FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Experimentální server na podporu výzkumu v oblasti EOBI dat

Martin Soukup

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Experimentální server na podporu výzkumu v oblasti EOBI dat

Martin Soukup

Vedoucí práce
Ing. Kamil Ekštein, Ph.D.

PLZEŇ 2024

Citace v seznamu literatury:
SOUKUP, Martin. *Experimentální server na podporu výzkumu v oblasti EOBI dat*. Plzeň, 2024. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ing. Kamil Ekštein, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin SOUKUP**
Osobní číslo:	**A20B0558P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační systémy**
Téma práce:	**Experimentální server na podporu výzkumu v oblasti EOBI dat**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s formátem elektronické knihy objednávek EOBI, používaným burzami a dalšími finančními institucemi. Zorientujte se v aktuálních technikách efektivní implementace problémově orientovaných serverů.
2. Navrhněte a naprogramujte pomocí jednoduchých, stabilních a běžně používaných technologií server, který bude na základě přicházejících požadavků replikovat elektronickou knihu objednávek (EOB, Electronic Order Book) finančního trhu. Server bude mít interface pro zasílání požadavků na změny v knize objednávek (buď REST API nebo sockety nebo nějaký podobný, jednoduchý mechanismus) a webový interface pro vizualizaci stavu knihy objednávek, statistiky nad touto knihou, apod.
3. Důkladně otestujte stabilitu vyvinutého serveru a zejména korektní rekonstrukci knihy objednávek.
4. Vše popište v průvodním dokumentu, především specifika návrhu implementace tak, aby mohl případně v díle pokračovat další student.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Kamil Ekštein, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **2. října 2023**
Termín odevzdání bakalářské práce: **2. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2023

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Plzni dne 30. dubna 2024

. . . . . .
Martin Soukup

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Cílem této bakalářské práce je navrhnout a implementovat systém zpracování burzovních dat. Práce se bude nejprve zabývat detailním zkoumáním fungovaní systému elektronické knihy objednávek německé nadnárodní společnosti Deutsche Börse Group, a burzovních dat, které tento systém zprostředkovává. Na základě této analýzy by měly být navrženy algoritmy, které data zpracují a uloží do formátu vhodného pro rychlou rekonstrukci knihy objednávek. Výsledkem by měla být implementace navrženého řešení včetně možnosti vizualizace získané knihy objednávek a základních operací nad ní. Takto vytvořený systém by měl pomoci v další analýze finančního trhu.

Abstract

The main goal of this bachelor's thesis is to design and implement a system for processing stock market data. The thesis will first examine in detail the functionalities of the electronic order book system of the German multinational company Deutsche Börse Group, and also examine the stock market data that this system provides. Based on this analysis, algorithms to process and store the data in a format suitable for rapid reconstruction of the order book will be designed. The result of this thesis should be an implementation of the proposed system, including the possibility to visualize the order book and perform basic operations on it. The system should help in further analysis of the financial market.

Klíčová slova

Elektronická kniha objednávek • zpracování a analýza dat • rekonstrukce EOBI

Poděkování

Rád bych poděkoval Ing. Kamilu Ekšteinovi, Ph.D. a doc. Ing. Janu Pospíšilovi, Ph.D. za odborné vedení této práce, věcné připomínky a cenné rady.

Obsah

Úvod 1

Elektronická kniha objednávek (Electronic Order Book), je klíčovým nástrojem užívaným finančními institucemi po celém světe. Poskytuje detailní přehled o nabídkových a poptávkových cenách akcií, měn, derivátů a dalších finančních instrumentů, a to včetně objemu obchodovaných jednotek. Napomáhá transparentnosti a efektivitě finančních trhů a je nedílnou součástí analytických nástrojů investorů a jiných účastníků tohoto segmentu trhu.

Tato bakalářská práce se bude prvotně zaměřovat na pochopení samotného fungování elektronické knihy objednávek a její interpretaci v praxi. Díky spolupráci s německou nadnárodní společností Deutsche Börse Group byla pro tuto práci poskytnuta reálná data, na nichž budou probíhat veškeré analýzy a testy.

Na základě získaných poznatků pak budou v rámci této práce navrženy a vytvořeny algoritmy, které zpracovávávají poskytnutá data a rekonstruují samotnou elektronickou knihu objednávek. Tento výstup by měl pomoci v dalším navazujícím výzkumu v oblasti analýzy tržních dat a fungování finančních mechanismů.

Elektronická kniha objednávek

2

Velké množství burz a jiných finančních institucí dnes ***EOB*** (Electronic Order Book) používá jako nástroj pro obchodování. Pro práci s burzovními daty je potřeba nejprve pochopit fungování a význam samotné knihy objednávek. V této kapitole proto budou detailně rozbrány základní principy obchodování na burzách a tvoření a typy objednávkových knih.

2.1 Objednávka

Objednávky (Orders) jsou základní stavební kameny obchodování a obchodních strategií. Jsou to instrukce, které definují, co chce účastník trhu obchodvat, zda prodává či nakupuje, kolik, kdy a především za jakých podmínek chce obchodovat. [Har02]

Tyto instrukce předávají obchodníci (Traders) makléřům (Brokers) či přímo burzám, kteří jejich objednávku sjednají. Objednávka musí vždy specifikovat zda obchodník prodává nebo nakupuje a kolik jakého instrumentu (nebo instrumentů). Může také dále určovat podmínky, které musí obchod splnit. Například jakou limitní cenu je obchodík ochoten akceptovat, jak dlouho má být objednávka validní, kdy může nejdříve dojít k naplnění nebo zda může být naplněna pouze částečně. Některé objednávky mohou dokonce vymezovat skupinu jiných obchodníků, se kterými může být obchod uzavřen. [Har02]

Objednávky dělíme na ***nabídkové*** (Bid) a ***poptávkové*** (Ask či Offer). Cena je pak identifikována jako ***nabídková cena*** (Bidding Price) či ***poptávková cena*** (Asking Price) a ***velikost*** (Size) určuje obchodované množství. Nejvyšší nabídková cena na trhu je označena jako ***nejlepší nabídka*** (Best Bid) a nejnižší poptávková cena poté ***nejlepší poptávka*** (Best Ask). Jedná se o nejlepší aktuálně dostupné ceny na konkrétním trhu. [Har02]

2.1.1 Typy objednávek

Různé trhy komodit (a také různé finanční instituce) umožňují obchodníkům specifikovat více typů objednávek, které poskytují určitou investiční volnost při plánování obchodu. Typy objednávek mohou výrazně ovlivnit výsledky obchodu.

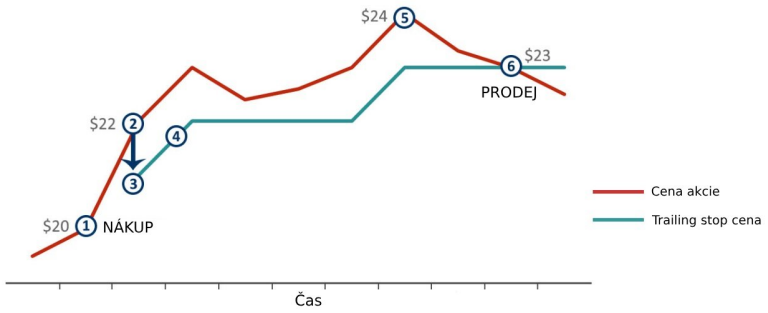
Market objednávka. Tento typ je pokynem k nákupu nebo prodeji instrumentů za nejlepší aktuálně dostupnou cenu. Obvykle je objednávka prováděna okamžitě. Nicméně cena, za kterou bude market objednávka provedena, není zaručena. Je důležité si uvědomit, že poslední obchodovaná cena nemusí nutně být cenou, za kterou bude market objednávka provedena. V rychle se pohybujících trzích se cena provedení objednávky často liší od poslední obchodované ceny. Při velkém obchodovaném množství také můžou být různé části objednávky vykonány při různých cenách. [SEC17]

Limitní objednávka. Je pokynem k nákupu nebo prodeji instrumentů za pevně danou limitní cenu nebo lepší. Poptávková limitní objednávka může být provedena pouze za limitní cenu nebo nižší. Prodejní pak může být provedena pouze za limitní cenu nebo naopak vyšší. Tento typ negarantuje vykonání objednávky, ale zaručuje obchodníkovi, že nezaplatí více než předem stanovenou cenu. [SEC17]

Stop-Loss objednávka. Je pokyn k nákupu nebo prodeji instrumentů, jakmile jejich cena dosáhne stanovené ***stop ceny*** (Stop Price). Jakmile je dosaženo této ceny, ***stop-loss objednávka*** se stává ***market objednávkou***. Poptávková objednávka zadává stop cenu nad současnou tržní cenou. Nabídková objednávka naopak zadává stop cenu pod současnou tržní cenou. [SEC17]

Stop-Limit objednávka. Jedná se o kombinaci stop-loss a limitní objednávky. Jakmile instrument dosáhne definované stop ceny, stává se z této objednávky limitní objednávka, která bude provedena za stanovenou cenu (nebo lepší). Výhodou stop-limit objednávky je, že obchodník má kontrolu nad cenou, při které se může objednávka vykonat. [SEC17]

Trailing Stop objednávka. Je typ stop-loss nebo stop-limit objednávky, ve které není stop cena vyjádřena konkrétní hodnotou. Místo toho je buď definovaným procentem nebo částkou nad či pod současnou tržní cenou instrumentu. Pokud se cena instrumentu pohybuje chtěným směrem, trailing stop cena se přizpůsobuje a sleduje tržní cenu o stanovenou částku. Pokud se však směr tržní ceny otočí, trailing stop cena zůstává neměnná a objednávka bude provedena, pokud se ceny protnou. [SEC17]

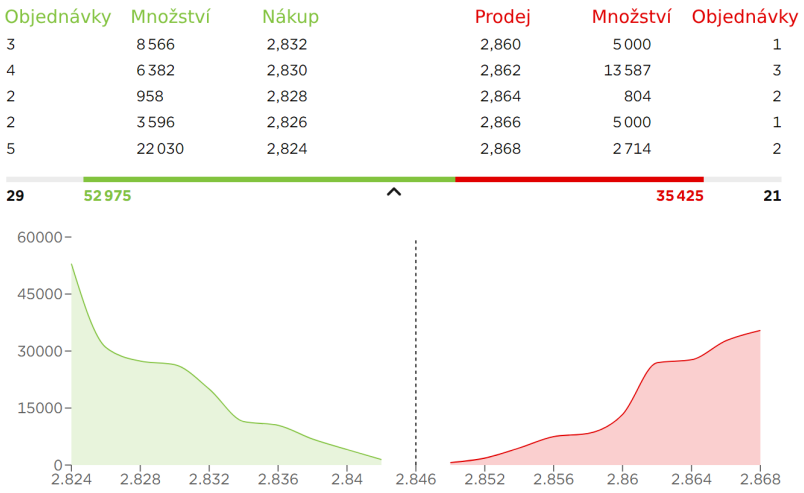


Obrázek 2.1: Ukázka nabídkové trailing stop objednávky (`https://shorturl.at/gqvBC`).

2.2 Limitní kniha objednávek

Limitní kniha objednávek (Limit Order Book, LOB) seskupuje výše popsané objednávky do jednoho celku. Obsahuje kompletní seznam nabídkových a poptávkových objednávek pro určitý instrument v daném čase a umožnuje obchodníkovi nahlížet na hloubku trhu, která je důležitá pro obchodní strategie.

