetível movimento, que na terminologia moderna significa que "os relógios estavam sincronizados".

2.3 Sincronização e reconciliação de dados

O surgimento de dispositivos com capacidade de processamento e largura de banda relativamente baixa, trouxe questões de gestão dos dados, em particular da exatidão de dados, em modo de desconexão. Este modo de operação permitiu que base de dados desconetadas incorporassem ações automatizadas para garantir a consistência de dados entre as máquinas locais e remotas [10].

A sincronização é uma atividade de transferência de cópias de estrutura e dados entre duas entidades. Também pode ser definida como a troca de registos entre duas bases de dados [11]. No âmbito das aplicações móveis podemos defini-la como o movimento de dados entre os dispositivos móveis, que realizam o armazenamento de dados remotos, e o servidor da base de dados corporativa, que faz a gestão e o armazenamento.

Em ciências da computação, a sincronização refere-se a dois conceitos diferentes, no entanto, relacionados. Estando a tratar-se de sincronização de processos e de dados, a sincronização de processos está relacionada com a ideia de que vários processos se unem em determinado ponto, para chegar a um acordo ou para se comprometerem com determinada ação. A sincronização de dados refere-se à ideia de manter cópias múltiplas de um conjunto de dados.

A utilização da sincronização e reconciliação em ambiente desconectado permite estabelecer consistência entre os dados armazenados em máquinas locais e em máquinas remotas. As políticas de reconciliação são projetadas para conciliar um único conjunto de dados entre dois ou mais dispositivos. Por exemplo, os contactos, email, ficheiros, e outros documentos.

A sincronização pode ocorrer de forma unidirecional ou bidirecional, em tempo real ou de forma periódica, de modo síncrono ou assíncrono. A sincronização assíncrona é adequada para o ambiente desconectado, onde os clientes podem desconectar-se da rede, permitindo manipulação dos dados no tempo e no espaço, isto é, onde e quando quiserem.

2.4 Abordagens sobre a sincronização e reconciliação

Diferentes formas de sincronização de dados podem ser definidas a partir de abordagens diferentes. A título contextual, o sistema proposto por Bayou [12], para suporte à colaboração entre utilizadores, centra-se na exploração de mecanismos que permitem ler e gravar ativamente os dados. Esta proposta assenta numa arquitetura baseada na divisão de funcionalidades entre clientes e servidores. Para Bayou, um servidor é qualquer máquina que contenha uma completa cópia de uma ou mais bases de dados. A arquitetura permite que qualquer membro do grupo tenha acesso a qualquer dado. Foca-se nos mecanismos específicos para as aplicações detetarem e resolverem conflitos de atualização.

Contrariamente a Bayou, Phatak e Nath [13] consideraram uma arquitetura cliente/servidor estendida, onde as cópias primárias são armazenadas no lado do servidor. Na eventualidade do cliente se encontrar desconectado, as transações são confirmadas localmente para que as suas atualizações estejam disponíveis para outras transações locais. Para estes autores, a reconciliação de dados pode ser processada em dois níveis de granularidade: itens de dados e transações. Propuseram um algoritmo que serializa as transações via versões confirmadas *(committed)*. Cada transação fornece um *timestamp* de exclusão integral, não negativo e exclusivo, na confirmação global (no servidor), que pode ser diferente do *timestamp* de início, possivelmente não exclusivo.

Numa outra perspetiva é abordada uma arquitetura que explora a semântica das aplicações, que divide objetos grandes e complexos em fragmentos menores, proposta por Saddik [14]. Os clientes são agrupados de acordo com determinados critérios, tendo em conta a região, data de acesso e consulta da aplicação. Tal arquitetura consta de três camadas: servidor de dados, replicação de servidores de grupos e dispositivo móveis. Para este autor, as réplicas são fragmentadas entre os servidores de réplicas de grupo (GRS's) de tal forma que cada GRS possui um esquema global que informa o nome dos GRS's que guardam o fragmento do mesmo objeto.

Wankhade *et al.* [15] apresentam um mecanismo de sincronização que permite a deteção de alterações e resolução de conflitos, através de um sistema *online* e *offline*, onde os dados são transferidos do servidor local para o servidor remoto. Caso a conexão seja perdida durante a transferência, os dados são guardados em servidores locais. Aquando da conexão o servidor local sincroniza os dados com o servidor remoto e se ambas as máquinas estiverem no modo *online*, a transferência de dados ocorrerá diretamente. No entanto, se uma das máquinas estiver no estado *offline*, o servidor local armazena os dados até que o outro dispositivo esteja no modo *online*.

As pesquisas no campo da sincronização e reconciliação de dados evoluíram com perspetivas diferenciadas, segundo o tempo e a visão de cada investigador. Estas perspetivas permitiram a mobilidade na forma de trabalho das organizações. Na sua pesquisa sobre "resolução de problemas em aplicativos através de sincronização de dados em caso de ausência da rede", *Shabani et al.* [16] apresentam o exemplo de um sistema eletrónico de gestão de estudantes (ESMS) que funciona com base no método de replicação otimista. Este sistema funciona em modo *online* quando há sinal de rede e *offline* em caso de ausência de rede. Os dados são transferidos entre servidores, que realizam o armazenamento e a gestão, e dispositivos móveis [16]. Tendo em conta o baixo processamento, a capacidade de energia e a baixa largura de banda, Choi *et al.* [17] desenvolveram o algoritmo de sincronização com base no *message digest* (*SAMD-Synchronization Algorithms based on Message Digest*) para a resolução de problemas de sincronização usando consultas SQL (*Structured Query Language*). Esta sincronização é feita com base em qualquer combinação de dados, independentemente do tipo de base de dados do lado do servidor ou do dispositivo, proporcionando estabilidade, adaptabilidade e flexibilidade. Este algoritmo compara as duas imagens para selecionar as linhas necessárias para sincronização. Esta comparação é feita com base no resumo *(hash)* das linhas envolvidas.

Os métodos de sincronização também podem depender dos sistemas de gestão de base de dados e da existência de ferramentas tais como procedimentos armazenados, *triggers, timestamp*. Kim [18] procura diferenciar um sistema integrado de gestão de base de dados incorporado, que realiza a gestão efetiva de pequenas bases de dados em dispositivos móveis, e um servidor DBMS (*Database Managment System*) que guarda uma base de dados de grandes dimensões. A reconciliação é efetuada com base nos registos do servidor, que possuem um campo adicional de *timestamp*. Este campo indica a última hora em que o

registo foi confirmado no servidor. As modificações e inserções dos dados são provenientes de pedidos de sincronização vindos dos clientes ou do servidor. Cada *timestamp* possui um significado exclusivo no seu registo. Esta proposta emprega a estratégia de transferir apenas os registos atualizados no cliente para o servidor de modo a minimizar a sobrecarga de comunicação.

Sedivy *et al.* [19] apresentam uma *Framework* para aplicações móveis para aumentar a tolerância a falhas, baseada numa base de dados *SQLite* apoiada na API *Google App Engine*, para aceder a bases de dados *Big-Table*. Utilizam a replicação como estratégia para distribuição de dados, tolerância a falhas e aumento de disponibilidade.

2.5 Replicação de base de dados

A replicação é um processo para melhorar a fiabilidade e a tolerância a falhas. Trata-se de replicação de dados quando estes são mantidos em dispositivos de armazenamento diferentes [20]. Trata-se de replicação de bases de dados quando existem várias cópias idênticas do objeto. Ao lidar com réplicas de bases de dados, várias ações são executadas durante no decurso de uma transação [21]. Podem definir-se cinco etapas:

1. Solicitação de sincronização;
2. Coordenação do servidor de sincronização;
3. Execução;
4. Coordenação;
5. Resposta do Cliente;

Segundo Page *et al.* [22], a replicação é fundamental para garantir a confiabilidade em ambiente onde podem ocorrer falhas de comunicação. A utilização de réplicas está vinculada ao problema da atualização. Podendo esta ser realizada em modo síncrono e assíncrono. A replicação síncrona garante a atualização dos dados de um modo seguro, consistente e confiável. A replicação síncrona pode, no entanto, afetar o desempenho das transações pois estas ficam mais propensas a bloqueios. A replicação assíncrona tenta suplantar estas limitações, sendo ideal para processamento *offline*. A utilização da replicação assíncrona pode, contudo, levar a perca de dados quando os clientes desconectados atualizam (a sua réplica d') os mesmos dados. Um dos clientes pode sincronizar primeiro levando a que os registos dos outros clientes sejam descartados [23][24][25].

As bases de dados relacionais utilizam trincos (*locks*) para impedir a inconsistência dos dados, restringindo as operações das unidades lógicas que podem levar a conflitos. As bases de dados replicadas permitem que os conflitos ocorram e depois resolvem-nos.

Quando se utiliza a replicação assíncrona, as operações de alteração dos dados são guardadas em tabelas de registo de operações no servidor. Estas alterações são chamadas de "transações diferidas". As transações diferidas são processadas localmente e, quando confirmadas, são disseminadas, periodicamente para reconciliação [23][25][26].

A replicação de atualização diferidas tem algumas vantagens mais também tem desvantagem:

- Vantagens:

    1. Melhor desempenho;
    2. Baixa taxa de *deadlock*;
    3. Tolerância a falhas.

- Desvantagens:

    1. Altas taxas de transações abortadas;
    2. Ponto único de falha.

Um impasse (*deadlock*), ver Figura 2.1, pode ocorrer quando uma transação entra em estado de espera quando solicita um recurso detido por outra transação. Em bases de dados replicadas os mesmos dados podem residir em vários locais, oferecendo um ambiente de partilha de recursos. Este ambiente global e local ocasiona conflitos que resultam em bloqueios. Na figura 2.1 a transação T1 bloqueou o recurso x e, a transação T2 bloqueou o recurso y. Portanto as transações estão à espera uma da outra. Ao nível das bases de dados são, comummente, usadas três técnicas para lidar com impasse (*deadlock*) [23][27].

1. Evitar *deadlock*: Os *deadlock* são tratados antes de sua ocorrência. Permite que a transação aguarde o término da transação anterior.

2. Prevenção de *deadlock*: É uma abordagem que impede que o sistema confirme a alocação de recursos que acabará levando a um *deadlock*.

3. Deteção de *deadlock*: Nessa abordagem, o sistema permite a ocorrência de *deadlock*. A técnica de deteção de *deadlock* tenta detetá-lo por forma a executar um procedimento que visa recuperá-lo.

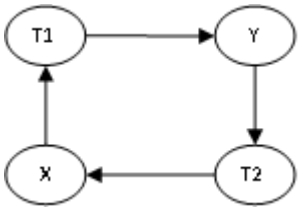

Figura 2.1: Exemplo de *deadlock*

Existem vários mecanismos para prever a existem de *deadlocks* ao nível das bases de dados. A sua utilização tem em conta a carga de serviço utilizada por cada participante no processo de sincronização, isto é, quantas transações por segundo são executas em cada réplica, qual o tempo de execução de cada transação e que ações são tomadas em cada réplica.

A melhoria do desempenho é conseguida com base na utilização das réplicas armazenadas localmente, permitindo que as solicitações sejam satisfeitas sem a necessidade de acesso ao servidor remoto [28].

No contexto da replicação de dados, existem duas abordagens a serem consideradas. A abordagem tradicional que consiste na separação dos objetos e da base de dados, onde os dados são alojados numa base de dados num servidor, designada por *back-end*, sendo os restantes componentes *front-end* (consultas, formulários, relatórios e outros) separados destes. Na segunda abordagem o servidor de base de dados contém os dados e os objetos. Nesta abordagem cada cliente pode receber uma réplica com um subconjunto de dados e objetos sendo o servidor responsável pela replicação [20].

2.5.1 Replicação em aplicações móveis

O fundamento da replicação na computação móvel está centrado na manutenção das funcionalidades oferecidas por dispositivos que possibilitam os utilizadores deslocarem-se para diferentes localizações. A necessidade de manter a conectividade entre os dispositivos móveis e o servidor central de forma transparente leva a um aumento de tráfego entre o terminal móvel e o servidor remoto. Esta carga pode ser reduzida com a utilização de réplicas.

2.5.2 Esquemas de replicação e categorização

A utilização da replicação em aplicações está relacionada com a capacidade de criar cópias de uma parte dos objetos da aplicação ou de todo o conjunto [29]. Com base na literatura e na pesquisa efetuada, Os protocolos de replicação no SGBD são classificados com base em critérios que especificam o momento em que é efetuada a atualização e o utilizador que atualizou [1]. Estas categorizações são [1] [28].

- Cópia primária ansiosa;
- Fácil atualização em qualquer lugar;
- Cópia primária preguiçosa;
- Atualização preguiçosa em qualquer lugar.

O processo de atualização nos protocolos de replicação pode ser controlado de duas formas. Na primeira, todas as atualizações partem de uma cópia primária, e na segunda, todas as atualizações emanam de qualquer cópia. No primeiro caso as atualizações ocorrem em

primeiro lugar na cópia primária, que se propagam ansiosamente para cada cópia secundária. Em segundo lugar uma atualização é permitida em qualquer lugar, mas propagadas para lugares remotos. Na replicação ansiosa mesmo que as réplicas estejam conectadas, as atualizações podem falhar devido à presença de *deadlocks*. De modo geral este protocolo de replicação apresenta os seguintes problemas [7]:

1. Ausência de esquema ansioso durante o período de desconexão;
2. Aumento da probabilidade de *deadlocks*.

Todo esquema de replicação deve alcançar quatro objetivos:

1. **Disponibilidade e escalabilidade**: Permitir alta disponibilidade e escalabilidade, evitando a instabilidade;
2. **Adaptabilidade**: Permite que a réplica modifique os seus dados;
3. **Seriabilidade**: Fornecer seriabilidade a execução das transações de cópia única;
4. **Convergência**: Os dados refletem o estado integral das réplicas.

Estes objetivos são alcançados com base nas propriedades de esquemas de replicação de duas camadas [7]:

1. As atualizações das réplicas são alcançadas com base em tentativas;
2. O estado do servidor resulta de uma execução de serialização de cópia única;
3. Uma transação é durável devido à transação base;
4. As réplicas convergem para o estado do servidor;
5. Ausência de reconciliação devido à comutação de transações.

Na base da uniformidade, os mecanismos de replicação são categorizados em homogéneos e heterogéneos. Esta categorização dos mecanismos de replicação no que diz respeito aos protocolos, implicam esquemas para a implementação de réplica em SGBD's: réplicas físicas; replicação baseada em *threads* e réplica baseada em *logs*.

A replicação física de base de dados não é muito prática devido ao elevado volume de dados que envolve, bem como à limitada largura de banda que constitui o centro da sincronização. A replicação baseada em *threads* permite resolver o problema, no entanto, diminui o desempenho resultando, deste modo, na diminuição da disponibilidade. O único método possível de ser utilizado sem os constrangimentos das situações expostas é a replicação de *logs*. O método de replicação de *logs* emprega os mecanismos dos SGBD e assegura a compatibilidade das transações usando a lógica dos *logs*.

2.5.3 Semântica de replicação baseada em *logs*

Em [1], apresenta-se o problema da heterogeneidade no mecanismo de replicação baseado em *log* com base em duas fases: a fase de configuração e de utilização. A primeira fase começa com a determinação das bases de dados que participam na replicação, e inclui a produção de um conjunto de regras de semântica(ontologia) aplicar a cada *log*. Esta ontologia deverá ser produzida por um mecanismo integrante do serviço de replicação do SGBD. O produto do mapeamento de ontologia é registado num repositório de regras de mapeamento que permite aproveitar a aplicação de regras na fase de utilização. A fase de utilização assegura a conformidade do registo, de acordo com as regras de mapeamento, antes de enviá-lo para realizar atualizações - ver Figura 2.2.