Pokud do LOB dorazí nová nákupní či prodejní objednávka, tak algoritmus vypořádání objednávek zkontroluje, zda je možné příchozí objednávku přiřadit k nějaké dříve podané prodejní (respektive nákupní) objednávce. Pokud ano, vykonání obchodu proběhne okamžitě. Pokud ne, objednávka se stane aktivní a zůstává v tomto stavu, dokud není přiřazena k další příchozí prodejní (respektive nákupní) objednávce nebo zrušena. Zrušení typicky nastává v případě, že vlastník objednávky již nechce nabízet za uvedenou cenu. [Gou+13]



Obrázek 2.2: Ukázka limitní knihy objednávek (`https://shorturl.at/bdCN1`).

2.2.1 Význam pro trh

Před vznikem LOB probíhala většina finančních obchodů na ***kotacemi řízených trzích*** (Quote-driven Markets). Zde skupina velkých tvůrců trhu (Market Makers) centralizovala poptávkové a nabídkové objednávky tak, že zveřejňovali ceny, za které jsou ochotni nakoupit a prodat obchodovaný instrument. Prodejní cenu stanovovali vyšší než nákupní, aby získali zisk za poskytnutí likvidity a za převzetí rizika. [Gou+13]

Trh, na kterém se kupující a prodávající setkávají prostřednictvím LOB, se nazývá ***objednávkami řízený trh*** (Order-driven Market). Na těchto trzích jsou nákupní a prodejní objednávky postupně přiřazovány v čase podle určitých pravidel priority. Priorita je vždy založena na ceně a poté, ve většině trhů, na čase podle pravidla FIFO (přednost mají dříve vytvořené objednávky). [Abe+16]

Tvorba ceny. LOB hraje významnou roli v procesu tvorby ceny. Kupující a prodávající určují cenu obchodovaného instrumentu na základě nabídky a poptávky. Nejvyšší kupní cena a nejnižší prodejní cena tvoří rozpětí, které je aktuální tržní cenou instrumentu. S přicházejícími objednávkami a plněním stávajících se tržní cena neustále mění.

Vliv velkých objednávek. Příchod velké objednávky může razantně ovlivnit tržní cenu instrumentu. Například velká nákupní objednávka může vypořádat více prodejních objednávek na nejnižších cenách a tím celkovou tržní cenu zvýšit. Obchodníci mohou toto chování analyzovat a upravovat své obchodní strategie.

Detekce manipulace na trhu. Kniha objednávek může také pomoci při odhalování potenciální manipulace na trhu. Příkladem může být taktika známá jako ***spoofing***, kdy obchodník umístí velkou objednávku, aby vytvořil iluzi vysoké poptávky nebo nabídky, ovlivní tím ostatní obchodníky a poté svou objednávku zruší. Kniha objednávek zvyšuje transparentnost trhu a slouží tím jako důležitý nástroj pro regulátory a finanční instituce k udržování spravedlivého trhu. [Tam23]

T7 EOBI 3

Pro vypracování této bakalářské práce byla využita data z konkrétního burzovního systému německé nadnárodní společnosti Deutsche Börse Group. Jejich ***T7 Enhanced Order Book Interface*** umožňuje získat zákazníkům informace o stavu a pohybech v objednávkových knihách vybraných obchodovaných instrumentů v reálném čase. V této kapitole bude popsán zmíněný systém.

3.1 Představení

T7 Enhanced Order Book Interface poskytuje kompletní viditelnou knihu objednávek. Zákazníkům jsou zasílány ucelené informace o každé individuální objednávce, jejich vykonání a stavu v reálném čase. Vedle ***T7 EMDI*** (Enhanced Market Data Interface) a ***T7 MDI*** (Market Data Interface) se jedná o další rozhraní, které nabízí přístup k obchodním datům.

Přestože velká část funkcionality je podobná zmíněnému **T7 EMDI**, toto rozhraní je navrženo pro zákazníky, kteří požadují vyšší transparentnost, nízkou latenci a vysokou propustnost dat. Data jsou zasílána pomocí datagramů na multicastové doméně, kam se zákazníci připojují. Mezi klíčové vlastnosti tohoto rozhraní patří:

- Data jsou zasílána v plném rozsahu bez omezení hloubky trhu.
- Informace jsou šířeny pomocí binárních zpráv fixní délky. Ty mohou mít jednak podobu inkrementačních zpráv, které obsahují změny ve stavu knihy objednávek od poslední zprávy (například nové, zrušené objednávky nebo provedené obchody). Nebo podobu tzv. ***snapshotů***, které nesou kompletní stav knihy objednávky v čase zaslání zprávy.
- Všechny inkrementační zprávy a snapshoty dodržují specifické pořadí zaslání:
    1. strana (Side): nejprve nabídka, poté poptávka,
    2. cena (Price): nejlepší cena první,
    3. čas (Time): nejvyšší časová priorita první.

3.2 Funkční charakteristiky

Rozhraní **T7 EOBI** poskytuje podrobná tržní data v reálném čase pro obchodovatelné instrumenty. Informace, které tento systém poskytuje:

- **Kompletní knihu objednávek:** Na rozdíl od ostatních rozhraní, které omezují hloubku knihy objednávek (počet zobrazených objednávek), **T7 EOBI** poskytuje informace o všech viditelných objednávkách, včetně ***identifikátoru instrumentu*** (Instrument Identifier), strany objednávky (nákup či prodej), ceny, množství a ***časové priority*** (Priority Timestamp).

- **Informace o vykonaných obchodech:** Pro každý uskutečněný obchod zašle cenu, za kterou se obchod vykonal, a množství.

- **Stav trhu:** Poskytuje informace o obchodním stavu jednotlivých produktů a jejich instrumentů, včetně všech změn během dne u komplexních instrumentů.

- **Data pro obnovu:** Poskytuje snapshoty celých tržních dat v konkrétních časech, což umožňuje obnovu dat v případě výpadků.

Každá objednávka je **jednoznačně** identifikovatelná pomocí kombinace tří parametrů: identifikátoru instrumentu, strany a časové priority.

Aby bylo možné tržní data odesílat co nejrychleji, zveřejňuje rozhraní **T7 EOBI** pouze velmi specifické tržní informace, ze kterých si zákazníci mohou zbytek informací odvodit sami. Analýzou jednotlivých objednávek lze odvodit například cenová hladina a agregace objednávek na dané hladině. Ze zpráv o provedení obchodu lze identifikovat objednávky, které byly zcela vyplněny.

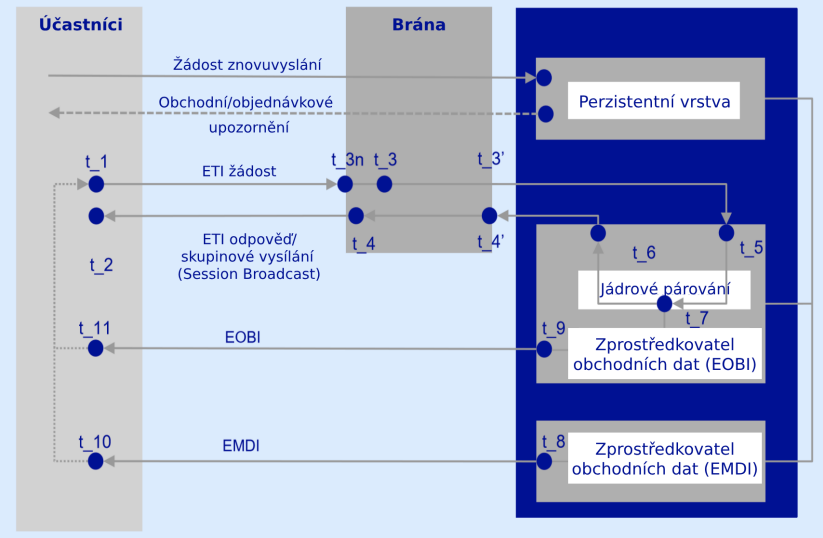
3.2.1 Časové značky

Systém **T7** používá různé typy ***časových značek*** (Timestamps). Ty se generují na vysokofrekvenčních ***branách*** (Gateways), ***párovacích algoritmech*** (Matching Engines) a na ***serverech tržních dat*** (Market Data Servers). Tyto časové značky lze využít k analýze jednosměrné doby přenosu dat. Pro každou příchozí transakci jsou časové značky vytvořeny v následujících místech zpracování:

- Brána: Při přijetí transakce bránou.

- Párovací algorimy: Při údržbě knih objednávek, při vytváření přímých zpráv nebo exekučních zpráv pro pasivní objednávky.

- Rozhraní obchodních dat (**T7 EMDI**, **T7 MDI**, **T7 EOBI**): Při změnách v knize objednávek a tvoření snapshotů (*SendingTime*), dále také při tvorbě časových značek: *Matching Engine-In timestamp, PriorityTimestamp, TransactTime, Gateway-In* použitých ve zprávách.



Obrázek 3.1: Přehled časových značek v systému **T7** ([Deu20]).

Popis jednotlivých časových značek:

- t_3n: Čas zaznamenaný bránou ETI, když první bit požadavku dorazí na síťovou kartu ***brány PS*** (Partition Specific Gateway). Pokud je dostupný objeví se ve značkách *RequestTime, TrdRegTSTimeIn*.
- t_3: Čas zaznamenaný aplikační bránou ETI v momentě přečtení požadavku ze soketu na straně brány účastníka. Pokud *t3_n* není dostupný, objeví se tento čas ve značkách *RequestTime, TrdRegTSTimeIn*.
- t_5: Čas zaznamenaný párovacím algoritmem při přečtení požadavku (*AggressorTime*).
- t_7: Čas zaznamenaný v momentě funkčního zpracování transakce. Je unikátní pro každý produkt. Může se objevit v kterékoliv ze značek jako *TrdRegTSTimePriority, ExecID, TransactTime* a dalších, v závislosti na tom, zda odpovídá nové objednávce, kotační transakci, vytvoření strategie, provedení objednávky nebo jako transakční časová značka pro jiné účely.
- t_8: Čas odeslání datagramu na síť rozhraním **T7 EMDI** (*SendingTime*).
- t_9: Čas odeslání datagramu na síť rozhraním **T7 EOBI** (*TransactTime*).

3.3 Sestavení knihy objednávek

Pro každý instrument v rámci produktu lze pomocí ***T7 EOBI snapshot kanálu*** přijímat zprávy se snapshoty a vytvořit tak počáteční knihu objednávek. Po jejím vytvoření se musí kniha objednávek udržovat pomocí odpovídajících aktualizací knihy objednávek přijatých na ***T7 EOBI incremental kanálu***.

3.3.1 Datové typy

T7 EOBI používá ve zprávách různé datové typy. Následující výčet je výběr typů použitých v níže popsaných zprávách.

- *signed int*: 1, 2, 4 nebo 8-bytový celočíselný datový typ se znaménkem na nejvyšším bitu. Pořadí bytů je dáno variantou little endian, to znamená, že na nejnižší adresu se ukládá nejméně významný byte ([Endianita]).
- *unsigned int*: 1, 2, 4 nebo 8-bytový celočíselný datový typ bez znaménka. Pořadí bytů je dáno variantou little endian ([Endianita]).
- *PriceType*: Cena vyjádřená pomocí 8-bytového celočíselného datového typu se znaménkem a s fixními 8 desetinnými místy. Pro určité produkty mohou být ceny záporné.
- *QuantityType*: Množství vyjádřené pomocí 8-bytového celočíselného datového typu se znaménkem a s fixními 4 desetinnými místy.
- *UTCTimestamp*: Datum a čas v UTC pásmu vyjádřený 8-bytovým celočíselným datovým typem bez znaménka jako počet nanosekund od UNIX epochy (1. ledna 1970 00:00:00 UTC).