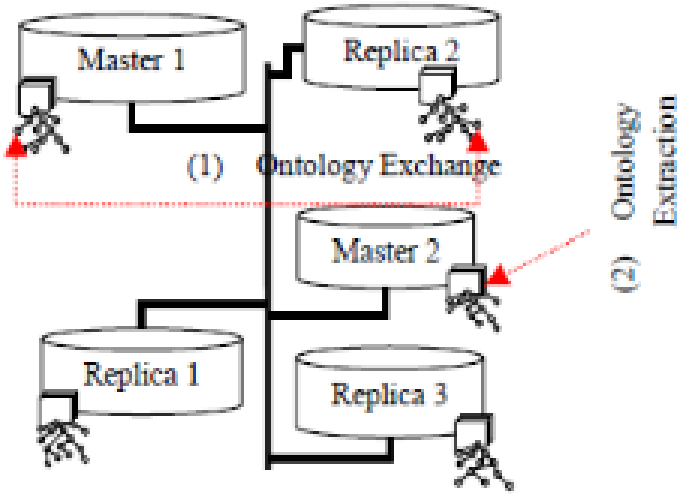

Figura 2.2: Extração e troca de Ontologia [1]

2.5.4 Protocolos de implementação das réplicas

A implementação dos mecanismos de replicação envolve a utilização de protocolos de replicação que garantam a consistência das réplicas. O ponto nevrálgico, e que constitui o desafio da utilização da replicação é a estratégia usada para manter a consistência mútua entre as diferentes réplicas que participam no processo. Segundo [28][30], existe uma correlação entre a consistência mútua e a consistência transacional. A consistência mútua consiste em que os dados replicados pelas diferentes réplicas convergem para o mesmo recurso. A consistência transacional está ligada ao histórico das transações de dados replicados, que é serializável se for equivalente a um histórico em série [28][30].

Existem duas formas principais de implementar um protocolo de replicação: replicação baseada em *kernel* e em *middleware* [28].

2.5.4.1 A replicação baseada em *kernel*

Na replicação baseada em *kernel* o acesso aos objetos da base de dados é efetuado diretamente, sendo bastante fácil acoplar duas réplicas. Estas réplicas utilizam o mecanismo de controlo de concorrência com processamento transacional. Nesta, as réplicas coordenam as suas operações para fins de controlo. Os clientes interagem apenas com uma base de dados central que efetua atualização dos dados e os envia às réplicas.

2.5.4.2 Replicação baseada em *middleware*

Contrariamente à replicação baseada no *kernel* a replicação em *middleware* consiste em encapsular a lógica de replicação. O *middleware* possui a responsabilidade de coordenar as solicitações de clientes e proporcionar ao utilizador a perspetiva de um sistema único. Cada réplica é uma base de dados sem conhecimento de replicação. Do ponto de vista da base de dados, o *middleware* é simplesmente um cliente normal. O servidor de replicação implementa o mecanismo de controlo de concorrência para lidar com acesso simultâneo.

Os utilizadores solicitam leitura, escrita, confirmação e cancelamento ao *middleware* em vez da base de dados. O *middleware* possui a responsabilidade de cuidar da coordenação da execução das transações nas diferentes réplicas, de modo a garantir que a propagação das informações seja equivalente em relação à cópia fornecida.

2.6 Modelo Relacional

Existem três tipos de modelos de dados: relacional, de rede e hierárquico. Há ainda quem considere um quarto modelo, orientado a objetos. O modelo mais comummente encontrado nas organizações é o modelo relacional.

Um modelo de dados é uma coleção de, pelo menos, três componentes: um conjunto de tipos de estruturas (define os tipos de dados e como se relacionam), um conjunto de operadores (operações que permitem manipular as estruturas de dados definidas) e um conjunto de regras de integridade (regras que definem que dados são válidos).

O modelo relacional, proposto por Edgar Codd [31], baseia-se no pressuposto de que os dados (que obedecem a certas restrições) podem ser tratados da mesma forma que as relações matemáticas. No modelo relacional, os objetos e as suas associações são designados por entidades. Cada entidade é caracterizada por um conjunto de atributos. Os atributos podem ser de dois tipos: descritores ou identificadores. Toda a relação tem uma chave primária, que é a chave eleita entre as chaves candidatas. Uma chave candidata é um conjunto mínimo de atributos com capacidade para determinar os restantes atributos da relação. Uma base de dados relacional é uma coleção de relações cujo conteúdo varia ao longo do tempo.

Em termos práticos, uma relação pode ser representada por uma matriz (tabela) com colunas e linhas, correspondendo as colunas aos atributos e as linhas às ocorrências/instâncias

da entidade representada pela relação. Os relacionamentos, ou associações, entre entidades são representados, em termos práticos, com recurso ao conceito de chave estrangeira.

Ainda em termos práticos, cabe ao sistema de gestão de base de dados relacional garantir a integridade e a consistência dos dados armazenados na base de dados. A integridade referencial é concretizada com recurso a chaves primárias e estrangeiras. A chave primária serve para identificar cada linha inserida na tabela, constituindo um desafio a ser superado no contexto da utilização de replicação em ambiente desconetados. Uma chave estrangeira é um conjunto de atributos que constituem uma chave primária noutra tabela, permitindo estabelecer a associação entre as duas tabelas.

2.7 Reconciliação

As bases de dados com uma topologia central evitam os conflitos, enquanto as bases de dados replicadas têm que detetar conflitos e resolvê-los. A reconciliação é o conjunto de condições que permitem manter os estados inicial e finais consistentes entre o conjunto de réplicas. Para reconciliar é necessário definir um conjunto de políticas de resolução de conflitos. Uma política de resolução de conflito define que critérios de operação devem ser seguidos na eventualidade de ocorrer um conflito. As operações efetuadas no mesmo recurso, em diferentes horários, definem os conflitos [32].

O processo de resolução de conflitos consiste em duas fases:

- Fase 1: Consiste na tomada de decisão de que dados ganham o conflito, os do servidor ou os do cliente;
- Fase 2: Consiste em atualizar os dados de acordo com a política do servidor.

2.8 Transações em base de dados

Haerder e Reuter [33], a manipulação de dados num ambiente multiutilizador requer algum isolamento para evitar operações não controladas. A realização de uma operação, em determinado momento, coloca a base de dados a um estado inconsistente, caso a unidade de trabalho não conclua. Se os utilizadores tomarem decisões com base em dados não confirmados os resultados poderão ser catastróficos. Desta forma, é preciso refletir sobre circunstância forma como é tratada a entrada/saída de dados, como e quando recursos se sobrepõem, e em que circunstância são reprocessadas as operações afetadas.

Haerder e Reuter, Bjork [34] e Davies [35] debruçaram-se sobre este assunto nos seus estudos sobre as chamadas esferas de controlo, mencionando que os dados que estão a ser operados por uma transação devem ser isolados de maneira a garantir a consistência [33].

2.8.1 Propriedades das transações

Reflete a ideia de que as atividades de um determinado utilizador estão isoladas das atividades concorrentes de outros utilizadores. Uma transação pode envolver uma sequência de interações com a base dados, usando interrogações que visam consultar ou modificar registos. O conceito de transação corresponde a um conjunto de operações, bem delimitado, que possui as seguintes propriedades:

- **Atomicidade**: O conjunto de operações que compõem a transação é indivisível (atómico). Todas as operações de uma transação têm de ser executadas com sucesso (faz o *commit*) ou todas as suas ações sobre a base de dados são desfeitas (faz o *rollback*).
- **Integridade**: Uma transação atingindo o seu fim normal preserva a consistência, isto é, a transação leva a base de dados de um estado consistente a outro.
- **Isolamento**: O sistema deve dar a ilusão de que cada transação é a única a executar. Quando várias transações concorrentes acedem aos mesmos dados, deve ser garantido que o resultado é o mesmo se as transações executassem em série.
- **Persistência**: Todos os efeitos provocados por uma transação bem-sucedida devem ser persistentes, sobrevindo mesmo a falhas posteriores. Os efeitos de uma transação bem-sucedida só podem ser desfeitos por outras transações posteriores.

As bases de dados multiutilizador estão sujeitas a várias transações em simultâneo, portanto, O SGBD deve implementar uma política de controlo para garantir a serialização e o isolamento de transações [36].

2.8.2 Gestão de transações

Segundo o *ANSI* [2], o *terminus* de uma transação é concretizado com recurso a um dos comandos: *COMMIT ou ROLLBACK*. O processamento de uma transação faz-se de modo sequencial, operação a operação, até que ocorra uma das seguintes situações [36][23]:

- Encontrar a instrução *COMMIT*. Note-se que apesar de ser solicitada a efetivação/confirmação da transação, ainda é possível esta não seja atingida, sendo interrompida;
- Atingir a um *ROLLBACK*. Uma transação anulada não produz efeitos sobre a base de dados, portanto, qualquer modificação deve ser desfeita e a base de dados retorna ao seu estado consistência;
- Passa ao estado confirmado quando as informações são registadas em disco permanentemente;
- Uma transação encontra-se no estado anormal quando há falhas provenientes da execução normal do hardware ou erros de programa.

[2] Instituto Nacional Americano de Padrões

2.9 Controlo de concorrência

O processo de coordenação da execução simultânea, e sem interferências, das operações das transações, é conhecido como controlo de concorrência. O principal objetivo do controlo de concorrência é garantir a serialização das transações, pois o acesso concorrente a dados partilhados podem dar origem a diversos problemas de integridade e consistência. O acesso simultâneo é benéfico quando envolve apenas leitura de dados. Os problemas de consistência podem ocorrer quando há operações de escrita envolvidas. Uma transação, ainda que correta, se executada individualmente, pode ser afetada por três tipos de interferências:

1. As atualizações perdidas;
2. Os dados não consolidados; e,
3. As leituras inconsistentes.

2.9.1 Atualização perdidas

As atualizações perdidas acontecem quando duas transações concorrentes, atualizam o mesmo recurso e as atualizações de uma delas são sobrescritas pela outra. Por exemplo, supondo que existem 100 unidades de dado produto em armazém e que são submetidas duas transações T_1 e T_2, onde T_1 corresponde à venda de 35 unidades e T_2 corresponde à aquisição de 25 unidades do produto considerado. Portanto, duas transações concorrentes vão escrever um dado partilhado (quantidade disponível de produto no armazém). Um escalonamento correto destas transações produzia o resultado correto (90 unidades) está representado na Tabela 2.1.

Tabela 2.1: Execução em série de transações

Tempo	Transação	Recurso armazenado
1	T_1: Lê recurso	100
2	T_1: Processa recurso=100-35	
3	T_1: Escrita do recurso	65
4	T_2: Lê recurso	65
5	T_2: Processa recurso=recurso+25	
6	T_2: Escrita do recurso	90

O problema das atualizações perdidas, representado na Tabela 2.2, surge devido à execução concorrente das operações que constituem cada transação. Neste caso, a transação T_1 executou a primeira operação (de leitura) e foi interrompida (pelo escalonador). A transação T_2 começou a executar e executou também a sua primeira operação (de leitura). Nesta altura as duas transações consideram que o valor corrente do produto em armazém é de 100 unidades (o que se assume como correto). De seguida a transação T_1 faz a sua atualização e escreve o resultado (65, neste caso). Quando a transação T_2 faz o seu trabalho de atualizar e escrever o resultado da atualização, sobrepõe o resultado da transação T_1, perdendo-se este. Esta situação deveu-se ao facto da transação T_2 estar a trabalhar

com uma imagem (100 unidades) desatualizada. Este escalonamento das operações que compõem as transações T_1 e T_2 não é aceitável (não é um escalonamento serializável).

Tabela 2.2: Atualizações perdidas

Tempo	Transação	Recurso armazenado
1	T_1: Lê recurso	100
2	T_2 : Lê recurso	100
3	T_1: recurso=recurso-35	
4	T_2: recurso=recurso+25	65
5	T_1: Escrita	65
6	T_2: Escrita	125

2.9.2 Dados não consolidados

O fenómeno de dependência não confirmada ou consolidada ocorre quando uma transação pode aceder aos resultados intermédios (não confirmados) de outras transações, violando, assim a propriedade do isolamento de transações. Continuando com o exemplo anterior, supondo que a transação, T_1 atualiza o valor da quantidade(decrementado) em 35 unidades, mais ainda não confirmou, e que a transação T_2 faz a sua leitura (obtendo o valor de 65). Se a transação T_1 fizer o *rollback*, a transação T_2 vai prosseguir a sua operação com dados erróneos - ver Tabela 2.3.

Tabela 2.3: Dados não consolidados

Tempo	Transação	Recurso armazenado
1	T_1: Lê recurso	100
2	T_1: recurso =recurso-35	
3	T_1: Escreve recurso	65
4	T_2: Lê o recurso	65
5	T_2: recurso=recurso+25	
6	T_1: *ROLLBACK*	100
7	T_2: Escreve o recurso	90

2.9.3 Leituras inconsistentes

As leituras inconsistentes acontecem quando durante uma transação são feitas duas leituras de um mesmo atributo e se obtém valores diferentes. Esta interferência entre transações pode ocorrer quando entre as duas leituras efetuadas no âmbito de uma transação, há uma alteração provocada por outra transação, que, entretanto, confirmou [36][28].

2.10 Escalonador

A execução concorrente de transações abre a possibilidade para conflitos. A resolução destes conflitos é tarefa do SGBD's, em concreto, é da responsabilidade do escalonador, que estabelece a ordem de execução e garante a serialização e o isolamento das transações empregando para o efeito, trincos e selos temporais.

2.11 Controlo de concorrência com métodos de bloqueios

Os métodos de bloqueio constituem a premissa utilizada para garantir a serialização de transações concorrentes. No intervalo de tempo em que uma transação precisa de um recurso, deve ser garantido que este recurso não modifica enquanto estiver a ser utilizado pela transação que o bloqueia. Um trinco é adquirido antes de aceder aos dados quando a transação é concluída [28][37].

2.11.1 Tipos de trincos

Num SGBD é possível encontrar três tipos de trincos: binário, partilhado e exclusivo.

2.11.1.1 Trinco binário

O trinco binário apresenta dois estados, bloqueado (*lock*) e desbloqueado (*unlock*). Quando o estado do recurso estiver bloqueado (*lock*), este não poderá ser acedido por outra transação. O trinco binário pode ocorrer ao nível da base de dados, tabela, página ou linha. O trinco binário limita a concorrência [28][29].

2.11.1.2 Trinco exclusivo

Os trincos exclusivos, quando empregues, obrigam as transações a aguardar a libertação dos recursos bloqueados antes de poderem fazer as suas atualizações [38].

2.11.1.3 Trinco partilhado

O trinco partilhado permite as transações concorrentes terem acesso de leitura a um recurso partilhado. Este trinco é utilizado quando uma determinada transação deseja aceder a um recurso sem que outra tenha adquirido um trinco exclusivo.