3.3.2 Typy zpráv

Rozhraní **T7 EOBI** poskytuje pro každou aktualizaci v knize objednávek explicitní zprávu. Níže budou popsány zprávy relevantní pro sestavení počáteční knihy objednávek.

3.3.2.1 Přidání objednávky

Vždy, když je do knihy objednávek příslušného instrumentu přidána viditelná objednávka, rozhraní odešle zprávu **Order Add** informující o přidání objednávky. Zpráva obsahuje tyto informace:

- *TrdRegTSTimeIn*: čas přijetí požadavku na přidání objednávky bránou rozhraní,

- *SecurityID*: identifikátor instrumentu,
- *TrdRegTSTimePriority*: čas priority,
- *DisplayQty*: zobrazené množství,
- *Side*: strana objednávky,
- *Price*: cena.

Pro unikátní identifikaci objednávky jsou zde použity hodnoty *SecurityID*, *Side* a *TrdRegTSTimePriority*. Tyto hodnoty jsou zároveň referenčním klíčem pro všechny budoucí aktualizace a změny objednávky.

Informace o příchozí objednávce, která byla zcela spárována s jednou nebo více objednávkami v knize objednávek, lze odvodit z přidružených zpráv o vykonání objednávky. Pokud je příchozí objednávka spárována pouze částečně, bude zbývající množství zasláno novou zprávou **Order Add**, avšak až po všech souvisejících zprávách o vykonání.

3.3.2.2 Změna objednávky

Pokud dojde ke změně časové priority, ceny nebo zobrazeného množství existující objednávky, odešle se zpráva o úpravě objednávky **Order Modify** nebo o úpravě objednávky se zachováním časové priority **Order Modify Same Priority**.

Order Modify se zašle v případě, že objednávce bude změnou přiřazena nová časová priorita. Taková změna je způsobena například změnou ceny nebo zvýšením množství. Zpráva obsahuje tyto informace:

- *TrdRegTSTimeIn*: čas přijetí požadavku na změnu objednávky bránou rozhraní,
- *TrdRegTSPrevTimePriority*: původní časová priorita objednávky,
- *PrevPrice*: původní cena,
- *PrevDisplayQty*: původní množství,
- *SecurityID*: identifikátor instrumentu,
- *TrdRegTSTimePriority*: nová časová priorita objednávky,
- *DisplayQty*: nové množství,
- *Side*: strana objednávky,
- *Price*: nová cena.

Novým referenčním klíčem pro identifikaci se stává nová priorita *TrdRegTSTimePriority* v kombinaci s *SecurityID* a *Side*.

Order Modify Same Priority se zašle v případě, že změna objednávky nezmění její časovou prioritu. Taková změna je například snížení množství objednávky. Oproti výše popsaným atributům tato zpráva neobsahuje *TrdRegTSPrevTimePriority* a *PrevPrice*, jelikož k modifikacím nedošlo. Naopak obsahuje navíc transakční čas (*TransactTime*).

3.3.2.3 Odstranění objednávky

Pokud dojde ke zrušení viditelné objednávky, zašle se zpráva **Order Delete**. Tato zpráva obsahuje všechny důležité parametry, aby bylo možné dotčenou objednávku rychle identifikovat a smazat z knihy objednávek. Obsažené informace jsou:

- *TrdRegTSTimeIn*: čas přijetí požadavku na odstranění objednávky bránou rozhraní,
- *TransactTime*: transakční čas,
- *SecurityID*: identifikátor instrumentu,
- *TrdRegTSTimePriority*: časová priorita objednávky,
- *DisplayQty*: zobrazené množství,
- *Side*: strana objednávky,
- *Price*: cena.

3.3.2.4 Vykonání objednávky

Aby mohli zákazníci snadněji zpracovávat vykonání objednávek a další aktualizace knihy objednávek, posílá rozhraní `T7 EOBI` při každém spárování příchozí objednávky specifické zprávy **Execution Summary**, **Partial Order Execution** a **Full Order Execution**.

Zpráva **Execution Summary** je zaslána v případě, že příchozí objednávka byla spárována s objednávkou či objednávkami, které jsou již aktivní v knize objednávek. Zpráva obsahuje:

- *SecurityID*: identifikátor instrumentu,
- *ExecID*: čas spárování objednávek,
- *LastQty*: celkové spárované množství,

- *AggressorSide*: strana příchozí objednávky,
- *LastPx*: nejhorší cena, za kterou byla objednávka spárována.

Dále také udává čas přijetí požadavku bránou rozhraní (*RequestTime*) a čas přijetí párovacím systémem (*AggressorTime*).

Zpráva **Execution Summary** může být využita pro rychlé obchodování (fast trading), ale i pro kontrolu správnosti knihy objednávek po zpracování dílčích zpráv o vykonání objednávek popsaných níže.

Zpráva **Partial Order Execution** obsahuje informace o individuálním provedení objednávky a je zaslána v případě, že objednávka byla pouze částečně provedena. Zpráva obsahuje:

- *Side*: strana vykonané objednávky,
- *Price*: cena, se kterou byla objednávka vytvořena,
- *TrdRegTSTimePriority*: časová priorita,
- *SecurityID*: identifikátor instrumentu,
- *LastQty*: množství spárované v tomto vykonání,
- *LastPx*: cena, za kterou se objednávka spárovala.

Zbývající množství pro dotčenou objednávku v knize objednávek se musí vypočítat odečtením spárovaného množství (*LastQty*) od původního množství v knize objednávek.

Zpráva **Full Order Execution** obsahuje informace o individuálním provedení objednávky a je zaslána v případě, že objednávka byla provedena v kompletním množství a za zobrazenou cenu. Zpráva obsahuje stejné informace jako výše zmíněná zpráva pro částečné vykonání objednávky.

Obě dílčí zprávy o vykonání objednávky také obsahují pole *TrdMatchID*, které představuje unikátní identifikátor (přes všechny produkty v rámci daného dne) jednotlivých kroků spárování objednávky. [Deu20]

Návrh řešení 4

Tato kapitola se zabývá návrhem řešení zpracování poskytnutých burzovních dat. Data byla získána z výše popsaného systému **T7 EOBI** v podobě CSV souborů, které obsahují zprávy popsané v kapitole 3.3.2 za různá časová období a pro různé obchodované instrumenty, a ze kterých lze knihu objednávek rekonstruovat. Cílem bylo navrhnout škálovatelné řešení, které umožní velký objem dat efektivně uložit a zpracovat do přehledného formátu pro rychlou rekonstrukci a vizualizaci knihy objednávek, která se poté stane základem pro další analýzy trhu. Navrhované řešení se skládá ze dvou hlavních částí:

- **Uložení dat**: Tato část se zaměřuje na ukládání burzovních dat v efektivním a přístupném formátu. Hlavním požadavkem bylo otestovat, zda by pro účely této práce nešlo využít nějakou ze známých, běžně používaných databází poskytovaných zdarma.

- **Předzpracování dat a rekonstrukce**: Tato část se zaměřuje na předzpracování a transformaci dat do jednotného formátu, který se stane základem pro následnou rekonstrukci knihy objednávek. Zahrnuje to sloučení dat, odstranění nepotřebných dat (například odstranění parametrů zpráv, které pro rekonstrukci nejsou důležité, nebo odstranění celých zpráv, které se týkají jiného instrumentu), transformaci hodnot (například časových značek nebo množství) a normalizaci formátu dat.

4.1 Uložení dat

Vzhledem k charakteru dat a jejich velkému objemu byly otestovány různé typy databází. Prvním zástupcem je **InfluxDB**, která je ***NoSQL*** (Not only SQL) a je optimalizována na ukládání ***časových řad*** (Time Series). Dále standardní a velmi oblíbený ***relační databázový systém*** (Relational Database Management System neboli RDBMS) **MariaDB**. A nakonec **SQLite**, která pro fungování nevyžaduje databázový server.

4.1.1 InfluxDB

Jedná se o zdarma dostupnou NoSQL ***databázi časových řad*** (Time Series Database neboli TSDB).

NoSQL databáze. Představují alternativu k tradičním relačním databázím, jako je například **MySQL** nebo **PostgreSQL**. Jsou používány pro práci s velkými objemy nestrukturovaných dat, které se obtížně vkládají do tabulkové struktury relačních databází.

Jsou navrženy pro nerelační datové modely a v posledních letech se stávají oblíbenými pro vývoj moderních aplikací. Ty čelí několika výzvám, které lze pomocí NoSQL databází vyřešit. Například aplikace zpracovávají velké objemy dat z různých zdrojů, jako jsou sociální média, chytré senzory a databáze třetích stran. Všechna tato různorodá data se nemusí hodit do relačního modelu. Vynucování tabulkových struktur může vést k redundanci a problémům s výkonem při škálování. Do určité míry lze ale problémy vyřešit normalizací tabulek v relačním modelu. Tento postup se skládá z reorganizace tabulek tak, aby splňovaly pravidla normálních forem (až 5 normálních forem). Přílišná normalizace však může být i kontraproduktivní a vést ke snížení výkonu.

Tyto databáze obecně poskytují flexibilnější datové modely, které umožňují rychlejší a iterativní vývoj. Nabízí například datové modely pro ukládání grafových struktur (pro sociální média) nebo hierarchických struktur (JSON, XML). Jsou vhodné pro práci s polostrukturovanými a nestrukturovanými daty. Podporují také škálovatelnost, a jsou typicky navrženy pro vertikální škálování pomocí distribuovaných hardwarových clusterů, na rozdíl od horizontálního škálování, které vyžaduje přidávání dalších serverů. NoSQL databáze také mohou nabízet speciální rozhraní a datové typy, navržené pro konkrétní datový model dané databáze. V závisloti na typu úlohy mohou dosahovat vyššího výkonu, než relační databáze. [Ama]

Databáze časových řad. Tento typ databází je optimalizovaný na ukládání a práci s velkým objemem ***časově orientovaných dat*** (Time Series Data). Ty jsou obecně dány ***časovou značkou*** (Timestamp) a přiřazenými hodnotami k danému času. Typickým příkladem takových dat jsou měření ze senzorů, čidel, telemetrie aplikací či události na finančním trhu. Databáze časových řad jsou charakteristické v přístupu k těmto datům. Umožňují rychlý zápis, kompresi, přístup k informacím v určitém časovém rozmezí a jejich sumarizaci.

Databáze časových řad se od ostatních databází zásadně liší svou architekturou a vlastnostmi. TSDB jsou optimalizovány pro efektivní ukládání časově orientovaných dat a jejich kompresi. Umožňují také definovat pravidla pro správu životního cyklu dat. V těchto databázích je běžné uchovávat data s vysokou přesností po krát-

kou dobu a poté je smazat nebo agregovat do dlouhodobějších agregátů. To znamená, že pro každý datový bod uložený v databázi musí existovat mechanismus pro jeho smazání po uplynutí jeho doby platnosti. Implementace takové správy životního cyklu dat je pro vývojáře aplikací nad běžnými databázemi složitá. S TSDB je tato funkce k dispozici ihned. Při práci s databázemi časových řad je také běžné požadovat shrnutí dat za rozsáhlé časové období. To vyžaduje procházení velkého množství datových bodů a provádění výpočtů. Příkladem můžou být výpočty různých metrik či statistik vývoje dat v měsících a jejich porovnání. Tento typ dotazu je pro tradiční databáze obtížně optimalizovatelný. TSDB jsou naopak pro takové použití stavěny. [Dix]

InfluxDB nebyla vytvořena z jiného typu databáze, ale byla od základu navržena jako specializovaná databáze pro časové řady. Je součástí komplexní platformy, která umožňuje sběr, ukládání, monitorování a vizualizaci časově orientovaných dat.

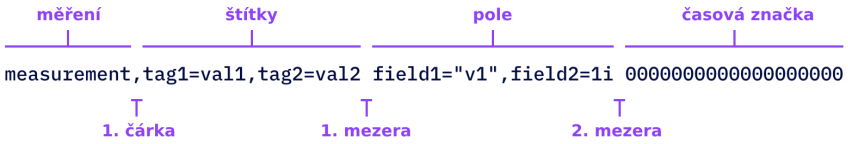
Datový model **InfluxDB** se výrazně liší od jiných řešení pro časově orientovaná data, jako jsou například **Graphite**, **RRD** nebo **OpenTSDB**. **InfluxDB** používá pro odesílání dat ***řádkový protokol*** (Line Protocol), který obsahuje následující atributy: ***měření*** (Measurement), ***štítky*** (Tag-set), ***pole*** (Field-set) a ***časovou značku*** (Timestamp). [Dix]

Měření (Measurement). Funguje v **InfluxDB** jako kontejner pro štítky, pole a časovou značku. Měření pak popisuje data, která jsou uložena ve zmíněných atributech. Měření jsou typu ***řetězec*** (String) a při srovnání s relačními databázemi se měření konceptuálně podobá tabulkám. Jedno měření může spadat pod více ***pravidel uchování dat*** (Retention Policies). Ty definují, jak dlouho jsou data v **InfluxDB** uložena a kolik kopií těchto dat je uloženo v clusteru. Měření jsou uložena v ***invertovaném indexu*** (Inverted Index), což umožňuje velmi rychlé vyhledávání konkrétních sérií dat.

Štítky (Tag-set). Jsou kolekcí ***párů klíčů a hodnot*** (Key/Value Pair), kde všechny hodnoty musí být typu řetězec. V **InfluxDB** jsou volitelné a nemusí se v datové struktuře používat. Stejně jako měření jsou štítky uloženy v invertovaném indexu. Je tedy vhodné je používat pro ukládání dat, které se často filtrují a vyhledávají.

Pole (Field-set). Jsou kolekcí párů klíčů a hodnot, kde hodnoty mohou být ***celočíselné*** (int64), ***desetinná čísla*** (float64), ***pravdivostní*** (bool) a nebo řetězcová. Pole jsou povinnou součástí datových struktur v **InfluxDB**, bez nich nelze ukládat data. Oproti měření a štítkům nejsou pole indexovaná. Dotazy, které používají ve filtrech hodnoty polí, musejí projít všechny záznamy a porovnávat splnění podmínek dotazu. To vede k horšímu výkonu oproti dotazům nad štítky. Obecně by tedy pole neměla obsahovat data, podle kterých se vyhledávají série dat. [Infa]

Časová značka (Timestamp). Je povinnou součástí datových struktur v **InfluxDB**. Mohou mít přesnost sekund, milisekund, mikrosekund nebo nanosekund. Díky možnosti nanosekund je **InfluxDB** vhodnou volbou pro oblasti financí a vědeckých výpočtů, kde by jiná řešení mohla být nedostatečně přesná. [Dix]



Obrázek 4.1: Řádkový protokol v **InfluxDB** (`https://shorturl.at/bixNT`).

InfluxDB používá jako jádro databáze stroj ***TSM*** (Time-Structured Merge Tree) . Je založen na ***LSM*** (Log-Structured Merge Tree). Jeho hlavní komponenty jsou ***WAL*** (Write Ahead Log), ***Cache***, ***soubory TSM*** a ***TSI*** (Time Series Index).

Příchozí data jsou řazena, komprimována a zapsána do souboru WAL pro okamžitou zálohu. Současně s tím jsou také zapsána do paměti cache, kde jsou automaticky dostupná pro vyhledávání a dotazy nad nimi. Paměť cache je periodicky zapisována na disk v podobě souborů TSM. S přibývajícím časem se hromadí jednotlivé soubory TSM, a tak je databázový stroj slučuje a komprimuje do souborů TSM vyšších úrovní. [Infb]

InfluxDB nabízí dva ***dotazovací*** (Query) jazyky nad daty. Vlastní skriptovací jazyk **Flux** optimalizovaný na časově orientovaná data a jazyk podobný ***SQL*** (Structured Query Language) **InfluxQL**.

Flux. Je skriptovací a dotazovací jazyk vyvinutý společností InfluxData. Na rozdíl od **InfluxQL** umožňuje **Flux** práci s různými zdroji dat (včetně **SQL** databází a souborů CSV), což usnadňuje integraci datových analýz napříč různými sériemi dat. Syntaxe a funkcionalita jazyka **Flux** se výrazně liší od **InfluxQL** nebo **SQL**. **Flux** je funkcionální jazyk vybavený operátory a funkcemi určenými pro komplexní transformace dat a analytické operace. Zaměřuje se převážně na manipulaci s časově orientovanými daty a matematické operace nad nimi.

Flux oproti **InfluxQL** rozšiřuje možnosti práce s daty. Nabízí nástroje pro provádění detailních analýz dat, které by v jednodušších dotazovacích jazycích mohly být obtížně proveditelné. Poskytuje funkce pro úpravu a transformaci dat před jejich analýzou. Data lze filtrovat, agregovat hodnoty nebo vytvářet zcela nové odvozené statistiky. **Flux** umožňuje provádět i základní úlohy strojového učení přímo v rámci

dotazovacího jazyka, což vede k automatizaci některých analytických procesů v pokročilých scénářích zpracování dat.

Flux je mladý a stále se vyvíjející jazyk ve srovnání s léta prověřenými dotazovacími jazyky, jako je **SQL**. Důsledkem toho je, že zatím nedisponuje tak rozsáhlými knihovnami funkcí a ani tak silnou komunitní podporou. Neustálý vývoj **Fluxu** s sebou nese i riziko změn a aktualizací, které mohou ovlivnit fungování stávajících skriptů a aplikací. Oproti zavedeným jazykům jako **SQL** je také k dispozici méně dokumentace, příkladů a obecných znalostí v rámci komunity. Jeho proces vývoje má vliv i na výkon. Optimalizace, techniky správy zdrojů a celkový výkon se neustále zdokonalují, ale nemusí zatím dosahovat úrovní zavedených jazyků. [Sot]

Zdrojový kód 4.1: Ukázka jazyka **Flux** ([Sot]).

```
from(bucket: "my-bucket")
  |> range(start: -24h)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> aggregateWindow(every: 1h, fn: mean)
```

InfluxQL. Je dotazovací jazyk navržený speciálně pro práci s databází **InfluxDB** verze 1.x. Syntaxe **InfluxQL** je podobná jazyku **SQL**, což usnadňuje jeho učení a používání pro vývojáře již obeznámené s **SQL**. Hlavní síla **InfluxQL** spočívá v jeho specializovaných funkcích a operátorech určených pro práci s časově orientovanými daty. Tyto funkce a operátory zahrnují filtrování, agregaci a transformaci dat. **InfluxQL** navíc podporuje ***kontinuální dotazy*** (Continuous Queries) a definici pravidel uchovávání dat, což umožňuje efektivní zpracování a správu časově orientovaných dat v reálném čase.

Ačkoliv **InfluxQL** sdílí podobnou syntaxi s jazykem **SQL**, v oblasti flexibility a možností za ním zaostává. **InfluxQL** je navržen speciálně pro práci s časově orientovanými daty. To má výhody při práci s tímto typem dat, ale pro jiná data nebude dosahovat možností, které nabízí **SQL**. Na omezení použití mohou narazit uživatelé, kteří potřebují provádět komplexnější manipulace, transformace a analýzy dat jdoucí nad rámec základních agregací a filtrování. Příkladem tohoto omezení může být absence funkce ***JOIN***, která je v relačních databázích nepostradatelnou funkcí pro spojení dat z různých tabulek. V **InfluxQL** mohou dotazy zahrnující složité výpočty, transformace dat nebo spojování více měření být obtížné, nebo dokonce zcela nemožné. [Sot]

Zdrojový kód 4.2: Ukázka jazyka **InfluxQL** ([Sot]).

```
SELECT
   MEAN("value")
FROM
   "temperature"
WHERE
   time > now() - 24h
GROUP BY
   time(1h)
```

4.1.2 MariaDB

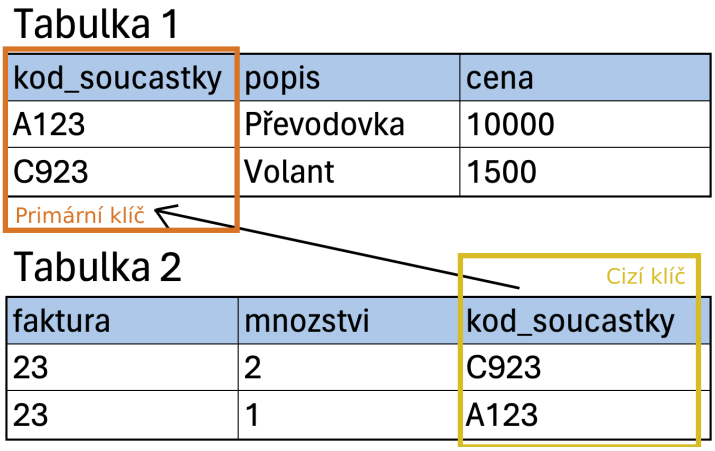
Jedná se o zdarma dostupný ***relační databázový systém*** (RDBMS).

Relační databáze. Je typ databáze, která data ukládá do tabulek. Tyto tabulky jsou tvořené řádky a sloupci. Každý sloupec má svůj název a definuje typ dat, která v něm mohou být uložena. Řádky v tabulce pak obsahují konkrétní záznamy vztahující se k dané tabulce. Mezi tabulkami lze vytvářet relace (vztahy), díky kterým lze jednoduše pochopit propojení mezi souvisejícími daty. Díky této interpretaci dat jsou relační databáze velmi populární pro práci s velkými objemy strukturovaných dat.

Data jsou typicky strukturována napříč několika tabulkami, které mohou být spojeny pomocí ***primárního klíče*** (Primary Key) nebo ***cizího klíče*** (Foreign Key). Tyto jednoznačné identifikátory definují rozdílné vztahy, které existují mezi tabulkami. Vztahy mezi tabulkami jsou typicky ilustrovány pomocí různých datových modelů.

Relační databáze jsou také typicky spojovány s ***transakčními databázemi*** (Transactional Databases), které se starají o vykonání příkazů neboli transakcí. Transakce mají specifické vlastnosti definované zkratkou ***ACID*** (Atomicity, Consistency, Isolation, Durability).

- ***Atomicita***: Všechny změny dat jsou provedeny tak, jako by se jednalo o jednu operaci. To znamená, že buď proběhnout všechny změny, nebo žádná z nich.
- ***Konzistence***: Data zůstávají v konzistentním stavu od počátku do konce, aby byla zaručena integrita dat.
- ***Izolace***: Mezistav jedné transakce není viditelný pro jiné transakce. Souběžně probíhající transakce se tedy jeví jako serializované (za sebou jdoucí).
- ***Odolnost***: Po úspěšném dokončení transakce jsou změny provedené na datech permanentní a přetrvají i v případě selhání systému. [IBM]



Obrázek 4.2: Příklad struktury tabulek a relací (vlastní zpracování).

MariaDB je komunitně vyvíjená a komerčně podporovaná ***větev*** (Fork) relačního databázového systému **MySQL**. Mezi vývojáře této databáze patří i původní vývojáři **MySQL** a přestože se jedná o dva odlišné databázové systémy, jsou spolu vysoce kompatibilní.