O trinco partilhado é permitido a qualquer transação que queira ler um recurso. Contrariamente ao trinco exclusivo que só é concedido se nenhuma transação estiver vinculada sobre o recurso [28] [37]. Os trincos são utilizados da seguinte forma:

- A transação que deseja utilizar um recurso, está obrigada em primeiro lugar a bloquear o recurso, quer solicite um trinco partilhado ou exclusivo;
- Na eventualidade do recurso não estar bloqueado por outra transação, o trinco é concedido à transação que o solicitou;
- Na eventualidade do recurso se encontrar bloqueado, o SGBD determina se o pedido é compatível com o trinco existente. Se o pedido for de um trinco partilhado e a transação solicitante desejar o mesmo tipo de bloqueio, a solicitação é concedida; caso contrário, a transação deve esperar até que o bloqueio existente termine;
- A transação manterá um trinco até libertá-lo explicitamente durante a execução ou no seu término.

2.11.2 Bloqueio de duas fases

Para garantir a serialização, as operações de bloqueio e desbloqueio das transações devem seguir protocolos. O protocolo mais usado é o protocolo de bloqueio em duas fases. Este protocolo define como as transações devem adquirir e liberar os trincos. Este protocolo é composto por duas fases [28]:

- Fase 1: Consiste na aquisição de todos os trincos solicitados;
- Fase 2: Consiste na libertação de todos os trincos adquiridos e na negação de obtenção de novos trincos.

Este protocolo é determinado pelas seguintes regras [28]:

- Não utilizar duas transações conflitantes;
- Nenhuma operação de desbloqueio deve preceder uma operação de bloqueio na mesma transação;
- Nenhum dado deve ser modificado até que todos os trincos sejam obtidos.

2.11.3 Métodos de replicação otimista

A replicação otimista baseia-se no propósito de submeter a solicitação de sincronização após a conclusão da transação e de todas as atualizações escritas. A replicação otimista permite que cada cliente complete o processamento de dados e em seguida faça a uniformização do controlo de consistência. Cada transação passa por três fases:

1. **Fase de leitura** - A transação lê a base de dados, executa as operações necessárias e faz as atualizações na réplica privada;
2. **Fase de validação** - A transação é validada e garantido que as alterações não afetem a integridade e a consistência;
3. **Fase de gravação** - Consiste na gravação de forma permanente das alterações.

2.11.4 Métodos de *timestamp*

O *timestamp* é um método de controlo de concorrência, onde o acesso a um determinado recurso é realizado apenas uma vez. Existe apenas uma transação que utiliza um determinado recurso. Este método diferencia-se dos anteriores pela não utilização de bloqueios evitando longas filas de espera. O funcionamento é feito através de um identificador único, criado pelo SGBD, que controla a execução de todas as transações concorrentes para garantir a consistência [28][37].

A leitura ou escrita de recursos utilizando o método *timestamp* é permitida se a última atualização do recurso foi realizada por uma transação que o antecede. Não sendo, a transação é revertida para que receba um novo *timestamp* reiniciando a transação. Para

cada transação iniciada, é associado um *timestamp* fixo exclusivo. Antes que uma transação tenha início, o SGBD fornecerá um tempo de referência. O método de *timestamp* apresenta duas propriedades fundamentais: exclusividade e a monotonicidade. A exclusividade garante a não existência de valores iguais de registo e a monotonicidade assegura que os valores sejam sempre crescentes [37].

Possibilidades de implementação:

- Utilizar o tempo do sistema como sendo o *timestamp*. Portanto, a hora de início da transação será igual à hora em que a transação chega ao sistema.
- Usar um contador que é incrementado sempre que a base de dados é atualizada e as resoluções de conflitos são efetivadas, exigindo a utilização de valores máximos e mínimos do contador.

A utilização do método *timestamp* sugere dois esquemas para decidir qual transação é revertida e qual continuará a ser executada. Estes esquemas são: o esquema *wait/die* e o esquema *wound/wait*[37].

2.12 Níveis de Isolamento

O desenvolvimento de aplicações sobre bases de dados implica o conhecimento dos níveis de isolamento das transações. O nível de isolamento define em que grau uma transação está exposta a modificações feitas por outras transações concorrentes [39].

Os SGBD's usam níveis de isolamento e trincos para controlar o grau de concorrência, cabendo, ao programador decidir qual é o nível que mais se adequa à sua aplicação [39].

A maior parte das aplicações desenvolvidas não utiliza o nível de isolamento mais restrito, permitindo assim uma concorrência mais elevada. Quanto mais restritivo for o nível de isolamento utilizado, menor será a concorrência efetiva e maior será a probabilidade de o sistema entrar em *deadlock*. Um *deadlock* segundo [28], constitui um impasse em que duas ou mais transações esperam indefinidamente por recursos bloqueados por outras transações.

Todo nível de isolamento deve controlar:

1. Quando é efetuado o bloqueio, quando os dados são lidos e que bloqueio é solicitado;
2. Durante quanto tempo são mantidos os trincos de leitura;
3. Quando é que os dados provenientes de uma operação de leitura foram modificados por outra transação. Isto implica:
    - Bloquear a transação até que o bloqueio exclusivo seja desbloqueado;

- Recuperar a versão confirmada da linha existente no momento em que a transação foi iniciada e;
- Ler as alterações de dados não confirmados.

As especificações *ANSI/ISO SQL2* definem quatro níveis de isolamento [39][40]:

1. *Read uncommitted*;
2. *Read committed*;
3. *Repeatable read*;
4. *Serializable*.

A escolha do nível de isolamento depende do uso previsto para o mesmo. Um nível de isolamento mais baixo, está mais sujeito a efeitos colaterais e, portanto, com mais leituras sujas (*dirty reads*) e mais atualizações perdidas, mas permite mais concorrência. Um nível de isolamento mais elevado reduz concorrência, requer mais recursos do sistema e aumenta as hipóteses de *deadlock*[41] .

2.12.1 Read uncommitted

Constitui o menor nível da hierarquia. Neste nível, uma transação T pode ler alterações feitas por transações que ainda estão a decorrer. Naturalmente, os dados lidos podem ainda sofrer outras alterações enquanto T está em progresso.

2.12.2 Read committed

Este nível é o nível pré-definido (por exemplo, *para o Oracle e Microsoft SQL Server).* Este nível assegura que uma transação T só pode ler dados confirmados. Para além disso assegura que nenhum valor escrito por T é alterado por outra transação até T terminar. Note-se que um valor lido por T pode ser modificado por outra transação durante o período de vida de T.

2.12.3 Repeatable read

À semelhança do nível *Read committed*, este nível assegura que a transação T só faz leituras de dados confirmados. Para além disso assegura que nenhum valor lido ou escrito por T é alterado por outra transação até T terminar.

2.12.4 Snapshot

Este nível de isolamento determina que os recursos lidos pela transação formam uma versão íntegra de dados transaccionalmente consistente desde o início transação. Uma transação executada neste nível nunca é bloqueada ao tentar uma leitura. Pois, reconhece apenas modificações confirmadas. Portanto, não bloqueia modificações efetuadas por outras, por que, não são visíveis para a transação corrente. Este isolamento é utilizado em aplicações onde o nível das operações é reduzido.

2.12.5 Serializable

Neste nível, todas as transações são isoladas e executadas de modo sequencial. As transações não podem ler dados modificados ainda não confirmados por outras transações. Este nível obedece a alguns princípios como:

1. Nenhuma transação pode modificar dados lidos pela transação atual até à sua confirmação.

2. Outras transações não podem inserir novos recursos que estejam no intervalo de chaves, lido por qualquer transação, até que esta seja concluída.

3. Bloqueia todos os recursos em uso até que conclua a serialização.

2.12.6 Tipos de problemas de concorrência

A execução concorrente de transações tem de ser gerida para evitar problemas no acesso a dados. Os tipos de problemas (anomalias) a que uma dada aplicação está exposta depende do nível de isolamento escolhido para a ligação à base de dados. As anomalias que podem surgir a cada nível de isolamento são [39][40]:

- *Dirty reads* (leitura suja): já visto anteriormente.
- *Lost update* (atualização perdida): já visto anteriormente.
- *Non-repeatable read* (leitura não reutilizável): já visto anteriormente.
- *Phantoms read* (leitura-fantasma): Ocorre quando uma transação executa a mesma consulta em dois momentos e obtém resultados diferentes, apesar de ela própria não ter alterado esses dados.

Tabela 2.4: Anomalias associadas aos níveis de isolamento

Nível de Isolamento	Anomalias			
	Dirty Reads	*Lost Update*	*Non-repeatable read*	*Phantoms read*
Read Uncommitted	Sim	Sim	Sim	Sim
Read Committed	Não	Sim	Sim	Sim
Repeatable Read	Não	Não	Não	Sim
Serializable	Não	Não	Não	Não

2.13 Arquitetura

Uma arquitetura é o modo como os dispositivos interagem para formar um sistema. Ao nível dos sistemas informáticos é comum encontrar três tipos de arquitetura: cliente/servidor, *peer-to-peer* e agentes de replicação.

2.13.1 Cliente/Servidor

Na arquitetura cliente/servidor, representada na Figura 2.3, o cliente solicita recursos e o servidor, processa os pedidos e disponibiliza os recursos. Na construção de aplicações usando esta arquitetura é necessário identificar as funções do cliente e as funções do servidor.
Ao nível das bases de dados relacionais, frequentemente, o SGBD encarrega-se da gestão das transações, do controlo de acesso, da recuperação em caso de falhas e da otimização de consultas de transações. Na arquitetura cliente/servidor um mesmo dispositivo pode desempenhar as funções de cliente e de servidor. No entanto, na maioria dos casos, as funções estão em dispositivos diferentes [30].

O servidor é responsável pela verificação da consistência das réplicas e retornar o resultado ao cliente. A distribuição das réplicas traz vantagens, uma vez que as cópias manipulam um subconjunto dos dados do servidor.

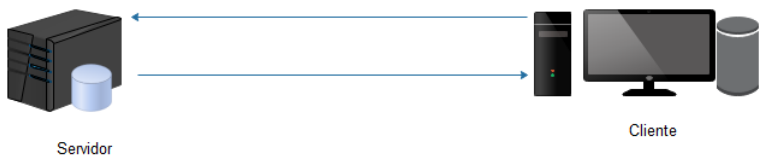

Figura 2.3: Arquitetura Cliente/Servidor

2.13.2 *Peer-to-Peer*

Nesta arquitetura não existe a diferenciação entre as funcionalidades das máquinas clientes e servidores, existindo uma comunicação direta entre os parceiros envolvidos. Portanto, cada dispositivo atua tanto como cliente como servidor - ver Figura 2.4. A arquitetura permite que os clientes comuniquem entre si, no entanto, a mobilidade afeta o desempenho da comunicação por estar diretamente relacionada com a qualidade do sinal disponível [30][25].

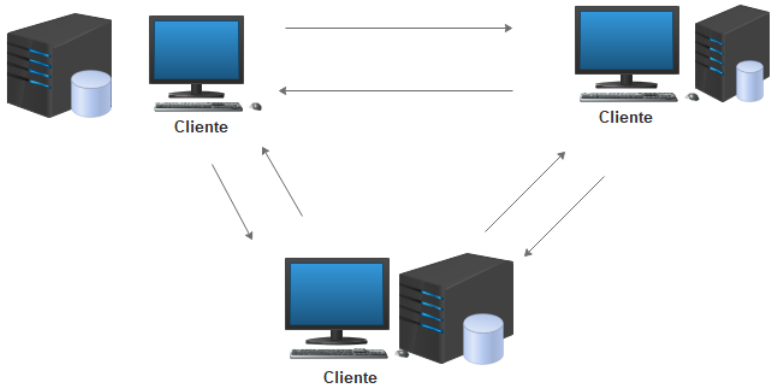

Figura 2.4: Arquitetura *Peer-to-Peer*

2.13.3 Agentes de replicação

Um agente móvel é qualquer aplicação, ou programa, auto controlável que reage a eventos externos. Para executar tarefas complexas transfere estes eventos para um servidor remoto para sua execução. A utilização de agente móvel visa substituir a atual arquitetura cliente/servidor - ver Figura 2.5. Sendo a forma mais flexível e eficiente de comunicação, por proporcionar proximidade à origem de dados. A utilização de agentes móveis oferece suporte a réplicas e a transações com base nas seguintes propriedades [42]:

1. Mobilidade;
2. Comunicação;
3. Identificação e Localização;
4. Segurança.

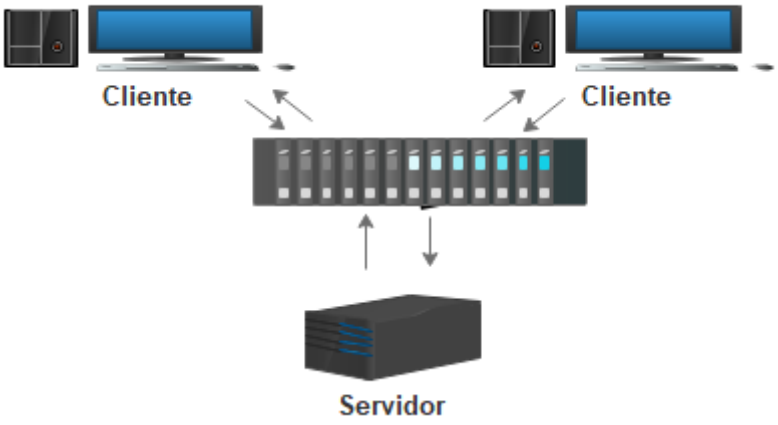

Figura 2.5: Arquitetura agente de replicação

2.14 Sincronização em base de dados não estruturadas

De acordo com alguns estudos [43], mais de 80% dos dados produzidos pelas empresas encontram-se em formato não-estruturado. Estes consistem em ficheiros de texto, e-mail, multimédia e imagens. A sincronização deste tipo de dados encerra encargos elevados em comparação com dados estruturados.

O armazenamento e o processamento de dados não-estruturados requerem novas abordagens. As bases de dados *NoSQL* [44][45], foram projetadas para manipular com facilidade dados não-estruturados, e apresentam boa escalabilidade. Atualmente encontram-se disponíveis várias bases de dados *NoSQL*[46]:

1. *Amazon DynamoDB*: Modelo de base de dados *NoSQL* da Amazon que fornece um serviço rápido de documentos, gráficos e colunas, confiável e económico, onde o

utilizador armazena dados em tabelas e interage por meio de consultas. Os dados são replicados de maneira síncrona para fornecer alta disponibilidade e durabilidade.

2. *MongoDB*: Base de dados de alto desempenho orientado a documentos com recurso a tolerância a falhas, adequado para aplicações como sistemas de gestão de conteúdo e análise em tempo real, sendo atualmente utilizado pela MTV, *Foursquare, The Guardian*. Evita a estrutura baseada em tabela. É utilizado para alta disponibilidade, desde instalações com um só servidor até infraestruturas grandes e complexas.

3. *Cassandra*: Oferece recursos como alta disponibilidade, tolerância, persistência e alta escalabilidade. É utilizado por uma variedade de aplicações, como sites de redes sociais, bancos e finanças e análise de dados em tempo real.

4. *O CouchDB*: Incorpora um protocolo de replicação e sincronização. É utilizado em casos em que a conexão de rede pode não estar disponível, sendo que a aplicação deve continuar a funcionar, como no caso de aplicativos baseados em dispositivos móveis [47].