MariaDB poskytuje několik typů datových uložišť, které nabízí různé výhody a nevýhody pro specifické požadavky. Příkladem může být Aria, MyISAM nebo InnoDB, který je od verze 10.2 výchozím datovým uložištěm. [Mar]

InnoDB. Je obecné datové uložiště vhodné pro většinu typů dat. Nabízí vysokou efektivitu a spolehlivost. Operace nad daty se řídí výše popsaným ACID modelem s transakčním zpracováním umožňujícím operace jako ***potvrzení*** (Commit), ***stornování*** (Rollback) a ***obnovení z poruchy*** (Crash Recovery). Důležitými vlastnostmi jsou také možnost zamykání jednotlivých záznamů v tabulkách, či podpora primárních a cizích klíčů. Pro optimalizaci ukládání dat a vyhledávání v nich se používá struktura ***B-strom*** (B-tree). [MyS]

B-strom je datová struktura, která uchovává seřazená data. Umožňuje sekvenční přístup k datům, jejich vyhledávání, vkládání nových dat, mazání stávajících, a to vše při zachování seřazení dat. Zobecňuje ***binární vyhledávací strom*** (Binary Search Tree) tím, že umožuje ***uzlům*** (Nodes) mít více než dva ***potomky*** (Children). Každý uzel obsahuje klíče ve vzestupném pořadí. Každý z těchto klíčů odkazuje na další dva potomky. Klíče levého potomka musí být vždy menší než aktuální klíč a klíče pravého potomka musí být vždy větší než aktuální klíč.

V kontextu databázového indexování každý klíč v uzlu B-stromu obsahuje kromě samotného klíče také hodnotu ke klíči přiřazenou. Tato hodnota je odkaz na skutečný záznam v databázi. Klíč a hodnota dohromady tvoří ***datový obsah*** (Payload). [Mad23] V tomto přístupu k datům se MariaDB zásadně liší od výše uvedené **InfluxDB**.

MariaDB využívá dotazovací jazyk ***SQL*** (Structured Query Language).

SQL. Je univerzální jazyk navržený pro správu a manipulaci s relačními databázemi. Byl vyvinut společností IBM v 70. letech 20. století a rychle se stal standardem pro celou řadu aplikací. Od svého vzniku prošel velkým rozvojem, což vedlo k jeho celosvětovému rozšíření. Robustnost **SQL** spočívá v jeho schopnosti efektivně a spolehlivě popsat komplexní dotazy, transakce a běžné úlohy správy dat. Stal se základním jazykem pro mnoho populárních relačních databázových systémů jako **PostgreSQL** nebo **MySQL**, čímž prokázal svou trvalou relevanci v neustále se vyvíjejícím světě správy dat. [Sot]

4.1.3 SQLite

Jedná se o knihovnu, napsanou v jazyce **C**, která implementuje rychlé, samostatné a spolehlivé databázové uložiště typu **SQL**. Je to světově nejpoužívanější databázový stroj, na kterém běží přes jeden bilion aktivních databází. [SQL]

Stejně jako **MariaDB** je **SQLite** zdarma dostupnou databází a nabízí relační databázový systém. Mají však rozdílnou architekturu a vhodnost pro různé úlohy.

Architektura. **SQLite** nepotřebuje pro svou funkcionalitu separátní server a lze ji implementovat přímo v rámci aplikace. To znamená, že samotná aplikace spravuje i databázi. **MariaDB** je postavená na modelu klient-server a potřebuje separátní server na obsluhu požadavků. Data jsou uložena na serveru a aplikace se k nim připojuje.

Škálovatelnost. **SQLite** je vytvořena pro použití na jednom zařízení. Není určena pro rozsáhlé aplikace zpracovávající velké objemy rychle příchozích dat. **MariaDB** naopak umožňuje horizontální škálování, což znamená, že s rostoucími nároky lze přidávat další servery do clusteru a tím zvládat větší provoz a objemy dat.

Dotazovací jazyk. Obě databáze využívají dotazovací jazyk **SQL**. **MariaDB** avšak podporuje větší množinu **SQL** příkazů a funkcí, čímž se stává vhodnější pro úlohy vyžadující provádění složitých dotazů a práci s velkým objemem dat.

Výkon. **SQLite** je optimalizována pro jednoduché úlohy a malé aplikace, které nemají velké vytížení. **MariaDB** je oproti tomu stavěna na výkon a zvládá prostředí s vysokou vytíženosti a komplexními dotazy. [Gee23]

Celé databázové uložiště **SQLite**, včetně tabulek je uloženo v jediném souboru s příponou *.db*. Ten lze uložit na libovolné místo a přesouvat mezi platformami.

4.1.4 Použité verze a nastavení

U všech vybraných databází byly zvoleny aktuálně dostupné oficiální verze systémů (listopad 2023). **InfluxDB** prošla od verze 1 do verze 2 velkou vnitřní implementační změnou, a proto byla pro srovnání testována jak verze starší, tak novější.

Všechny databáze byly ponechány v základním nastavení vytvořeném při instalaci. U **MariaDB** bylo ponecháno výchozí datové uložiště InnoDB.

V **SQLite** a **MariaDB** bylo užito standardizováneho dotazovacího jazyka **SQL**. Při provádění dotazů u **InfluxDB** byl pro verzi 1 použit dotazovací jazyk **InfluxQL** a pro verzi 2 jazyk **Flux**. Seznam použitých verzí:

- **InfluxDB** verze 1.11,
- **InfluxDB** verze 2.7.3,
- **MariaDB** verze 11.1.3,
- **SQLite** verze 3.44.2.

4.1.5 Formát dat

Pro účely testování byly do databáze ukládány pouze základní typy zpráv **Order Add**, **Order Modify** a **Order Delete**. Popis zpráv a jejich atributů lze najít v kapitole 3.3.2. Pro každou databázi bylo nutné data připravit do podporovaného formátu pro vložení:

- **InfluxDB** v1 – JSON,
- **InfluxDB** v2 – anotované CSV,
- **MariaDB** – SQL,
- **SQLite** – SQL.

JSON (JavaScript Object Notation). Je odlehčený formát pro výměnu dat mezi systémy. Je dobře čitelný a programově jednoduše zpracovatelný. Přestože vychází z JavaScriptu, JSON je nezávislý na programovacím jazyce a využívá konvencí společných pro jazyky rodiny **C**. Jeho struktura lze připodobnit k datové struktuře ***slovník*** (Dictionary). Jedná se o kolekci ***párů klíčů a hodnot*** (Key/Value Pair) oddělených čárkou. [JSO]

Zdrojový kód 4.3: Ukázka struktury JSON (vlastní zpracování).

```
{"measurement": "orderAdd", "tags": {"SecurityID": "5315926",
    "Side": "1"}, "time": "2021-01-04T00:14:29.072569371Z", "
   fields": {"TrdRegTSTimeIn": "2021-01-04T00:15:13.709451289
   Z", "DisplayQty": 1, "OrdType": 255, "Price": 225.32}}
```

CSV (Comma-Separated Values). Je souborový formát, který využívá čárky (případně jiné oddělovače) k oddělení jednotlivých hodnot a nové řádky pro rozdělení jednotlivých záznamů. Soubor CSV slouží pro uložení tabulkových dat v textovém formátu. Anotované CSV navíc na prvních řádcích definuje použitý oddělovač, názvy sloupců a jejich datové typy.

Zdrojový kód 4.4: Ukázka struktury anotovaného CSV (vlastní zpracování).

```
sep=;
#datatype measurement;string;tag;long;tag;double
m;TrdRegTSTimeIn;SecurityID;DisplayQty;Side;Price
orderAdd;2021-01-04T00:15:13.709451289Z;5315926;1;1;225.32
```

SQL (Structured Query Language). Jak již bylo zmíněno výše, jedná se o univerzální jazyk pro správu a manipulaci s daty. Pro vkládání dat do databáze se používají příkazy ***DML*** (Data Manipulation Language).

Zdrojový kód 4.5: Ukázka struktury **SQL** příkazu pro vložení dat (vlastní zpracování).

```
INSERT INTO orderAdd(TrdRegTSTimeIn, SecurityID,
   TrdRegTSTimePriority, DisplayQty, Side, Price)
VALUES('2021-01-04␣00:15:13.709451289',5315926,'2021-01-04␣
   00:14:29.072569371',1,1,225.32);
```

Pro každý typ zprávy bylo v **InfluxDB** vytvořeno vlastní ***měření*** (Measurement) (orderAdd, orderModify, orderDelete). Jako ***časová značka*** (Timestamp) byl použit parametr *TrdRegTSTimePriority* (*TrdRegTSPrevTimePriority* u Order Modify). Jako ***štítky*** (Tag-set) byly zvoleny atributy *SecurityID, Side*. Všechny ostatní atributy zpráv (specifické podle typu zprávy) představují ***pole*** (Field-set).

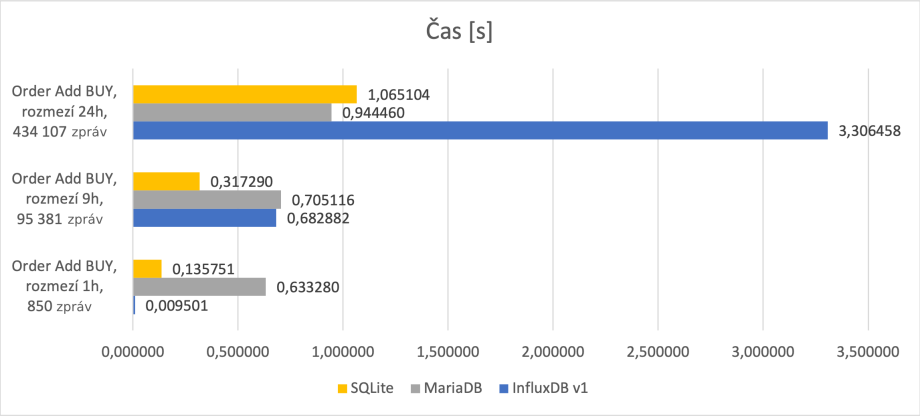
V **MariaDB** a **SQLite** byla pro každý typ zprávy vytvořena vlastní tabulka obsahující všechny atributy daného typu zprávy.

Rozdíl byl také v datových typech použitých pro uložení jednotlivých atributů. U **InfluxDB** jdou štítky uložit pouze jako typ ***řetězec*** (String), u zbylých databází byly atributy uloženy jako typ ***celé číslo*** (Integer). **InfluxDB** verze 2 umožňuje ukládat časovou značku s přesností na nanosekundy, u verze 1 a u **MariaDB** byla maximální možná přesnost mikrosekundy. **SQLite** nedefinuje formát pro uložení času a tak byly časové atributy uloženy jako řetězec.

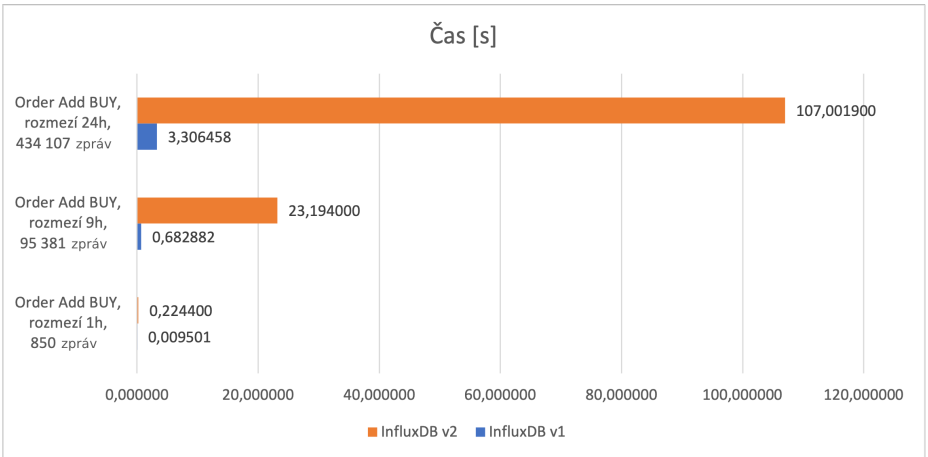
4.1.6 Průběh testu a porovnání databází

Pro účely této práce byla nejdůležitějším požadavkem na databázi rychlost vyhledání dat podle zadaných parametrů. Databáze musí bý schopná uložit velké objemy zpráv a umožnit rychlé nalezení požadované množiny pro rekonstrukci knihy objednávek.

Testování bylo prováděno pomocí skriptů v jazyce **Python** s využitím nativních knihoven pro komunikaci s jednotlivými databázemi. Skript měřil čas od odeslání požadavku do získání výsledných dat z databáze. Prováděno bylo několik dotazů simulující různé velikosti výsledné množiny dat. Na obrázcích 4.3 a 4.4 lze vidět výsledky testů pro požadavky, které se dotazovali na zprávy typu Order Add s nabídkovými objednávkami, za různé časové období.



Obrázek 4.3: Rychlost získání dat v **SQLite**, **MariaDB** a **InfluxDB** v1 (vlastní zpracování).



Obrázek 4.4: Rychlost získání dat v **InfluxDB** v1 vs v2 (vlastní zpracování).

Z výsledků testování na obrázcích 4.3 a 4.4 je vidět, že pro malou množinu dat je nejrychlejší **InfluxDB** v1, dokonce více než 10× rychlejší než ostatní databáze. Při

zvětšování výsledné množiny zpráv lze ale vidět exponenciální prodlužování času u obou verzí. Verze 2 má čas vůbec nejhorší, při dotazu na největší množinu dat dokonce více než 100× horší než **MariaDB** či **SQLite**. Nejmenší odchylky v časech přes všechny testy měla **MariaDB**, která byla nejrychlejší pro největší množinu dat. **SQLite**, přestože se jedná o odlehčenou databázi bez vlastního serveru, si držela nejlepší čas pro středně velkou množinu a velmi podobný čas jako **MariaDB** pro největší množinu.

4.1.7 Navrhované uložení dat

V rámci této části návrhu řešení bylo provedeno důkladné testování vybraných databází s cílem najít optimální řešení pro ukládání a práci s daty. Bohužel žádná z testovaných databází plně nesplnila požadované parametry.

Některé databáze sice splňovaly rychlosti přístupu k datům, ale nepodporovaly požadované datové formáty pro uložení některých parametrů zpráv. Jiné naopak nabízely požadované datové formáty, ale jejich rychlostní parametry nedosahovaly požadované úrovně.

Z těchto důvodů bylo nakonec rozhodnuto, že v rámci této bakalářské práce budou data ukládána ve formátu souborů CSV. Formát CSV sice nenabízí stejnou úroveň funkcionality jako databázové systémy, ale umožňuje flexibilitu a jednoduchost použití potřebnou pro navazující práci s daty.

4.2 Předzpracování dat a rekonstrukce

Tato část návrhu řešení popisuje proces předzpracování dat do vhodného formátu a následnou rekonstrukci knihy objednávek.

Sloučení dat. Pro efektivní rekonstrukci knihy objednávek bylo potřeba data sloučit do jednoho souboru, který by obsahoval veškeré dění na trhu pro konkrétní instrument za určitou časovou jednotku. Burzovní data získaná z **T7 EOBI** byla uložena v souborech CSV, kde každý soubor představoval jeden typ zprávy pro jeden konkrétní instrument za jeden obchodní den. Prvním krokem tedy bylo vybrat typy zpráv relevantní pro rekonstrukci knihy objednávek a sloučit dané soubory CSV ve správném pořadí do jednoho a vytvořit tak soubor CSV se zprávami pro daný instrument a daný den. Popis relevantních zpráv a jejich atributů lze najít v kapitole 3.3.2

Pro sloučení bylo potřeba identifikovat parametry, které určují, jak byly jednotlivé zprávy vykonávány za sebou. Jelikož každý typ zprávy má jiné parametry, bylo nutné vytipovat všechny možné kombinace kandidátů a společně s informacemi z oficiální dokumentace **T7 EOBI** určit výslednou množinu. Kromě parametrů popsaných u jednotlivých zpráv měl každý typ zprávy ještě dva společné parametry.

PARENT_ID a *ID*. Tyto parametry se společně s parametrem *TrdRegTSTimePriority* ukázaly jako klíčové pro správné seřazení. *PARENT_ID* určuje číslo UDP datagramu, ve kterém zpráva přišla a *ID* určuje pořadí daného typu zprávy od začátku dne. Korektnost seřazení byla ověřena závěrečnou kontrolou samotné správnosti rekonstrukce knihy objednávek, která je popsána dále v tomto dokumentu.

Úprava zprávy Order Modify. Jak bylo popsáno výše, tato zpráva informuje o změně aktivní objednávky, která mění její časovou prioritu. Zpráva obsahuje parametry původní objednávky a zároveň nové hodnoty, které se mají objednávce přiřadit. Jedním z cílů předzpracování dat bylo co nejvíce sjednotit formát parametrů a zjednodušit jednotlivé zprávy. Rozdílnost struktury parametrů u této zprávy by to značně ztížila. Z tohoto důvodu byla zpráva Order Modify nahrazena dvěma „fiktivními" zprávami. První Order Delete, která dostala parametry původní objednávky, a tím došlo k jejímu smazání, a druhá Order Add, která dostala parametry nové zprávy, a tím došlo k vytvoření nové aktivní objednávky.

Transformace časových značek. Všechny časové parametry zpráv v původních CSV souborech byly vyjádřeny jako nanosekundový ***UNIX čas*** (počet nanosekund od 1. ledna 1970 00:00:00 UTC). Jelikož každý soubor odpovídá jednomu dni, byly časové značky pro ušetření místa a zjednodušení zpracování transformovány na počet nanosekund od začátku daného dne. To mimo jiné znamená, že některé časové značky mohou po transformaci být záporné (původní časová značka odpovídala předchozím dnům).

Transformace množství a strany objednávky. Další úpravou dat byla transformace množství a strany. V původních souborech bylo ***množství*** (Quantity) vyjádřeno ve speciálním datovém formátu definovaným **T7 EOBI**, a ***strana*** (Side) vyjádřena jako číslo. Bylo potřeba tedy hodnoty opět modifikovat do více praktického a přehledného formátu. Transformace množství například vypadala následovně: 0.00090000 $\longrightarrow$ 9. Transformace strany pak následovně: $1 \longrightarrow B$, $2 \longrightarrow S$, kde B je nabídková objednávka a S poptávková.

Posledním krokem této části předzpracování bylo sjednotit parametry ze všech typů zpráv a do výsledného souboru je systematicky uložit. Aplikováním všech výše popsaných úprav dat vznikl výsledný sloučený soubor CSV, který obsahoval veškeré zprávy seřazené ve správném pořadí a s jednotnou strukturou parametrů.

Soubor obsahuje následující parametry: *i, PARENT_ID, ID, TrdRegTSTimeIn, TrdRegTSTimePriority, Side, Price, DisplayQty, op, Trans, Prio*, kde:

- *i*: pořadí zprávy v souboru,
- *TrdRegTSTimeIn, TrdRegTSTimePriority, Trans*: transformované časové značky,
- *Side, DisplayQty*: transformovaná strana a množství objednávky,
- *op*: identifikátor typu zprávy,
- *Prio*: původní prioritní UNIX časová značka zprávy.

Druhou částí předzpracování dat bylo z připraveného sloučeného souboru vytvořit nový soubor, ze kterého se již bude kniha objednávek rekonstruovat. Tento soubor už obsahuje pouze data nutná pro rekonstrukci a parametry v něm jsou upraveny tak, aby rekonstrukce byla co nejrychlejší.

Tento proces funguje následujícím způsobem: Pro každou ***cenu*** (Price) na straně ***nabídky*** (Bid) i ***poptávky*** (Ask) je dáno kumulativní ***množství*** (Quantity) obchodovaného instrumentu v daném čase. Zároveň jsou pro danou cenu stanoveny hodnoty *from* a *to*, které udávají, jak dlouho je tento stav zachován (jinými slovy, od kolikátého a do kolikátého řádku v souboru platí dané množství pro danou cenu). Takto vytvořený soubor pak obsahuje následující parametry: *i, Price, DisplayQty, Q, from, to, Trans, Prio*, kde *DisplayQty* je kumulativní množství a *Q* je absolutní změna množství. *Trans* je časová značka a udává čas, ke kterému se dá kniha objednávek zrekonstruovat.

Rekonstrukce knihy objednávek. Z takto předpřipravených dat lze již knihu objednávek efektivně zrekonstruovat. Každý řádek ve finálním souboru CSV představuje určitou změnu, ke které v knize objednávek došlo (modifikace ceny, množství, přidání nové objednávky atd.). Zároveň ale řádek představuje bod, ke kterému lze knihu zrekonstruovat. Každý bod je tedy definován ***časem*** (*Trans*), ***číslem řádku*** (*from*) a změnou, ke které v daném bodě došlo (*Price, DisplayQty, Q*). Je-li zadán čas, ke kterému má být zobrazen stav knihy objednávek, najde se v souboru bod s nejbližším časem k zadanému času. Číslo řádku tohoto bodu pak představuje horní hranici času pro rekonstrukci. V souboru jsou nalezeny všechny řádky, které jsou platné pro danou hranici (neboli všechny řádky, jejichž hodnota *from* je menší než daná hranice a zároveň hodnota *to* je větší než daná hranice). Výsledná množina řádků představuje stav knihy objednávek v zadaném čase a obsahuje všechny ceny na straně nabídky i poptávky a kumulativní množství obchodovaného instrumentu pro danou cenu.

Implementace 5

Tato kapitola popisuje implementaci navrženého systému. Implementovaný systém je rozdělen do tří samostatných programů. Všechny byly napsány v jazyce **Python** verze 3.8.8.

Jazyk **Python** byl zvolen z několika důvodů. Jedná se o univerzální programovací jazyk, který je vhodný pro širokou škálu úloh. V posledních letech se stává oblíbeným zejména pro úlohy analýzy a zpracování velkých dat a jejich vizualizaci. Podporuje integraci velkého množství knihoven a nástrojů pro práci s daty. Knihovny jako **NumPy** nebo **Pandas** usnadňují manipulaci s velkými objemy dat a provádění komplexních analýz. **Python** je také multiplatformní jazyk, což znamená, že jej lze spouštět na různých procesorech a operačních systémech. To usnadňuje vývoj a nasazení aplikací. Výhodou je také čitelnost a stručnost kódu napsaného v jazyce **Python**, což usnadňuje pochopení a údržbu programů v něm napsaných.

`merge.py`. Tento program se stará o sloučení souborů CSV s jednotlivými typy zpráv. Modifikuje je do navrženého formátu, transformuje a sjednocuje jejich parametry. Výstupem je soubor CSV obsahující veškeré vykonané zprávy jdoucí ve správném pořadí a s jednotnou strukturou parametrů.

`preprocess.py`. Tento program na vstupu přijímá soubor vytvořený programem `merge.py`. Ze sjednocených zpráv vybere pouze nutné pro rekonstrukci a jejich parametry dále zpracuje. Výstupem je soubor CSV, ze kterého lze již knihu objednávek zrekonstruovat.