As bases de dados *NoSQL* possuem desempenho, escalabilidade e disponibilidade que não são alcançadas pelas bases de dados estruturadas, para as quais o SQL não fornece capacidade computacional e de armazenamento suficientes, por não serem adequados neste domínio, pois, as junções e os bloqueios influenciam o desempenho. Cada sistema de gestão de base de dados possui características distintas. O modo de operação de cada sistema de gestão de base de dados é influenciado pela degradação de recursos, que é um dos fatores que influenciam a sincronização das bases de dados heterogéneas. Neste caso, constituem fatores de influência os seguintes [48]:

1. A diferenciação de fabricantes;

2. A forma de detenção da inconsistência;

3. O modo de processamento de dados;

4. A forma de resolução de conflitos.

Cada ferramenta possui a solução específica para a sincronização de dados. Em qualquer processo de sincronização existem sempre um cliente e um servidor que detêm um conjunto de recursos a serem transmitidos ou recebidos. Existem várias abordagens para a sincronização e a reconciliação. Uma possível solução para implementação da sincronização em base de dados heterogéneas, consiste na utilização de um valor de *hash* por forma a evitar a dependência do fornecedor na utilização da sincronização. Neste caso, é preciso conhecer (e obter) o conjunto de dados a serem sincronizados e partir deste é gerado o resumo *hash*. Obtidos os resumos do cliente e do servidor, verifica-se a existência de inconsistências, comparando os dois valores de *hash*. A comparação permite saber a existência ou não de alterações desde a última sincronização efetuada.

O desenvolvimento de aplicações com base *NoSQL* está dependente do tipo de aplicação que se quer desenvolver, sendo útil quando há a necessidade de processamento de grandes quantidades de dados. Os dados são armazenados como documentos completos que contêm uma chave (ID) que representa uma identificação única dentro de uma coleção de documentos [45].

A pesar do grande volume de dados, da disponibilidade e da tolerância das bases *NoSQL*, segundo Anant Jhingran, citado por Leu Garber [45], a adoção do *NoSQL* será de pequena escala e apenas em alguns nichos porque as bases de dados relacionais estão mais maduras e representam enormes investimentos [44].

Capítulo 3

Implementação

3.1 Introdução

No presente capítulo, desenvolve-se uma solução para o manuseamento de dados em modo desconectado de um sistema de processamento de encomendas. Nela estão definidas as etapas do processo de desenvolvimento, em particular, como os novos dados e as alterações aos dados existentes são sincronizados entre as bases de dados local e remota.

Para a concretização da aplicação, subjacente ao caso de estudo, foi necessário compreender o contexto e as exigências da própria aplicação. Para o efeito foi necessário realizar o levantamento dos requisitos bem como a sua análise foi seguida a abordagem XP (*eXtreme Programming*). Com esta metodologia, em cada etapa do processo de desenvolvimento são redefinidos os requisitos, os testes de aceitação, a integração continua e a melhoria do código que se acharem conveniente no ambiente organizacional. [49][50]. A recolha de requisitos é realizada na fase inicial do desenvolvimento, como especificações a serem implementadas. Constitui-se num processo de descobrir quais as ações que o sistema deve realizar e quais as restrições impostas sobre estas ações.

3.2 Especificação de requisitos da aplicação

O tratamento da informação é fundamental para que se possa encontrar uma solução aplicável. A seguir são apresentadas as especificações do sistema.

3.2.1 Requisitos funcionais

A aplicação deve responder aos seguintes requisitos funcionais:

Código	Requisitos funcionais	Casos de uso
RF01	A aplicação deve permitir registar Encomenda.	UC01
RF02	A aplicação deve permitir registar Cliente.	UC02
RF03	A aplicação deve permitir editar Cliente.	UC03
RF04	A aplicação deve permitir atualizar Encomenda.	UC04
RF05	A aplicação deve permitir consultar Cliente.	UC05
RF06	A aplicação deve permitir consultar Encomenda.	UC06
RF07	A aplicação deve permitir registar Stock	UC07
RF08	A aplicação deve permitir atualizar Stock.	UC08
RF09	A aplicação deve permitir registar Produtos.	UC09
RF10	A aplicação deve permitir atualizar Produtos.	UC10
RF11	A aplicação deve permitir registar Localização.	UC11
RF12	A aplicação deve permitir registar encomenda satisfeita.	UC12

Tabela 3.1: Requisitos funcionais

3.2.2 Requisitos Não Funcionais

Constituem todas as necessidades de desenvolvimento de sistemas que não podem ser atendidas através de funcionalidades. Estes englobam os custos de desenvolvimento e operacionais: usabilidade, precisão, desempenho, segurança, fiabilidade, portabilidade, robustez e restrições [51].

Código	Requisitos não funcionais	tipo
RNF01	A aplicação permitirá uma navegação óbvia, intuitiva, flexível de forma a fomentar maior desempenho.	Usabilidade
RNF02	Em caso de falhas de comunicação entre as máquinas local e remota, a aplicação deverá permitir aos utilizadores trabalharem na melhor comodidade.	Fiabilidade
RNF03	A avaliação de desempenho está sujeita ao número de clientes envolvidos na transferência e receção de recursos. Portanto, a aplicação está dependente da taxa de ocupação, do estado da rede, do tipo de máquina e da concorrência.	Desempenho
RNF04	A permissão de acesso ao conjunto de dados é aplicável a qualquer cliente com permissão de emissão de encomenda.	Segurança

Tabela 3.2: Requisitos Não funcionais

3.3 Análise de requisitos

Feito o levantamento de requisito é necessário realizar a análise e sua modelagem para que os mesmos reflitam inevitavelmente o ambiente organizacional e a política de persistência de recursos que legitime a tomada de decisão da gestão da organização, pois uma deficiente análise dos requisitos implica a um mau funcionamento.

3.3.1 Diagrama de Casos de Uso

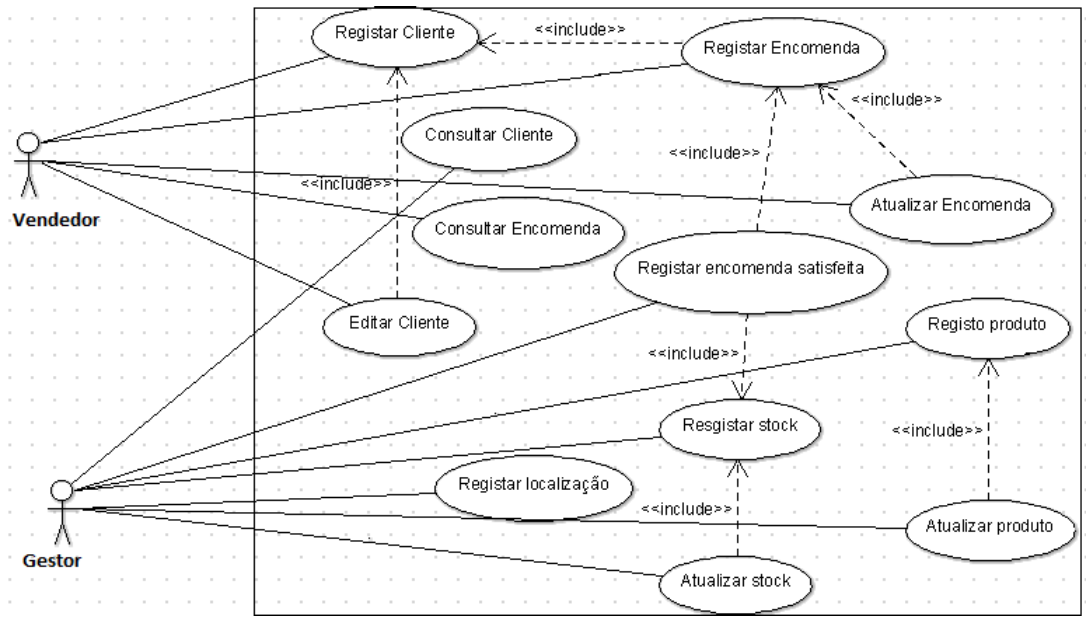

Figura 3.1: Diagrama de caso de uso

Na figura 3.1, estão identificados dois atores do sistema, o vendedor e o gestor. O gestor tem por função satisfazer as encomendas emitidas pelo vendedor, bem como registar os dados necessários para satisfação dos pedidos de clientes.

3.3.1.1 Detalhes de casos de uso

Nas tabelas que se seguem apresentam-se as especificações das ações que o ator vendedor e gestor realizam no sistema.

Caso de Uso para registar encomenda

UC01-Registar Encomenda.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção encomenda; 2. Pesquisar o último cliente (caso em que o cliente já tem registo); 3. Escolher a descrição do produto; 4. Atribuir a quantidade; 5. Pressionar a opção inserir; 6. Clicar em finalizar;
Cenário alternativo	1. Caso o cliente não exista: executar o UC02.

Tabela 3.3: Registo de encomenda.

UC02-Registar Cliente.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção de registo; 2. Escolher a opção encomenda; 3. Pressionar o botão registo cliente; 4. Preencher os campos; 5. Pressionar o botão inserir;

Tabela 3.4: Registo de Cliente.

UC03-Editar Cliente.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção de registo; 2. Escolher a opção encomenda; 3. Pressionar o botão registo cliente; 4. Pesquisar o cliente; 5. Editar o registo; 6. Pressionar no botão inserir.

Tabela 3.5: Editar Cliente.

UC04-Atualizar Encomenda.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção de registo; 2. Escolher a opção encomenda; 3. Inserir o código da encomenda; 4. Pressionar em procurar; 5. Realizar a alteração da encomenda; 6. Pressionar em finalizar.

Tabela 3.6: Atualizar Encomenda.

UC05-Consultar Cliente.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção de registo; 2. Escolher a opção encomenda; 3. Inserir o código do cliente; 4. Pressionar em consultar;

Tabela 3.7: Consultar cliente.

UC06-Consultar Encomenda.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção de registo; 2. Escolher a opção encomenda; 3. Inserir o código da encomenda; 4. Pressionar em procurar;
Cenário alternativo	1. Caso os dados estejam incorretos: executar o UC04.

Tabela 3.8: Consultar encomenda.

UC07-Registar Stock.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção de registo; 2. Escolher a opção stock; 3. Preencher os campos; 4. Pressionar em inserir.
Cenário alternativo	Caso os dados existam executar o UC08.

Tabela 3.9: Registar stock.

UC08-Atualizar Stock.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção de registo; 2. Escolher a opção stock; 3. Preencher os campos; 4. Pressionar em atualizar

Tabela 3.10: Atualizar stock.

UC09-Registar Produto.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção de registo; 2. Escolher a opção produto; 3. Preencher os campos; 4. Pressionar em inserir;
Cenário alternativo	Caso os dados existem executar o UC10.

Tabela 3.11: Registar produto.

UC010-Atualizar Produto.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção de registo; 2. Escolher a opção produto; 3. Escolher a descrição do produto; 4. Preencher os campos; 5. Pressionar em atualizar;

Tabela 3.12: Atualizar Produtos.

UC011-Registar Local.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção de registo; 2. Escolher a opção local; 3. Preencher os campos; 4. Pressionar em novo;

Tabela 3.13: Inserir localização.

UC012- Satisfazer Encomenda.	
Pré-condição	O utilizador deve abrir o sistema.
Pós condições	Sistema aberto.
Cenário principal	1. Escolher a opção de registo; 2. Escolher a opção Encomenda Não-safisfeita; 3. Selecionar o código da encomenda; 4. Pressionar na opção ok.

Tabela 3.14: Satisfazer Encomenda.

3.3.2 Diagrama de sequência

A descrição detalhada do funcionamento de um sistema constitui uma das etapas importantes no desenvolvimento de sistemas. Ao nível do desenvolvimento das aplicações são utilizados modelos que permitem comunicar as especificações do sistema com outras partes envolvidas no ambiente organizacional. Para ilustrar os detalhes da aplicação utilizou-se o diagrama de sequência da linguagem *(UML-Unified Modeling Language)*.

Um diagrama de sequência constitui a forma de descrever as interações que se estabelecem entre o utilizador e o sistema, sendo uma maneira de visualizar e validar os diferentes cenários de execução. O conjunto de etapas descritas nos diagramas de sequência das figuras 3.2, 3.3, 3.4, 3.5 e 3.6 permitem prever a forma como o sistema se comporta e descobrir as responsabilidades de cada artefacto ativado a partir de um evento, por uma ação realizada pelo utilizador.

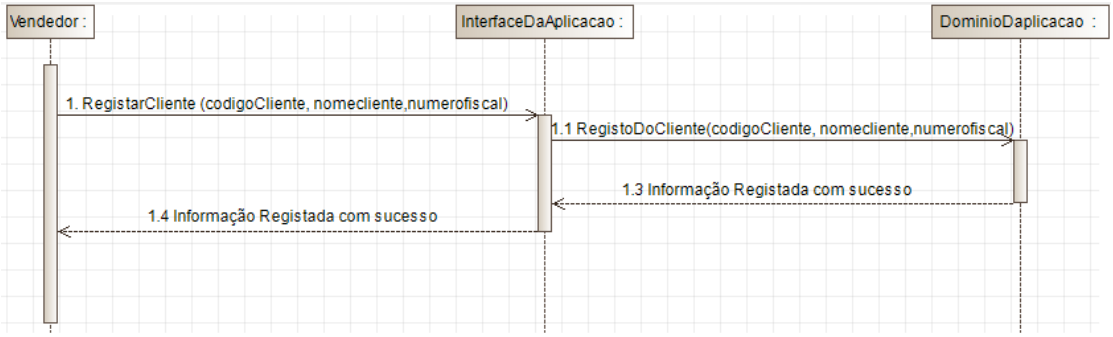

Figura 3.2: Diagrama de sequência para registo de cliente

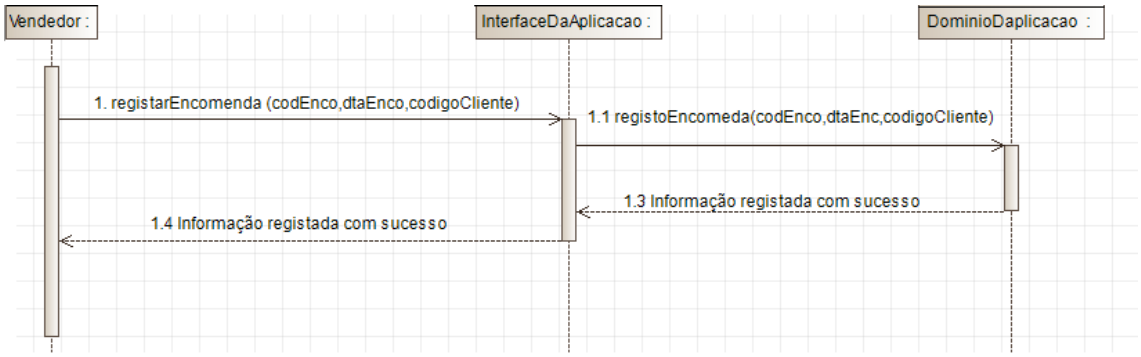

Figura 3.3: Diagrama de sequência para registar encomenda

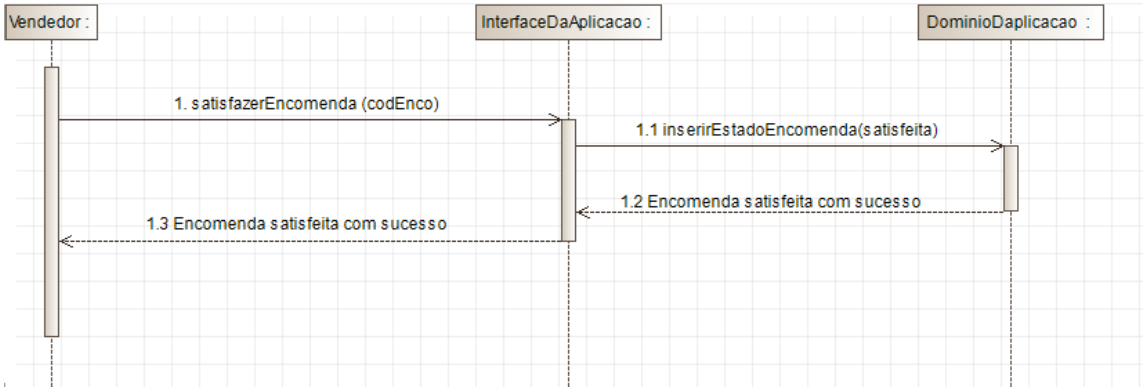

Figura 3.4: Diagrama de sequência para satisfazer encomenda

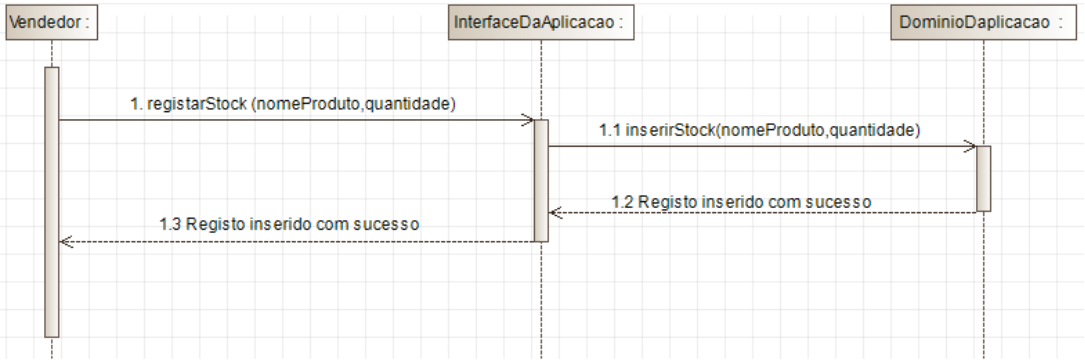

Figura 3.5: Diagrama de sequência para registar stock

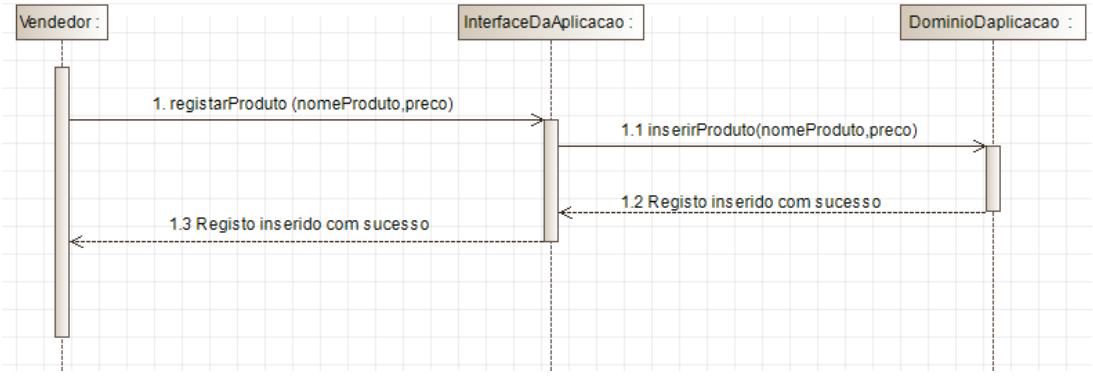

Figura 3.6: Diagrama de sequência para registar produto

3.3.3 Diagrama de classe

Após sucessivos refinamentos, é necessário elaborar o modelo estático que define as classes, atributos e relacionamentos que permanecem constantes no sistema - Figura 3.7.

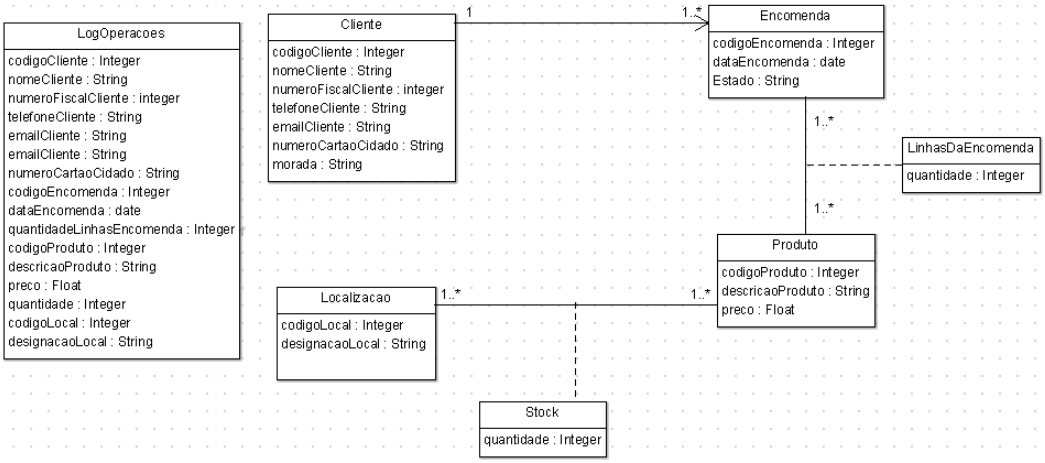

Figura 3.7: Diagrama de classe

3.3.4 Controlo e deteção de alterações

Assume-se que o número de alterações que nos dados local é pequeno em relação ao número de alterações que ocorrem nos dados no servidor. Na solução proposta foi criada uma tabela *LogOperacoes* (*log* de operações), tanto no cliente como no servidor, com os atributos *timestampOld* e *timestampNew* que determinam a marca do tempo das mudanças ocorridas. Como ao nível do processo de sincronização se descartou a utilização do servidor de tempos, que permite a sincronização dos relógios entre o cliente e o servidor remoto, o *timestamp* não constituiu um método adequado. Para o efeito utilizou-se o controlo de versão com recurso à função *NewId()*. Esta função gera o *hash* que representa a versão dos dados que são operados na máquina local e enviados ao servidor remoto. Como a máquina local continuará a enviar dados de tempo a tempo, o identificador torna-se obsoleto após os dados serem confirmados no servidor remoto.

3.3.5 Regras de negócio

1. A camada de lógica de negócio da máquina local solicita a sincronização à camada de negócio do servidor remoto. Este envia a mensagem com o endereço IP e a sua respetiva porta;

2. A camada de lógica de negócio do servidor remoto responde com base na solicitação feita;

3. A camada de lógica de negócio da máquina local gera um identificador, insere-o nas linhas a enviar, coleta todas as informações e as envia para a camada de lógica de negócio do servidor remoto;

4. A camada de lógica de negócio do servidor remoto recebe o conjunto de recursos da máquina local, os coloca na tabela *LogOperacoes* e os insere nas devidas relações;

5. A camada de lógica de negócio do cliente elimina o conjunto de recursos, depois de ter recebido a confirmação da receção dos mesmos por parte do servidor remoto.

3.3.6 Visão geral da solução

Para ilustrar o processo apresenta-se uma visão geral do processo de sincronização na figura 3.8. O processo de sincronização é sempre iniciado pelo cliente que o solicita ao servidor de sincronização designado por Servidor de *Sync* (*Sync Server*). Após o *Sync Server* responder ao pedido o processo prossegue. O cliente inicia o processo de envio de dados. A camada de lógica de negócio do *Sync Server*, através do método *inserirDados* (ver anexo A.4), insere os dados na tabela *LogOperacoes*. Para sinalizar a conclusão do processo de envio, ou seja, que já não há mais dados, são enviadas duas mensagens, uma a indicar o fim da operação e a outra que determina os dados que devem ser reconciliados.

Para efeitos de histórico, à medida que os dados são inseridos nas suas tabelas de destino, são também inseridos numa tabela de histórico. A ligação entre o *Sync Server* e o servidor de bases de dados remoto, onde são depositados os dados da organização, é estabelecida com o nível de isolamento *Serializable* (ver anexo A.1, A.2, A.3).

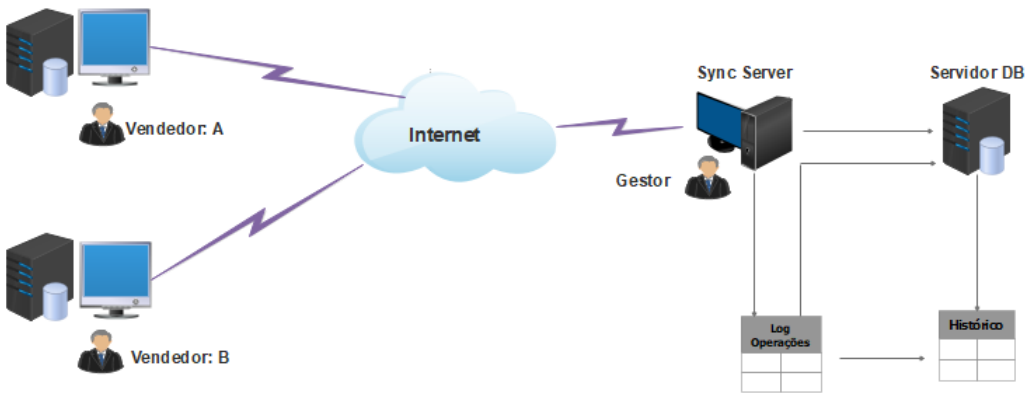

Figura 3.8: Modelo abstrato do processo de sincronização

3.3.7 Estabelecimento da comunicação entre o servidor e aplicação cliente

A comunicação entre a aplicação cliente e o *Sync Server* faz-se recorrendo ao uso de *sockets*. Um *socket* é o ponto terminal de um canal de comunicação entre dois processos através de uma rede. Em termos abstratos pode ser usado para por diversos computadores a comunicar entre si. Oferece suporte básico de comunicação e oculta detalhes de implementação. A utilização de *sockets* envolve processos que oferecem serviços e executam tarefas segundo solicitações feitas pelos clientes. Foram desenvolvidos na década de 80, do século XX, na universidade da Califórnia, Bekeley. Inicialmente eram conhecidos por *Berkley Sockets*, tendo sido implementados no sistema operativo 4.2 *UNIX*[52].

Quando o cliente pretende sincronizar, cria um *socket* associado a um porto. O *Sync Server* está à escuta num porto previamente acordado, e quando o pedido de sincronização ocorre, cria um novo *socket* dedicado ao resto da comunicação com o cliente

Etapas do processo de sincronização:

1. Estabelecimento da ligação com a fonte de dados;
2. Executar a consulta e repassa a resposta ou resultado;
3. Fechar a conexão com a fonte de dados.

3.4 Modelo de dados

Utiliza-se um modelo de dados para representar as entidades, relações e os atributos que participam no ambiente de negócio. O diagrama entidade-associação (DEA) é, frequentemente, usado na modelação de dados. A Figura 3.9 mostra o DEA do exemplo em estudo. Note-se que se optou por não desdobrar a associação vários-para-vários (*M:N*) referente à "LinhaEncomenda" para manter o diagrama relativamente compacto.

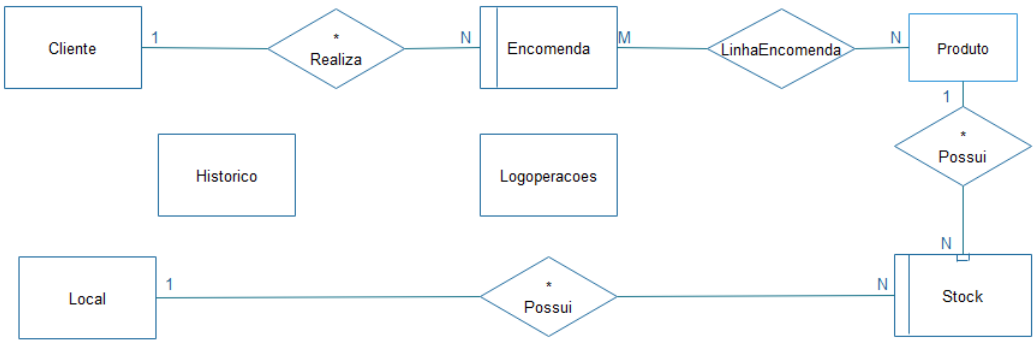

Figura 3.9: Diagrama Entidade-Associação do exemplo

De seguida apresenta-se o esquema relacional correspondente ao DEA do exemplo.

Cliente(codigoCliente, nomeCliente, numeroFiscalCliente, telefoneCliente, emailCliente, numeroCartaoCidadao, morada)
Encomenda(codigoEncomenda, dataEncomenda, codigoCliente, estado)
LinhasDaEncomenda(codigoEncomenda, codigoProduto, quantidadeLinhasDaEncomenda)
Produto(codigoProduto, descricaoProduto, preco)

Stock(codigoLocal, codigoProduto, quantidade)
Localizacao(codigoLocal ,designacaoLocal)

LogOperacoes(numeroRegisto, tipoEvento, codigoClienteOld ,nomeClienteOld, numeroFiscalClienteOld, telefoneClienteOld, emailClienteOld, numeroCartaoCidadaoOld, moradaOld , codigoEncomendaOld, dataEncomendaOld, quantidadeLinhasDaEncomendaOld, codigoProdutoOld, descricaoProdutoOld, precoOld, timeStampOld, codigoClienteNew, nomeClienteNew, numeroFiscalClienteNew, telefoneClienteNew, emailClienteNew, numeroCartaoCidadaoNew, moradaNew, codigoEncomendaNew, dataEncomendaNew, codigoProdutoNew, quantidadeLinhasDaEncomendaNew, descricaoProdutoNew, precoNew, timeStampNew, sincVersao)

Historico(numeroRegisto, tipoEvento, codigoClienteOld ,nomeClienteOld, numeroFiscalClienteOld, telefoneClienteOld, emailClienteOld, numeroCartaoCidadaoOld, moradaOld , codigoEncomendaOld, dataEncomendaOld, quantidadeLinhasDaEncomendaOld, codigoProdutoOld, descricaoProdutoOld, precoOld, timeStampOld, codigoClienteNew, nomeClienteNew, numeroFiscalClienteNew, telefoneClienteNew, emailClienteNew, numeroCartaoCidadaoNew, moradaNew, codigoEncomendaNew, dataEncomendaNew, codigoProdutoNew, quantidadeLinhasDaEncomendaNew, descricaoProdutoNew, precoNew, timeStampNew, sincVersao).

As relações identificadas no modelo de dados deram origem a tabelas SQL (o SGBD usado foi o (*Microsoft SQL Server*). A estrutura da base de dados do *SGBD* associado ao Servidor de *Sync* é a mesma do SGBD associado ao cliente. O significado dos atributos pode ser consultado no Dicionário de dados (Apêndice B).

Os identificadores locais só são válidos na máquina local não têm qualquer utilidade no *SGBD* remoto. Para reduzir os conflitos associados à emissão das encomendas no *SGBD* do cliente (máquina local) recorreu-se à função *NewID()* do *SGBD* para gerar os identificadores da encomenda. Para a identificação global dos registos foi empregue a função *Max()*. Esta função, com alguma manipulação para lidar com os erros, permitiu atribuir uma ordem sequencial a todas ocorrências que são inseridas no *SGBD* remoto.