`reconstruction.py`. Tento program spouští server, který zajišťuje provoz webové vizualizace knihy objednávek, a na kterém je zároveň implementované ***API*** (Application Programming Interface) pro vzdálený přístup ke stavu knihy objednávek. Jako vstup pro rekonstrukci používá soubory CSV vytvořené programem `preprocess.py`.

5.1 Použité knihovny

K vypracování zmíněných programů bylo použito několik knihoven pro **Python**: Nejdůležitější z nich budou dále stručně popsány.

NumPy. Jedná se o zdarma dostupnou knihovnu pro jazyk **Python**, která najde uplatnění téměř ve všech vědeckých a technických úlohách. Představuje univerzální standard pro práci s numerickými daty v **Pythonu**. Na této základní knihovně jsou postaveny i jiné oblíbené knihovny pro **Python** jako **Pandas**, **SciPy**, **Matplotlib**, **scikit-learn**, **scikit-image** a jiné, které používají API knihovny **NumPy** pro datovou analýzu.

NumPy rozšiřuje **Python** o různé datové struktury jako například vícerozměrné matice a pole. Umožňuje také nad těmito strukturami provádět širokou škálu matematických operací. Tyto mocné datové struktury a vysokoúrovňové matematické funkce umožňují efektivní výpočty a práci s numerickými daty.

Oficiální dokumentace knihovny: [Num]

Pandas. Je zdarma dostupnou knihovnou pro jazyk **Python**, která poskytuje rychlé a flexibilní datové struktury navržené tak, aby práce se strukturovanými daty (například soubory CSV či tabulkami SQL) byla snadná a intuitivní. Jejím cílem je stát se základním stavebním kamenem pro analýzu a manipulaci s daty v **Pythonu**. Je postavená na knihovně **NumPy**, takže používá její datové struktury a funkce. **Pandas** je často používána pro předzpracování dat pro následné vizualizace, například pomocí knihovny **Matplotlib**, nebo pro další datové a statistické analýzy (knihovna **SciPy**).

Oficiální dokumentace knihovny: [Pan]

Flask. Je zdarma dostupný mikroframework pro tvorbu webových aplikací v jazyce **Python**. Mikroframework je to z důvodu, že jádro aplikace **Flask** je velmi odlehčené a lehce rozšiřitelné samotným vývojářem podle jeho specifických potřeb. **Flask** například nemá vlastní ***ORM*** (Object Relational Mapper) pro práci s databázemi. Díky své jednoduchosti a modularitě je **Flask** vhodný pro široké spektrum webových aplikací. Populární je také pro implementaci API.

Oficiální dokumentace mikroframeworku: [Fla]

Dash. Je zdarma dostupný framework pro jazyk **Python** určený k tvorbě interaktivních webových aplikací. Je postavený na mikroframeworku **Flask**, **Plotly.js** (JavaScriptová knihovna pro vizualizaci dat) a **React.js** (JavaScriptová knihovna pro tvorbu uživatelských rozhraní). Aplikace **Dash** se skládají ze dvou hlavních částí. První je ***rozvržení*** (Layout), které definuje jak bude aplikace vypadat. Druhou je ***obsluha*** (Callback), které zajišťuje vykonávání funkcí aplikace a interakci s uživatelem.

Oficiální dokumentace frameworku: [Plo]

5.2 Sloučení zpráv

Algoritmus sloučení zpráv je implementován v souboru `merge.py`. Program pro své fungování požaduje konfiguraci několika parametrů, které se všechny nastavují v hlavní funkci `main`. Výstupem programu je soubor CSV se sloučenými zprávami, které jsou seřazené podle pravidla popsaného níže.

Konfigurační parametry jsou:

- `delim`: oddělovací znak vstupních souborů CSV;
- `path`: cesta ke vstupním souborům CSV se zprávami;
- `output_format_params`: specifikace názvů sloupců výstupního souboru;
- `qty_rounder`: koeficient transformace množství v objednávkách;
- `data_files`: názvy vstupních souborů CSV a typ jednotlivých zpráv;
- `outfile`: název výstupního souboru.

V konfigurační části hlavní funkce `main` jsou také parametry `date`, `instrument`, `security` a `base_date`, které se automaticky tvoří na základě názvu prvního vstupního souboru definovaného v parametru `data_files`. Pro správné vytvoření je potřeba dodržet formát pojmenování vstupních souborů. Ten musí mít minimálně následující formu: `..._RRRRMMDD.typ_instrument_security.csv`, například tedy: `2350_00_D_03_A_20191202.OrderAdd_FGBL_4128839.csv`.

Formát vstupních souborů. Program také předpokládá, že veškeré vstupní soubory jsou vzestupně seřazené podle hodnot ve sloupcích *PARENT_ID* a *ID*, a pro jednotlivé typy zpráv očekává výskyt následujících sloupců:

- **Order Add**: *PARENT_ID, ID, TrdRegTSTimeIn, Side, Price, DisplayQty, TrdRegTSTimePriority*;
- **Order Modify**: *PARENT_ID, ID, TrdRegTSTimeIn, Side, PrevPrice, Price, PrevDisplayQty, DisplayQty, TrdRegTSPrevTimePriority, TrdRegTSTimePriority*;
- **Order Modify Same Priority**: *PARENT_ID, ID, TrdRegTSTimeIn, Side, Price, PrevDisplayQty, DisplayQty, TransactTime, TrdRegTSTimePriority*;
- **Order Delete**: *PARENT_ID, ID, TrdRegTSTimeIn, Side, Price, DisplayQty, TransactTime, TrdRegTSTimePriority*;

- **Execution Summary**: *PARENT_ID, ID, RequestTime, AggressorSide, LastPx, LastQty, ExecID*;
- **Partial Order Execution**: *PARENT_ID, ID, Side, Price, LastQty, TrdRegTSTimePriority*;
- **Full Order Execution**: *PARENT_ID, ID, Side, Price, LastQty, TrdRegTSTimePriority*.

Algoritmus nejprve pro všechny řádky ve všech souborech zavádí třídu `Tagged_line`, která definuje daný řádek jako dvojici `line` a `source_op` (řádek a typ zprávy). Vytvořené instance této třídy reprezentující jednotlivé řádky jsou uloženy v poli `merge_list`. Instance nejsou generovány a ukládány do paměti najednou, ale pomocí příkazu `yield` jsou vytvořeny až v momentě, kdy je jich potřeba pro seřazení.

Zdrojový kód 5.1: Třída `Tagged_line` v souboru `merge.py` (vlastní kód).

```
class Tagged_line:
    def __init__(self, line, source_op):
        self.line = line
        self.source_op = source_op
```

Zdrojový kód 5.2: Funkce generující instance třídy `Tagged_line` v souboru `merge.py` (vlastní kód).

```
def gen_tagged_lines(file, source_op, delim):
    reader = csv.DictReader(file, delimiter=delim)
    for line in reader:
        yield Tagged_line(line, source_op)
```

Pro samotné sloučení a správné seřazení jednotlivých zpráv ze vstupních souborů je využito Pythonského modulu `heapq` a jeho algoritmu seřazení na haldě `heapq.merge` [Pyt]. Vstupem je pole `merge_list` a pravidlo pro seřazení je následující: Řádky jsou seřazeny vzestupně podle hodnot *PARENT_ID* a *ID* v případě všech typů zpráv kromě zprávy **Partial Order Execution**, ta je zařazena podle hodnot *PARENT_ID* a *TrdRegTSTimePriority*. Výstupem funkce je pak iterátor `sorted_lines` se seřazenými zprávami.

Zdrojový kód 5.3: Algoritmus seřazení na haldě v souboru `merge.py` (vlastní kód).

```
sorted_lines = heapq.merge(*merge_list,key=lambda x: (int(x.
    line["PARENT_ID"]), int(x.line["ID"])) if x.source_op != "
    PE" else (int(x.line["PARENT_ID"]), get_nansec_from_time(
    base_date,x.line["TrdRegTSTimePriority"])))
```

Program poté přechází do druhé části, ve které modifikuje určité hodnoty zpráv (typy modifikací popsány v 4.2) a zapisuje seřazené zprávy do výstupního souboru CSV.

Sloupce ve výstupním souboru jsou dány parametrem `output_format_params`, který obsahuje hodnoty *i, PARENT_ID, ID, TrdRegTSTimeIn, TrdRegTSTimePriority, Side, Price, DisplayQty, op, Trans, Prio*, kde pro jednotlivé zprávy jsou tyto hodnoty tvořeny podle následujících tabulek.

Tabulka 5.1: Tvoření parametrů výstupního souboru ze vstupních parametrů zprávy **Order Add** (respektive rozložené zprávy **Order Modify**)

parametr	**hodnota**
i	Pořadí zprávy ve výstupním souboru.
PARENT_ID	*PARENT_ID*
ID	*ID*
TrdRegTSTimeIn	Transformovaná časová značka *TrdRegTSTimeIn* na počet nanosekund od začátku dne.
TrdRegTSTimePriority	Transformovaná časová značka *TrdRegTSTimePriority* na počet nanosekund od začátku dne.
Side	B pokud $Side = 1$, S pokud $Side = 2$.
Price	*Price*
DisplayQty	*DisplayQty* vynásobené `qty_rounder` převedené na celé číslo.
op	A (respektive XXX)
Trans	Transformovaná časová značka *TrdRegTSTimePriority* na počet nanosekund od začátku dne. Pokud jde o počáteční zprávy dne **Order Add** (popsáno níže), pak je hodnota transformovaná časová značka *TrdRegTSTimeIn* na počet nanosekund od začátku dne minus jedna.
Prio	*TrdRegTSTimePriority*

Tabulka 5.2: Tvoření parametrů výstupního souboru ze vstupních parametrů zprávy **Order Delete** (respektive rozložené zprávy **Order Modify**)

parametr	**hodnota**
i	Pořadí zprávy ve výstupním souboru.
PARENT_ID	*PARENT_ID*
ID	*ID*
TrdRegTSTimeIn	Transformovaná časová značka *TrdRegTSTimeIn* na počet nanosekund od začátku dne.
TrdRegTSTimePriority	Transformovaná časová značka *TrdRegTSTimePriority* (respektive *TrdRegTSPrevTimePriority*) na počet nanosekund od začátku dne.
Side	B pokud $Side = 1$, S pokud $Side = 2$.
Price	*Price* (respektive *PrevPrice*)
DisplayQty	*DisplayQty* (respektive *PrevDisplayQty*) vynásobené záporným `qty_rounder` převedené na celé číslo.
op	D (respektive YYY)
Trans	Transformovaná časová značka *TransactTime* (respektive *TrdRegTSTimePriority*) na počet nanosekund od začátku dne.
Prio	*TrdRegTSTimePriority* (respektive *TrdRegTSPrevTimePriority*)

Tabulka 5.3: Tvoření parametrů výstupního souboru ze vstupních parametrů zprávy **Order Modify Same Priority**

parametr	**hodnota**
i	Pořadí zprávy ve výstupním souboru.
PARENT_ID	*PARENT_ID*

(tabulka pokračuje na další stránce)

Tabulka 5.3 *(pokračování z předchozí stránky)*

parametr	**hodnota**
ID	*ID*
TrdRegTSTimeIn	Transformovaná časová značka *TrdRegTSTimeIn* na počet nanosekund od začátku dne.
TrdRegTSTimePriority	Transformovaná časová značka *TrdRegTSTimePriority* na počet nanosekund od začátku dne.
Side	B pokud *Side* = 1, S pokud *Side* = 2.
Price	*Price*
DisplayQty	Rozdíl *DisplayQty* a *PrevDisplayQty* vynásobených `qty_rounder` převedené na celé číslo.
op	MS
Trans	Transformovaná časová značka *TransactTime* na počet nanosekund od začátku dne.
Prio	*TrdRegTSTimePriority*