3.5 Tecnologias e Ferramentas Utilizadas

Para o desenvolvimento da aplicação foi necessário recorrer a um conjunto de tecnologias e ferramentas:

3.5.1 NetBeans 8.2

As aplicações (cliente e *Sync Server*) foram desenvolvidas em java, com recurso ao *IDE NetBeans 8.2* que serve de suporte a todo o código da aplicação. A escolha cingiu-se pela proximidade com este *IDE* e também pela sua facilidade na integração de base de dados com *SQL Server*.

3.5.2 SQL Server Management Studio 2012

Por forma a configurar e administrar a base de dados utilizou-se o *SSMS 2012*, aplicação da *Microsoft*. Esta ferramenta possui editores de *scripts* e ferramentas gráficas que permitiram a criação e a interrogação da base de dados. Esta ferramenta é também muito importante na análise e na otimização do desempenho da base de dados (pois permite aceder ao plano de execução).

3.6 Garantia da persistência de recursos.

Como já foi referido, a solução proposta segue uma arquitetura cliente/servidor, onde tanto o cliente como o servidor (de *Sync*) acedem ao "seu" SGBD. A comunicação entre a máquina local(cliente) e o *Sync Server* é feita, para um endereço (IP) e uma porta, usando *sockets*.

Antes que a sincronização seja concretizada, a camada de lógica de negócio da aplicação cliente gera um identificador do conjunto de ocorrências que dispõem e o insere no campo *sincVersao*, identificando a versão dos dados a enviar ao servidor remoto.

As alterações efetuadas ao nível local são enviadas ao servidor e aí alojadas na tabela *LogOperacoes*. Portanto, a tabela *LogOperaçoes* no cliente serve para guardar as alterações feitas à replica local dos dados. No servidor, serve como memória tampão para os dados serem posteriormente encaminhados para as tabelas definitivas. As três operações típicas de alteração de dados, inserção, modificação e exclusão, requerem estratégias próprias para manter a consistência dos dados.

3.6.1 A operação de inserção

No momento em que o utilizador acionar o botão inserir é gerado um *NewId()* juntamente com outros dados do cliente em questão.

3.6.2 A operação de exclusão

O processo de eliminação de um registo passa pela obtenção do identificador da ocorrência a excluir. No caso da aplicação corrente são permitidas exclusões de ocorrências das tabelas *Encomenda, linhas das encomendas e LogOperacoes*. Algumas questões no que concerne à eliminação de dados da tabela *LogOperacoes*:

A remoção de dados da tabela local *LogOperacoes* induz à remoção dos registos correspondentes da tabela Linhas da Encomendas. O resultado desta operação não precisa de ser propagada para o servidor

3.6.3 Operação de atualização

Algumas considerações:

1. Caso a tabela *LogOperacoes* possua o identificador da linha a ser atualizada e a operação seja uma inserção, a linha original é atualizada e os seus valores são deslocados para os campos antigos. Este efeito de deslocamento dos recursos permite que, no momento que ocorre a sincronização o servidor remoto receba dados atualizados;

2. Caso a tabela *LogOperacoes* não possua o identificador do registo a ser inserido, os novos recursos são inseridos.

3.7 Transferência de dados

Os dispositivos móveis trouxeram novas oportunidades de negócio às empresas, mas trouxeram também novos desafios, nomeadamente, ao nível do acesso às bases de dados corporativas a partir de dispositivos móveis.

Conforme referido anteriormente, as aplicações foram desenvolvidas em linguagem java. O java é uma linguagem de programação orientada a objetos e multiplataforma. Programas desenvolvidos em java são compilados para *bytecode* executadas numa máquina virtual.

No desenvolvimento da solução, no que à transferência de dados diz respeito, procurou-se tirar o melhor proveito possível dos recursos usados, i.e., dos mecanismos disponibilizados pelo SGBD e pela ferramenta de desenvolvimento.

3.7.1 Transferência linha-a-linha

Uma primeira abordagem proposta para concretizar a transferência está esquematizada na Figura 3.10. Num primeiro passo, o cliente envia uma mensagem, via *socket*, ao *Sync Server*, informando-o que tem dados para enviar. O servidor reage criando um novo canal de comunicação e uma nova *thread* para lidar com as comunicações e receber os novos dados. De seguida autoriza o cliente a enviar os dados, informando-o do canal a ser usado para o efeito. Após receber autorização, o cliente inicia o processo de envio de dados, linha-a-linha. Esta passagem de dados é sincronizada com o *Sync Server*, sendo confirmada a boa receção de cada linha. À medida que recebe os dados, o servidor insere-os na tabela *LogOperacoes*. Finalizado o envio, o cliente informa o servidor e aguarda autorização para eliminar os seus dados locais (da "sua" tabela *LogOperacoes*).

Ao nível das operações sobre as bases de dados, tanto o cliente como o *Sync Server* usam transações explícitas. No entanto, o cliente comunica com o "seu" SGBD usando o nível de isolamento mais elevado, *Serializable*, enquanto o *Sync Server* usa o nível de isolamento *Read Uncommitted*.

Terminada a fase de sincronização, o *Sync Server* inicia o processo de reconciliação (usando transações explícitas e altera o nível de isolamento para *Read Committed*).

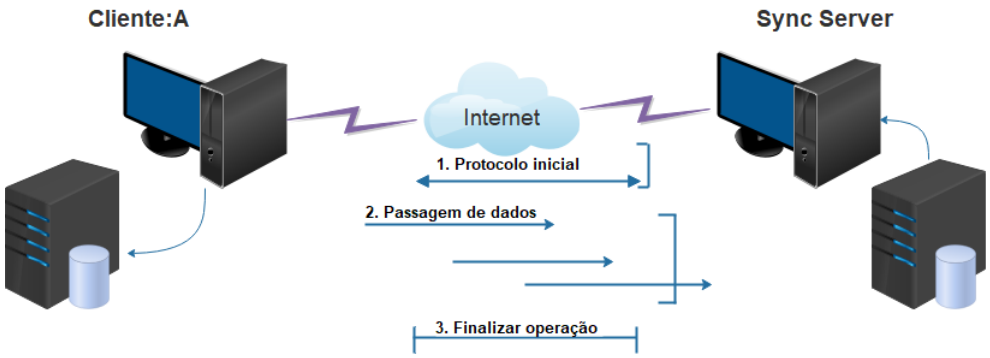

Figura 3.10: Transferência de dados, linha-a-linha, entre o cliente e o *Sync Server*

3.7.2 Transferência via *Linked Server*

A segunda abordagem proposta tira partido do facto de tanto o cliente como o *Sync Server* guardarem os seus dados em SGBD's. Neste caso, recorre-se à funcionalidade *Linked Server*, que é apropriada para estabelecer uma conexão segura entre um servidor *MS SQL Server* e outras fontes de dados (*MS Excel, MS Access*, ou outra instância do *MS SQL Server*, por exemplo).

Nesta abordagem, à semelhança da abordagem anterior, o cliente inicia o processo ligando-se ao *Sync Server* a informar que tem dados. O cliente recebe, então, os parâmetros necessários para configurar o *Linked Server* no "seu" SGBD local, juntamente com a informação acerca do SGBD remoto. A fase seguinte corresponde à passagem de dados SGBD a SGBD, sem intervenção do cliente. Terminada a operação, o cliente informa o *Sync Server* que a operação de sincronização acabou e que este pode avançar para a reconciliação. A Figura 3.11 ilustra este processo.

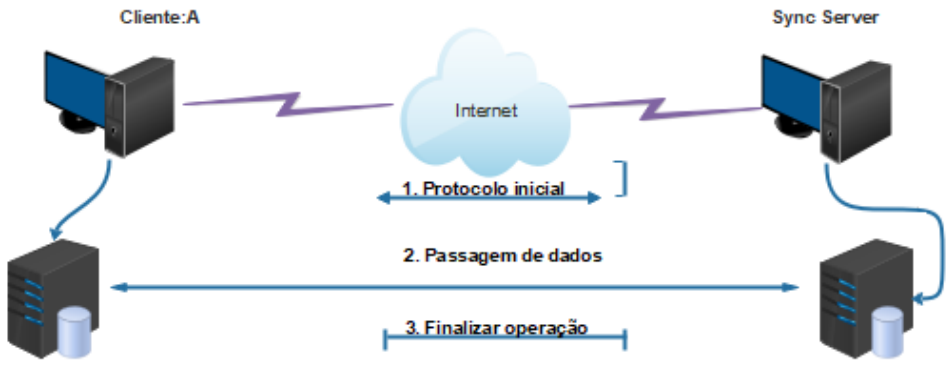

Figura 3.11: Transferência de dados por SGBD

A reconciliação é idêntica nas duas abordagens e faz-se de modo semiautomático, com recurso a procedimentos armazenados (e a *triggers*). As situações que não podem ser tratadas automaticamente, são transferidas para outra tabela para tratamento manual.

3.8 Sincronização

A sincronização constitui o processo de transferência das alterações do estado da base de dados do cliente para o servidor remoto, sendo o servidor o ponto de convergência de todos recursos.

Neste processo de transferência (e reconciliação) podem ocorrer dois tipos de conflitos: de integração [3] e de lógica[4].

A utilização de bases de dados em ambientes desconectados propicia a ocorrência de conflitos de integração, em que o identificador de um registo local colide com o identificador de outro registo na base de dados remota. Para contornar este problema foi adotado o seguinte método em (5) cinco fases:

1. Fase 1: Consiste na atribuição de um identificador de controlo de versão das operações ocorridas na máquina local. Este identificador permite distinguir os registos dos diferentes clientes (máquinas locais).
2. Fase 2: Consiste na seleção do conjunto de registos sinalizado com o controlo da versão das operações a serem enviadas;
3. Fase 3: Consiste em eliminar o conjunto de registos enviados ao servidor remoto;
4. Fase 4: Consiste na definição de critérios de validação para determinar, entre os registos recebidos, que registos são mantidos e que registos são descartados pelo servidor remoto.
5. Fase 5: Confirmar as transações.

3.8.1 *Linked servers* e bases de dados distribuídas

A gestão dos dados em ambiente de replicação é dividida entre o cliente e o servidor.

Existem três aspetos a considerar em relação às bases de dados replicadas:

1. Localização dispersa dos dados;
2. Fragmentação e duplicação;
3. Controlo de acessos.

3.8.2 Gestão de transparência

Reflete a capacidade de quão bem a base de dados é replicada, proporcionando aos utilizadores a visão de ser um sistema único. São consideradas duas formas de transparência. A transparência de localização em que a instrução usada para executar determinada tarefa é independente da locação dos recursos e do sistema no qual uma operação é executada, e

[3] Atualização assíncronas

[4] Quando os dados se contradizem; a<b; a=5 e b=3

a transparência de nomenclatura que atribui identificação de nomes a cada ocorrência na base de dados.

O SQL Server fornece transparência de replicação e de fragmentação.

1. Transparência de replicação: A existência de réplicas e a maneira de as distribuir. Está relacionada com a capacidade de ocultar existência de duas ou mais cópias.

2. Transparência de fragmentação: Consiste em dividir a base de dados em fragmentos menores, por forma a melhorar o desempenho, a disponibilidade e a confiabilidade. Portanto, cada réplica constitui um subconjunto da relação.

Uma transação distribuída constitui o conjunto de tarefas relacionadas que podem ser confirmadas ou anuladas como um todo. Ao nível das ferramentas de desenvolvimento podemos utilizar o *SQL Server* como servidor *vinculado* que possibilita a execução de transações distribuídas.

No SQL Server a responsabilidade de controlo e das transações distribuídas é do componente o *MSDTC (Microsoft Distributed Transaction Coordinator)*. É função do *MSDTC* coordenar todos os servidores vinculados que participam da transação.

3.8.3 Replicação com SQL Server

O SQL Server fornece três tipos de replicação para uso em aplicativos distribuídos:

1. Replicação *Snapshot*: Possui como função distribuir os recursos num momento específico sem monitorar as atualizações de dados. É adequado para o caso de se efetuarem poucas mudanças;

2. Replicação transacional: Responde a cada alteração efetuada ao recurso. Alteração ou inserção de um recurso é entregue ao cliente quando há alguma alteração no servidor. É apropriado para lidar com o problema de sincronização, pois, garante baixa latência entre a alterações feitas no servidor e o momento da sua receção no cliente;

3. Replicação *merge*: Permite o rastreamento de alterações do servidor e clientes. Quando um cliente se conecta à rede, ele será sincronizado com o cliente. A diferença entre a replicação de *merge* e transacional é que a replicação de *merge* não sincroniza cada alteração, o editor reflete apenas a alteração final.

3.8.4 Arquitetura

A capacidade que um sistema possui em apresentar os seus recursos, está diretamente ligada à sua arquitetura. A arquitetura consiste na separação dos elementos do sistema permitindo mudanças nos variados domínios da aplicação. A utilização da arquitetura está ligada à separação de responsabilidades dos componentes envolvidos no negócio organizacional e está dependente do papel a ser atribuído a cada componente.

A aplicação foi desenvolvida com base na arquitetura de três camadas, quer por parte do cliente quer por parte do servidor. Consideramos a camada de lógica de domínio do servidor, da persistência e de lógica de negócio do cliente que possui o papel de interagir com a interface do sistema. Por se tratar de uma aplicação que funciona em ambiente de desconexão, as três camadas encontram-se quer do lado do servidor assim como do lado do cliente.

Ao nível da persistência de recursos, estão definidas as regras dos sistemas de gestão de base de dados com *stored procedures, cursors e constraints*, tendo um impacto de dependência do sistema de gestão de base de dados, caso a aplicação venha ser usada com outro sistema de gestão de base de dados, pois que, as restrições impostas estão dependentes destes. Ao nível da interação é papel da interface fornecer uma interação do utilizador com aplicação por forma a receber e apresentar recursos. Nas camadas de servidores e cliente residem os processos de negócio que permitem o envio e a receção de recursos, que são mantidos quer do lado do servidor quer do lado do cliente. O servidor por oferecer um serviço público continuará a funcionar indefinidamente.

Capítulo 4

Resultados

Após a realização do levantamento das técnicas de sincronização e reconciliação, especificações, recolha de requisitos, modelagem e implementação, torna-se essencial apresentar os resultados, bem como analisá-los e articulá-los com a literatura existente.

Na análise efetuada, constatou-se que a utilização das distintas técnicas de sincronização e reconciliação de dados depende do tipo de aplicação a desenvolver, pois que, diferentes sistemas de gestão de base de dados, utilizam diferentes métodos de resolução de conflitos.

Verificou-se que cada sistema de gestão de base de dados, possui a sua forma de sincronização incorporada, que exige uma solução específica para a reconciliação de dados. Também, podemos verificar que as bases de dados NoSQL, possuem um desempenho, escalabilidade e disponibilidade que não são alcançados pelas bases de dados estruturadas, por estas utilizarem bloqueios, no entanto, são utilizadas em aplicações onde há a necessidade de garantir a consistência dos dados.

Após a implementação da aplicação, é necessário avaliar a solução. Os resultados foram avaliados com base na sincronização e reconciliação de dados em ambiente de operações online/offline, o que permitiu testar as diferentes estratégias de resolução de conflitos. Os resultados da avaliação da aplicação construída, com base na resolução de conflitos de integração descrito na secção "3.8 Sincronização", são apresentados nas Figuras 4.1 e 4.2.

Os testes foram realizados numa infraestrutura com 6 computadores: 1 computador a desempenhar a função de servidor (*Sync Server + SQL Server 2012*) e outros 5 a desempenhar a função de cliente (incluindo o SGBD local, *SQL Server 2012*). As máquinas possuem as seguintes especificações: um processador Intel® Core™ i5-3470 CPU @3.20GHz, com memória RAM de 8GB .

Os cinco computadores a desempenhar a função de cliente correspondem aos dispositivos desconetados, que vão recolhendo dados e que a dado momento precisam de os enviar para sincronização. Para efeitos de teste, supõe-se que estes dispositivos pretendem fazer a sincronização sensivelmente ao mesmo tempo (que é a situação que cria mais *stress* no sistema).

Os dados de teste foram gerados por um programa específico, desenvolvido para injetar um certo número de registos nas tabelas (*Cliente, Encomenda, LinhadaEncomenda, LogOperacoes*) da base de dados local (do cliente), considerando dados fictícios, aleatórios, mas verosímeis. Em cada cliente foram criadas cargas de 2000, 4000, 6000, 8000 e 10000 registos (encomendas + linhas de encomenda), perfazendo uma carga total de 10000, 20000,

30000, 40000 e 50000 registos, respetivamente.

Para aferir o tempo envolvido nas operações de sincronização e reconciliação, consideraram-se várias configurações tendo em conta a abordagem seguida (linha-a-linha ou *Linked Server*) e a carga total de registos. Cada configuração foi testada cerca de 20 vezes, para um total de 204 testes. As figuras 4.1 e 4.2 ilustram os resultados obtidos (tempos médios).

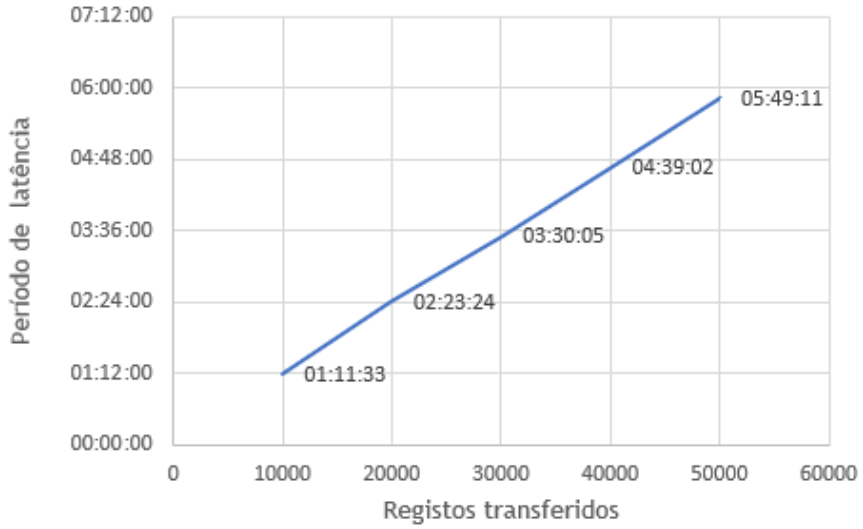

Figura 4.1: Transferência linha-a-linha

Figura 4.2: Transferência com base no *Linked Server*

Os resultados mostram que a sincronização com recurso aos mecanismos otimizados das bases de dados distribuídas apresenta vantagens relativamente à sincronização segundo a abordagem linha-a-linha. Por exemplo, a transferência de 10000 registos consumiu cerca de 4293 segundos usando a abordagem linha-a-linha e somente 26 segundos usando a abordagem *Linked Server*.

A abordagem linha-a-linha consome muito mais tempo porque cada linha transferida da tabela *LogOperacoes* local para a tabela *LogOperacoes* remota envolve várias operações,

nomeadamente, recuperação da linha a partir do SGBD local, empacotamento dos dados numa mensagem, envio sincronizado da mensagem para (a respetiva *thread* d') o *Sync Server*, aferição da boa receção da mensagem e, finalmente, desempacotamento dos dados e sua inserção na tabela *LogOperacoes* remota. Considerando, a existência de vários clientes a executar em simultâneo (cinco, neste caso), há ainda que lidar com as questões de concorrência no acesso à base de dados remota por parte das *threads* lançadas pelo *Sync Server*.

A abordagem baseada no *Linked Server* demora de facto muito menos tempo, mas tem como efeito lateral mais transações abortadas, devido ao aumento do tempo de espera das transações.

De acordo com os testes, verificou-se que é possível realizar consultas nos modos *offline* e *online*. Também é possível realizar consultas na base de dados remota em simultâneo com a sincronização e a reconciliação de dados.

Capítulo 5

Considerações finais

5.1 Conclusões

Neste trabalho foi abordada a sincronização e a reconciliação de dados nos modos *offline* e *online*, o que possibilitou a exploração de diferentes técnicas de sincronização e reconciliação de dados. Foi, também, realizado o levantamento das diferentes ferramentas existentes no âmbito da sincronização e reconciliação de dados e criada uma aplicação exemplo para lidar com o processo de sincronização e reconciliação de dados em ambiente de desconexão, através do processamento de grandes quantidades de dados transferidos da base de dados local para o servidor remoto. Estes resultados permitem concluir que foram alcançados os objetivos do trabalho.

O trabalho, focou-se no desenvolvimento de uma solução para sincronização de grandes quantidades de recursos e a sua persistência, em ambiente de pouca largura de banda de rede que permita que os utilizadores sejam capazes de executar tarefas, como a inserção e a modificação, sem a necessidade de uma comunicação continuada, fazendo o uso da base de dados local, permitindo, desta forma, aumentar a disponibilidade.

Com base neste princípio foram desenvolvidas duas aplicações: A primeira cingiu-se na utilização do sistema de gestão de base de dados sem a utilização dos protocolos de transações distribuídas, utilizadas pelos sistemas de gestão de base dados, servindo como meio de persistência dos recursos e a segunda utilizamos os mecanismos oferecidos pelo motor de base de dados usando a componente das bases de dados distribuídas, *linked server*.

Por forma a garantir o envio de recursos de versões, adotaram-se os níveis de isolamento *serializable e read committed* na utilização da tabela Log de operações *(LogOperacoes)*. A utilização do nível de *serializable* permitiu garantir um isolamento adequado, evitando que transações incompletas pudessem ser transferidas no momento da efetivação da sincronização com o servidor remoto. Por outro lado, sendo o servidor remoto quem mantêm as operações de todas as máquinas envolvidas, foi necessário a utilização de identificação exclusiva do conjunto de registos por forma a garantir a confirmação do conjunto de dados recebido do cliente. A confirmação dos dados no servidor baseado no nível de isolamento *read commited*.

Na solução proposta os clientes funcionam com a atribuição dos identificadores locais. O servidor remoto, aquando da integração dos dados descarta este indentificador atribuindo, então, um identificador global, por forma a garantir a consistência global dos dados.

No desenvolvimento da aplicação foram encontradas algumas dificuldades como, por exem-

plo, na utilização do *trigger* que permitia a atribuição da versão de dados a ser enviada ao servidor. Pois que, no momento em que o cliente retornava o conjunto de dados, o *trigger*, bloqueava os registos por algum tempo, o que ocasionava o impasse.

No contexto das aplicações desconetadas e consoante os recursos do dispositivo móvel, o programador pode optar pela sincronização baseada em *Linked Server*, se o dispositivo o suportar, ou por transferência linha-a-linha, no caso de um dispositivo mais modesto. Em ambos os casos, a reconciliação é processada ao nível do *Sync Server* (que se supõe executar numa máquina competente).

5.2 Trabalhos futuros

No futuro, pretende-se refinar a aplicação desenvolvida com o intuito de obter uma versão estável, mas mais leve, utilizando bases de dados desktop, ou mesmo ficheiros de texto, ao nível do dispositivo móvel.

Deseja-se também aprofundar o conceito estudado nesta dissertação e aplicá-lo na construção de uma solução orientada para instituições hospitalares de Angola, com o objetivo de efetuar o registo dos utentes dos serviços de saúde, disponibilizando estes registos sempre que necessário, e a pessoal autorizado, em qualquer lugar e a qualquer hora. Deste modo, as instituições de saúde, e o governo, teriam uma tarefa menos árdua na obtenção de dados credíveis, possibilitando, assim, avaliar os indicadores de saúde de forma fiável.

Bibliografia

[1] A. F. Nejad, S. Kharazmi, S. Bayati, S. K. Golmohammadi, and H. Abolhassani, "Semantic log based replication model for optimizing heterogeneous DBMS interaction," *Proceedings - 2009 1st International Conference on Advances in Databases, Knowledge, and Data Applications, DBKDA 2009*, no. 2, pp. 138–142, 2009. xv, 10, 12

[2] M. S. Hossain and M. A. Hossain, "A Sysnoptic Reviewof Reconciliation Algorithm for Mobile Databases," *Information Technology Journal*, 2006. [Online]. Available: http://scialert.net/qredirect.php?doi=itj.2006.83.87{&}linkid=pdf 1

[3] B. S. Ramya, K. Shirin Bhanu, and M. Seetha, "A Stateful Database Synchronization Approach for Mobile Devices," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 2, no. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.650.7220&rep=rep1&type=pdf, 2012. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.650.7220{&}rep=rep1{&}type=pdf 1

[4] J. L. Feng, X. Q. Qiao, and Y. Li, "The research of synchronization and consistency of data in mobile environment," *Proceedings - 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, IEEE CCIS 2012*, vol. 2, pp. 869–874, 2013. 1

[5] N. Malhotra and C. Anjali, "Implementation of Database Synchronization Technique between Client and Server," *International Journal of Engineering Science and Innovative Technology (IJESIT)*, vol. 3, no. 4, pp. 460–465, 2014. [Online]. Available: http://www.ijesit.com/Volume3/Issue4/IJESIT201404{_}63.pdf 1, 2

[6] R. Ahuja, R. Bagrodia, L. Bajaj, and M. Takai, "EVALUATION OF OPTIMISTIC FILE REPLICATION IN WIRELESS MULTIHOP NETWORKS," *,https://pdfs.semanticscholar.org/eb2c/84d167d62c8619b0799b004516fa193548b3.pdf. Acessado Acessado aos 11 de setembro de 2017.* [Online]. Available: https://pdfs.semanticscholar.org/eb2c/84d167d62c8619b0799b004516fa193548b3.pdf 1

[7] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," *ACM SIGMOD Record*, vol. 25, no. 2, pp. 173–182, 1996. [Online]. Available: http://portal.acm.org/citation.cfm?doid=235968.233330 2, 11

[8] B. Kitchenham, R. Pretorius, D. Budgen, O. P. Brereton, M. Turner, M. Niazi, and S. Linkman, "Systematic literature reviews in software engineering-A tertiary study," *Information and Software Technology*, vol. 52, no. 8, pp. 792–805, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2010.03.006 5

[9] J. Arkady, Pikovsky;Rosenblum, Michael;Kurths, *Synchronization*, 12th ed., H. Chirikov, Boris ;Moss, Frank ; Cvitanovic´, Predrag;Swinney, Ed. Potsdam: 2001. [Online]. Available: www.cambridge.org/9780521592857 5

[10] C. Pierce, "is a File Synchronizer?" 5

[11] A. A. Imam, S. Basri, and R. Ahmad, "Data synchronization between mobile devices and server-side databases: A systematic literature review," *Journal of Theoretical and Applied Information Technology*, vol. 81, no. 2, pp. 364–382, 2015. 6

[12] A. Demers and K. Petersen, "The Bayou architecture: Support for data sharing among mobile users," *Mobile Computing ...*, no. January 2014, pp. 2–7, 1994. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=512726 6

[13] S. H. Phatak and B. Nath, "Transaction-Centric Reconciliation in Disconnected Client – Server," pp. 459–471, 2004. 6

[14] A. E. Saddik, "A Framework for Two Phase Reconciliation in Mobile Databases Table of Contents," vol. 5, no. 613, 2005. 7

[15] P. S. Wankhade, M. S. Thorat, M. S. Jadhav, M. S. Thombare, and M. K. Bhangire, "Online-Offline Data Synchronization for End User," no. 6, pp. 33–35, 2014. 7

[16] I. Shabani, B. Çiço, and A. Dika, "Solving Problems in Software Applications through Data Synchronization in Case of Absence of the Network," *International Journal of Computer Science Issues*, vol. 9, no. 1, pp. 10–16, 2012. [Online]. Available: https://www.ijcsi.org/papers/IJCSI-9-1-3-10-16.pdf 7

[17] M.-Y. C. M.-Y. Choi, E.-A. C. E.-A. Cho, D.-H. P. D.-H. Park, C.-J. M. C.-J. Moon, and D.-K. B. D.-K. Baik, "A database synchronization algorithm for mobile devices," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 392–398, 2010. 7

[18] S.-w. Kim, "Synchronization in an Embedded DBMS Environment," *Journal of Computer Science*, vol. 6, no. 7, pp. 30–35, 2006. [Online]. Available: http://digilib.unsri.ac.id/download/200607A0615082009.pdf 7

[19] J. Sedivy, T. Barina, I. Morozan, and A. Sandu, "MCSync - Distributed, Decentralized Database for Mobile Devices," *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pp. 1–6, 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6354613 8

[20] B. Kemme and G. Alonso, "Database replication," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 5–12, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?doid=1920841.1920847 8, 10

[21] F. Nilsson and P. Olsson, "A survey on reliable communication and replication techniques for distributed databases." 8

[22] T. W. Page, R. G. Guy, J. S. Heidemann, D. H. Ratner, P. L. Reiher, A. Goel, G. H. Kuenning, and G. J. Popek, "Perspectives on optimistically replicated, peer-to-peer filing," *Software - Practice and Experience*, vol. 28, no. 2, pp. 155–180, 1998. 8

[23] T. Singh, P. S. Sandhu, and H. S. Bhatti, "Replication of Data in Database Systems for Backup and Failover – An Overview," vol. 2, no. 4, pp. 10–13, 2013. 8, 9, 15

[24] Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri, and A. Silberschatz, "Update propagation protocols for replicated databates," *ACM SIGMOD Record*, vol. 28, no. July 2015, pp. 97–108, 1999. 8

[25] Z. Sann and T. T. Soe, "Agricultural Loan System Using Data Replication Method," vol. 2, no. 5, pp. 725–732, 2017. 8, 9, 23

[26] A. Sousa, J. Pereira, L. Soares, A. Correia, L. Rocha, R. Oliveira, and F. Moura, "Testing the dependability and performance of group communication based database replication protocols," *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 792–801, 2005. 9

[27] I. Priyadarshini, P. S. S. Sane, and R. Jadhav, "Deadlock Detection in Distributed Database," vol. 2, no. 2, pp. 95–98, 2013. 9

[28] T. M. Connolly and C. E. Begg, *Database Systems*, 6th ed., 2015. 10, 12, 17, 18, 19, 20

[29] R. Ramakrishnan and J. Gehrke, "Database management systems," p. 1065, 2003. 10, 18

[30] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems*, 2011, vol. 12. [Online]. Available: http://www.springerlink.com/index/10.1007/978-1-4419-8834-8 12, 23

[31] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970. 13

[32] Y. Lee, Y. Kim, and H. Choi, "Conflict resolution of data synchronization in mobile environment," in *International Conference on Computational Science and its Applications*. Springer, 2004, pp. 196–205. 14

[33] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Computing Surveys*, vol. 15, no. 4, pp. 287–317, 1983. [Online]. Available: http://portal.acm.org/citation.cfm?doid=289.291 14

[34] C. R. Wallace and N. Soparkar, "Spheres of control: an approach to advanced recovery," pp. 1–16, 1996. 14

[35] G. Weikum and G. Vossen, *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*. Elsevier, 2001. 14

[36] C. Rob, Peter and Coronel, *Sistemas de Banco de Dados:Projecto, implemetação e gerencimaneto*, 2014, vol. 8ed, no. 9. 15, 17

[37] C. Coronel, S. S. A. Morris, and P. Rob, *Database systems : design, implementation, and management*, 2011. 18, 19, 20

[38] M. TechNet, "Modos de bloqueio," 2018. [Online]. Available: https://technet.microsoft.com/pt-br/library/ms175519(v=sql.105).aspx 18

[39] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ANSI SQL isolation levels," *ACM SIGMOD Record*, vol. 24, no. 2, pp. 1–10, 1995. [Online]. Available: http://portal.acm.org/citation.cfm?doid=568271.223785 20, 21, 22

[40] P. M. Studer, "ANSI SQL Isolation Levels," Ph.D. dissertation, 2014. [Online]. Available: https://wiki.hsr.ch/Datenbanken/files/Paper{_}ANSI{_}SQL{_}Isolation{_}Levels{_}Stefan{_}Luetolf{_}V2{_}1.pdf 21, 22

[41] G. Milener and C. Guyer, "Understanding Isolation Levels | Microsoft Docs," 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/connect/jdbc/understanding-isolation-levels 21

[42] F. M. Darmstadt, K. R. Stuttgart, F. B. S. Cornell, B. Welch, and M. View, "Mobile Software-Agents," pp. 1–24, 1997. 24

[43] N. Freire, "Entity Recognition and Resolution in Semi-structured Data," *Ieee-Tcdl.Org*, no. June, 2011. [Online]. Available: http://www.ieee-tcdl.org/mediawiki/TCDL/Bulletin/current/papers/freire.pdf 24

[44] W. Nosql and N. Leavitt, "to Their Promise ?" pp. 12–14, 2010. 24, 26

[45] P. Kirti, "Database for Unstructured , Semistructured data- NoSQL Focus on Availability," vol. 4, no. 2, pp. 466–469, 2015. 24, 26

[46] G. H. Hygerth and H. Michael, "Efficient Web Scraping and Database Synchronization," pp. 0–3. 24

[47] S. Eken, F. Kaya, Z. Ilhan, A. Sayar, A. Kavak, U. Kocasarac, and S. Sahin, "Analyzing distributed file synchronization techniques for educational data," *2013 International Conference on Electronics, Computer and Computation, ICECCO 2013*, pp. 318–321, 2013. 25

[48] A. A. Imam, S. Basri, and R. Ahmad, "Data Extraction Formulation for Efficient Data Synchronization between Mobile Databases and Server-side Database," *American Scientific Publishers*, vol. 4, no. 1, pp. 521–531, 2016. 25

[49] D. Karlström, "Introducing Extreme Programming – An Experience Report," *Third International Conference on eXtreme Programming and Agile Processes in*

Software Engineering, pp. 24–29, 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.613{&}rep=rep1{&}type=pdf{%}0Ahttp://www.frankwestphal.de/xp2002/DanielKarlstroem.pdf 27

[50] D. McKinney and J. Froeseth, "Agile CS1 Labs: eXtreme Programming Practices in an Introductory Programming Course," *Extreme Programming …*, pp. 164—-174, 2004. [Online]. Available: http://tocs.ulb.tu-darmstadt.de/8733295.pdf{%}5Cnhttp://link.springer.com/chapter/10.1007/978-3-540-27777-4{_}17 27

[51] R. Wazlawick, *Análise e Projeto de Sistemas de Informação Orientados.* Elsevier Brasil, 2010, vol. 2. 28

[52] H. K. Srivastava, R. Sinha, and S. Gupta, "Implementation of Socket Programming and RMI Using Simulating Environment," *International Journal of Scientific & Engineering Research*, vol. 4, no. 5, pp. 691–700, 2013. 35

Apêndice A

Anexos

A.1 Método de obtenção de registo do cliente

```
public boolean GetCursor(){
        String SQLString = "SELECT tipoEvento,codigoClienteOld,nomeClienteOld,numeroFiscalClienteOld,telefoneClienteOld,emailClienteOld,
                           numeroCartaoCidadaoOld,moradaOld,codigoEncomendaOld,dataEncomendaOld,quantidadeLinhasDaEncomendaOld, codigoProdutoOld,
                           descricaoProdutoOld,precoOld,timeStampOld,codigoClienteNew,nomeClienteNew,numeroFiscalClienteNew,telefoneClienteNew,
                           emailClienteNew,numeroCartaoCidadaoNew,moradaNew,codigoEncomendaNew,dataEncomendaNew,codigoProdutoNew,
                           quantidadeLinhasDaEncomendaNew,descricaoProdutoNew,precoNew,timeStampNew,sincVersao from LogOperacoes";
        rs = null;
        stmt = null;


        try {
           Conectar.closseConnection(con);
           con = Conectar.getConnection();
           con.setTransactionIsolation(java.sql.Connection.TRANSACTION_SERIALIZABLE);
           con.setAutoCommit(false);

           stmt = con.prepareStatement(SQLString);

            //obtém lista de registos
           rs = stmt.executeQuery();
           return true;
        } catch (SQLException ex) {
           return false;
        }
    }
```

Figura A.1: Método de obtenção registos

A.2 Método para enviar recuros ao servidor remoto

```
public String GetNextLogRec(){


        try {
           if (rs.next()){
            String c = (" insert into LogOperacoes(tipoEvento,codigoClienteOld,nomeClienteOld,numeroFiscalClienteOld,telefoneClienteOld,
                        emailClienteOld,numeroCartaoCidadaoOld,moradaOld,codigoEncomendaOld,dataEncomendaOld,quantidadeLinhasDaEncomendaOld,
                        codigoProdutoOld,descricaoProdutoOld,precoOld,timeStampOld,codigoClienteNew,nomeClienteNew,numeroFiscalClienteNew,
                        telefoneClienteNew,emailClienteNew,numeroCartaoCidadaoNew,moradaNew,codigoEncomendaNew,dataEncomendaNew,codigoProdutoNew,
                        quantidadeLinhasDaEncomendaNew,descricaoProdutoNew,precoNew,timeStampNew,sincVersao)
                        values(" + rs.getString(1) + "," + rs.getInt(2) + ",'" + rs.getString(3) + "'," + rs.getInt(4) + ",'" + rs.getString(5) +
                        "','" + rs.getString(6) +"','" + rs.getString(7) + "','" + rs.getString(8) + "'," + rs.getInt(9) + ",'" + rs.getTimestamp(10) +
                        "'," + rs.getInt(11) + "," + rs.getInt(12) + ",'" +rs.getString(13) + "'," + rs.getDouble(14) + ",'" + rs.getTimestamp(15) + "'," +
                        rs.getInt(16) + ",'" + rs.getString(17) + "','" + rs.getString(18) + "','" + rs.getString(19) + "','" + rs.getString(20) + "','" + rs.getString(21) + "','" + rs.getString(22) + "'," +
                        rs.getInt(23) + ",'" + rs.getTimestamp(24) + "'," + rs.getInt(25) + "," + rs.getDouble(26) + ",'" + rs.getString(27) + "'," +
                        rs.getDouble(28) + ",'" + rs.getTimestamp(29) + "','" + rs.getString(30) + "')");
            return c;
           }
           else{

            return "";
           }
        } catch (SQLException ex) {
           Conectar.closseConnection(con, stmt, rs);
           return "";
        }
    }
```

Figura A.2: Método para enviar recursos ao servidor remoto

A.3 Método para apagar registo na máquina local

```
    public void ApagaRegistos() {
        try {
            stmt = con.prepareStatement("DELETE FROM LOGOPERACOES");
            stmt.executeUpdate();

            con.commit();
            con.setAutoCommit(true);
            con.setTransactionIsolation(0);
        } catch (SQLException ex) {
            System.out.println("Erro ao apagar registos de Logoperacoes " + ex.getMessage());

        } finally {

            Conectar.closseConnection(con, stmt);

        }

    }
```

Figura A.3: Método para inserir dados no servidor

A.4 Método de receção de dados servidor remoto

```
1   public void InserirDaddos(String registo) {
2           LogOperacoes inserirDados = new LogOperacoes();
3
4
5           try {
6               Conectar.closseConnection(con);
7               con = Conectar.getConnection();
8               con.setAutoCommit(false);
9               stmt = con.prepareStatement(registo);
10              stmt.execute();
11              con.commit();
12
13          } catch (Exception ex) {
14              System.out.println("Erro ao inserir dados na tabela log operações" + ex.getMessage());
15
16          } finally {
17
18              Conectar.closseConnection(con, stmt);
19
20          }
21
22      }
```

Figura A.4: Método para inserir dados no servidor

A.5 Painel principal da aplicação

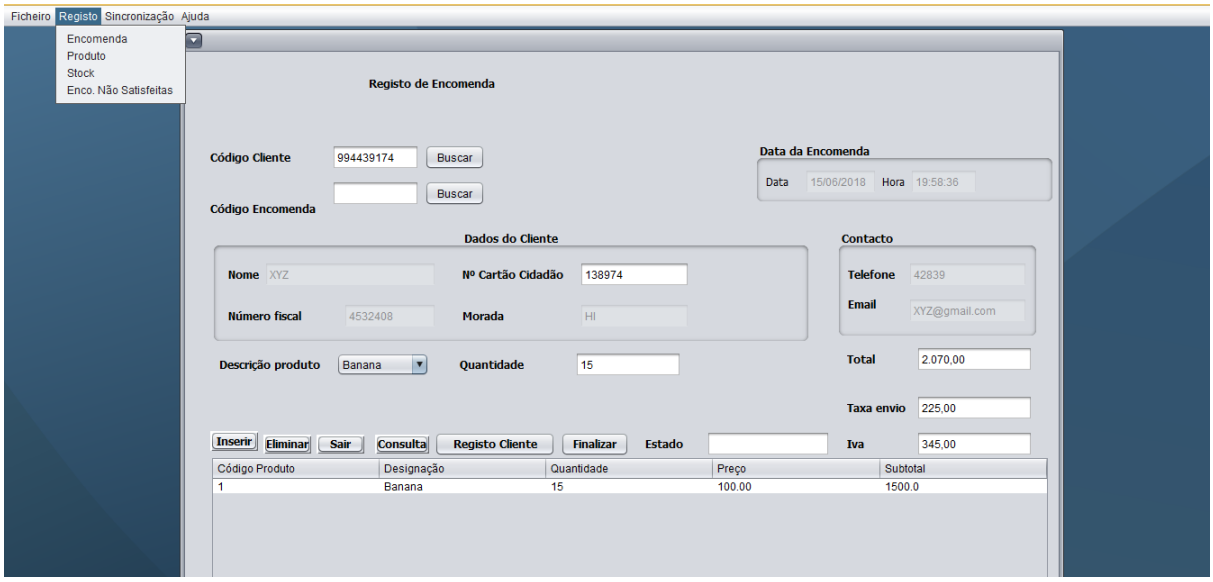

Figura A.5: Painel principal da aplicação

Apêndice B

Apêndice

B.1 Dicionário de dados

Entidade	Nome do Atributo	Tipo de dados	Descrição	Chave primaria ou estrangeira	Formato
Cliente	codigoCliente	int	código do cliente	PK	999999
	emailCliente	nvarchar(50)	e-mail do cliente		xxxxxxx
	morada	nvarchar(50)	morada do cliente		xxxxxxx
	nomeCliente	nvarchar(50)	nome do cliente		xxxxxxx
	numeroCartaoCidadao	nvarchar(15)	número do cidadão do cliente		xxxxxxx
	numeroFiscalCliente	int	número fiscal do cliente		999999999
	telefoneCliente	nvarchar(15)	número de telefone		(351) 923 879 945
Encomenda	codigoEncomenda	int	código da encomenda	PK	999999
	dataEncomenda	datetime	data de emissão de encomenda		dd-mm-aaaa
	codigoCliente	int	código do cliente	FK	999999
	estado	nvarchar(50)	estado da encomenda		xxxxxxx
LinhasDaEncomenda	codigoEncomenda	int	código da encomenda	FK	999999
	codigoProduto	int	código do produto		999999
	quantidadeLinhasDaEncomenda	decimal	quantidade de produto encomendados		9.999.999,99
Produto	codigoProduto	int	código do produto	pk	999999
	descricaoProduto	nvarchar(50)	descrição do produto		xxxxxxx
	preco	decimal	preço do produto		9.999.999,99
Stock	codigoLocal	int	código do local do stock	FK	999999
	codigoProduto	int	código do produto	FK	999999
	quantidade	int	quantidade do produto em stock		9.999.999,99
Localizacao	codigoLocal	int	código do local do stock	PK	999999
	designacaoLocal	nvarchar(50)	cidade do local de stock		xxxxxxx
LogOperacoes	codigoClienteNew	int	referência o código do cliente		999999
	codigoClienteOld	int	referência o código do cliente		999999
	codigoEncomendaNew	int	Referencia o código da encomenda		999999
	codigoEncomendaOld	int	referência o código da encomenda		999999
	codigoProdutoNew	int	referência o código do produto		999999
	codigoProdutoOld	int	referência o código do produto		999999
	dataEncomendaNew	datetime	referência a data da encomenda		dd-mm-aaaa
	dataEncomendaOld	datetime	referência a data da encomenda		dd-mm-aaaa
	descricaoProdutoNew	nvarchar(50)	referência o nome do produto		xxxxxxx
	descricaoProdutoOld	nvarchar(50)	referência o nome do produto		xxxxxxx
	emailClienteNew	nvarchar(50)	referência o email do cliente		xxxxxxx
	emailClienteOld	nvarchar(50)	referência o email do cliente		xxxxxxx
	moradaNew	nvarchar(50)	referência a morada cliente		xxxxxxx
	moradaOld	nvarchar(50)	referência a morada cliente		xxxxxxx
	nomeClienteNew	nvarchar(50)	referência o nome do cliente		xxxxxxx
	nomeClienteOld	nvarchar(50)	referência o nome do cliente		xxxxxxx
	numeroCartaoCidadaoNew	nvarchar(50)	referência o numero do cartão do cidadão		xxxxxxx
	numeroCartaoCidadaoOld	nvarchar(50)	referência o numero do cartão do cidadão		xxxxxxx
	numeroFiscalClienteNew	int	referência o numero fiscal do cidadão		999999999
	numeroFiscalClienteOld	int	referência o numero fiscal do cidadão		999999999
	numeroRegisto	int	número de registos inseridos		9999999999
	precoNew	decimal	referência o preço do produto		9.999.999,99
	precoOld	decimal	referênciao preço do produto		9.999.999,99
	quantidadeLinhasDaEncomendaNew	decimal	referência a quantidade da encomenda		9.999.999,99
	quantidadeLinhasDaEncomendaOld	decimal	referência a quantidade da encomenda		9.999.999,99
	sincVersao	nvarchar(36)	valor de hashi da versão dos dados		xxxxxxx
	telefoneClienteNew	nvarchar(15)	referência o numero de telefone do cliente		(351) 923 879 945
	telefoneClienteOld	nvarchar(15)	referência o numero de telefone do cliente		(351) 923 879 945
	tipoEvento	nvarchar(1)	tipo de operação a ser inserida exemplo insert		X
	timeStampNew	datetime	selo temporal da inserção ou atualização		2018-06-15 21:55:26.787
	timeStampOld	datetime	selo temporal da inserção ou atualização		2018-06-15 21:55:26.787

Figura B.1: Tabela representando o dicionário de dadosa