Tabulka 5.4: Tvoření parametrů výstupního souboru ze vstupních parametrů zprávy **Partial Order Execution** (respektive **Full Order Execution**)

parametr	**hodnota**
i	Pořadí zprávy ve výstupním souboru.
PARENT_ID	*PARENT_ID*
ID	*ID*
TrdRegTSTimeIn	NOVALUE
TrdRegTSTimePriority	Transformovaná časová značka *TrdRegTSTimePriority* na počet nanosekund od začátku dne.
Side	B pokud *Side* = 1, S pokud *Side* = 2.
Price	*Price*

(tabulka pokračuje na další stránce)

Tabulka 5.4 *(pokračování z předchozí stránky)*

parametr	**hodnota**
DisplayQty	*LastQty* vynásobené záporným `qty_rounder` převedené na celé číslo.
op	PE (respektive E)
Trans	`INT_MIN` (−9223372036854775808)
Prio	*TrdRegTSTimePriority*

Tabulka 5.5: Tvoření parametrů výstupního souboru ze vstupních parametrů zprávy **Execution Summary**

parametr	**hodnota**
i	Pořadí zprávy ve výstupním souboru.
PARENT_ID	*PARENT_ID*
ID	*ID*
TrdRegTSTimeIn	Transformovaná časová značka *RequestTime* na počet nanosekund od začátku dne.
TrdRegTSTimePriority	NOVALUE
Side	B pokud *AggressorSide* = 1, S pokud *AggressorSide* = 2.
Price	*LastPx*
DisplayQty	*LastQty* vynásobené `qty_rounder` převedené na celé číslo.
op	Sum
Trans	Transformovaná časová značka *ExecID* na počet nanosekund od začátku dne.
Prio	NOVALUE

Jako první se do výstupního souboru zapisují zprávy informující o stavu knihy objednávek z předchozího dne. Soubor se zprávami **Order Add** jako prvních několik

záznamů totiž obsahuje zprávy popisující všechny aktivní objednávky, jejich ceny a množství v čase vypnutí burzovního systému **T7 EOBI** předchozí den. Tyto zprávy lze poznat podle hodnoty ve sloupci *TrdRegTSTimeIn*, která je ve všech zprávách stejná. Program tyto úvodní zprávy řeší separátně, jelikož je pro ně potřeba modifikovat hodnotu parametru *Trans* jinak, než ve zbylých zprávách stejného typu (viz tabulka 5.1). Hodnota byla takto zvolena z důvodů následné rekonstrukce. Tyto zprávy mají prioritní časové značky z předchozích dnů, a tak jsou po transformaci záporné. Pro rekonstrukci bylo potřeba vytvořit kladnou časovou značku, která by odpovídala pomyslnému prvnímu stavu dne, kdy došlo pouze k načtení stavu knihy objednávek z předchozího dne. Proto se použila hodnota sloupce *TrdRegTSTimeIn* minus jedna.

Jakmile dojde k zapsání těchto zpráv, pokračuje program v zapisování a modifikaci (podle tabulek 5.1, 5.2, 5.3, 5.4, 5.5) následujících zpráv obsažených v iterátoru `sorted_lines`.

5.3 Předzpracování dat

Algoritmus předzpracování dat je implementován v souboru `preprocess.py`. Program pro své fungování požaduje konfiguraci několika parametrů, které se nastavují přímo v souboru. Program přijímá jako vstup soubor CSV vytvořený programem `merge.py`. Výstupem je soubor CSV s předzpracovanými daty pro finální program rekonstrukce.

Konfigurační parametry jsou:

- `delim`: oddělovací znak vstupního souboru;
- `infile`: název vstupního souboru;
- `incols`: názvy sloupců vstupního souboru;
- `outfile`: název výstupního souboru;
- `outcols`: specifikace názvů sloupců výstupního souboru;
- `ops_to_include`: typy zpráv, které se mají použít pro předzpracování.

Program načte vstupní soubor a jako první z něj vybere pouze řádky, na kterých jsou vybrané typy zpráv. Přepočítá hodnotu sloupce *i*, aby odpovídala novému počtu řádků. U zpráv, kde hodnota parametru *Side* = S, jsou hodnoty parametru *Price* vynásobeny koeficientem −1. Důvodem je jednoduché rozlišení nabídkových (kladné hodnoty *Price*) a poptávkových (záporné hodnoty *Price*) objednávek ve výstupním souboru.

Jako další jsou zavedeny nové sloupce *from* a *to*, které udávají v jakém rozmezí řádků v souboru platí dané množství pro danou cenu. Také je zaveden sloupec *Q*, do kterého program uloží původní hodnotu *DisplayQty*. Program poté vykoná cyklus, ve kterém pro každou cenu (hodnotu v parametru *Price*) vypočítá kumulativní množství na každém řádku s danou cenou a toto množství uloží do parametru *DisplayQty*. Zároveň napočítá hodnoty *from* a *to*, kde hodnota *from* je číslo daného řádku a hodnota *to* je číslo následujícího řádku, na kterém došlo ke změně pro stejnou cenu.

Zdrojový kód 5.4: Hlavní cyklus předzpracování v souboru `preprocess.py` (vlastní kód).

```
1 for i in df["Price"].unique():
2     pom = df.index[df["Price"] == i]
3     df.loc[pom, "DisplayQty"] = np.cumsum(df.loc[pom, "
    DisplayQty"])
4     df.loc[pom[:-1], "to"] = df.loc[pom[1:], "i"].values
```

Takto modifikované hodnoty uloží do nového souboru CSV specifikovaného v parametru `outfile`.

5.4 Rekonstrukce knihy objednávek

Algoritmus rekonstrukce knihy objednávek společně se serverem pro spuštění webové vizualizace a API jsou implementovány v souboru `reconstruction.py`. Program používá soubory vytvořené programem `preprocess.py`.

Konfigurace programu se nachází ve třídě `Config`. Definuje následující proměnné:

- `addr`: adresa serveru, na které se má spustit webová vizualizace a API;
- `port`: port serveru;
- `path`: cesta ke vstupním souborům;
- `df_cols`: názvy sloupců vstupních souborů, se kterými má program pracovat;
- `df_cols _type`: datové typy sloupců definovaných v `df_cols`;
- `delim`: oddělovač vstupních souborů CSV.

Třída `Tools` implementuje pomocné funkce použité v programu pro převod časových značek. Funkce `calc_time_from_nansec` převádí časovou značku (vyjádřenou jako počet nanosekund od začátku dne definovaného souborem CSV) na řetězec formátu datum a čas. Funkce `calc_nansec_from_time` provádí převod obráceně.

Třída `OB` reprezentuje knihu objednávek. Program používá celkem dvě instance této třídy, jednu pro webovou vizualizaci a druhou pro API. Pro vytvoření instance jsou potřeba tři parametry: `instrument`, `security` a `date`. Třída definuje parametry popisující a uchovávající aktuální stav knihy objednávek v dané instanci. Parametry jsou:

- `__instrument`: název instrumentu, pro který se aktuálně kniha objednávek tvoří;
- `__security`: identifikační číslo produktu v daném instrumentu, pro který se aktuálně kniha objednávek tvoří;
- `__date`: datum, pro které se aktuálně kniha objednávek tvoří;
- `__data`: datový rámec aktuálně načteného souboru CSV s daty EOBI;
- `__min_timestamp`: nejmenší časová značka ve sloupci *Trans* v aktuálně načteném souboru CSV;
- `__timestamp`: časová značka, ke které je kniha objednávek aktuálně zrekonstruovaná;
- `__bookA`: datový rámec reprezentující poptávkové objednávky v aktuálně zrekonstruované knize objednávek;
- `__bookB`: datový rámec reprezentující nabídkové objednávky v aktuálně zrekonstruované knize objednávek;
- `__executes`: provedené obchody, ke kterým došlo v čase, pro který je kniha objednávek aktuálně zrekonstruovaná;
- `__changed`: pravdivostní hodnota informující o tom, zda došlo k úspěšné změně zdrojového souboru CSV na základě uživatelem zadaných parametrů.

Program po spuštění zavolá hlavní funkci `main`. Ta jako první vytvoří instanci třídy `OB` s výchozími hodnotami `instrument` = *"FGBL"*, `security` = "4128839" a `date` = "20191202". Program tedy předpokládá, že v adresáři daném proměnnou `path` se nachází minimálně datový soubor pro tyto zadané hodnoty. Instance je poté uložena do proměnné `dash_OB` a reprezentuje knihu objednávek na bázi webové aplikace **Dash**. Program dále deklaruje proměnnou `api_OB`, která bude reprezentovat knihu objednávek pro API, ale bude inicializována až s prvním příchozím požadavkem.

Jako další se vytvoří instance třídy `Flask` reprezentující server a uloží se do proměnné `server`. Poté se vytvoří instance třídy `Dash` a jako parametr se jí předá

vytvořený `server`. Tato instance se uloží do proměnné `app` a představuje webovou aplikaci pro vizualizaci, která bude dostupná na vytvořeném serveru.

Před spuštěním vytvořeného serveru je potřeba vytvořit vzhled webové aplikace a definovat funkce pro obsluhu této aplikace a také požadavků přicházejících na API. Vzhled a rozvržení je nastavené v parametru aplikace `app.layout`. Funkce `update_orderbook` s návěštím `@app.callback` je funkce, která se stará o obsluhu webové aplikace a je volána v případě jakékoliv změny provedené uživatelem. Funkce `api_get_data` s návěštím `@server.route` je funkce, která se stará o obsluhu požadavků příchozích na API. Po definování těchto funkcí již program spouští server na adrese `addr` a portu `port`.

Algoritmus rekonstrukce knihy objednávek. Knihu objednávek lze zrekonstruovat k jakékoliv (kladné) časové značce ve sloupci *Trans* ve zdrojovém souboru CSV, který program zrovna používá. Pro připomenutí, řádky těchto souborů představují určitou změnu (množství, ceny, vykonání objednávky), ke které došlo v knize objednávek v čase daném ve sloupci *Trans*. Časové značky v souboru jsou vyjádřeny celočíselným datovým typem jako počet nanosekund od začátku daného obchodního dne, který popisuje zdrojový soubor. Uživatel má možnost zadat čas (ve formátu `hh:mm:ss.nnnnnnnnn` nebo `hh:mm:ss`), ke kterému chce knihu objednávek buď zobrazit (ve webové aplikaci), nebo zaslat stav (požadavkem na API).

Samotný algoritmus rekonstrukce je popsán v kapitole 4.2 a je implementován v metodě `calc_order_book_state` třídy `OB`.

Metoda `calc_order_book_state` se tedy stará o rekonstrukci knihy objednávek. Nepřejímá ale jako vstupní parametr pouze číslo řádku, ke kterému má knihu rekonstruovat, nýbrž číselný interval řádků.

Pro (uživatelem) zadaný čas $\boldsymbol{t}$ najde program nejprve interval řádků v souboru, který se pro rekonstrukci použije. Tento interval je tvořen dvěma čísly $\boldsymbol{(a; b)}$, kde $\boldsymbol{b}$ je číslo řádku, ke kterému se kniha objednávek zrekonstruuje. Číslo $\boldsymbol{a}$ je opět číslem řádku a jeho určení závisí na několika faktorech. Pro lepší pochopení je nejprve potřeba vysvětlit, proč se systém intervalů používá.

Ve zdrojovém souboru se vyskytují řádky se záporným číslem ve sloupci *Trans*. Tyto řádky reprezentují vykonání obchodu (zprávy Partial Order Execution a Full Order Execution). Ve sloupci *Trans* mají záporné číslo, jelikož tento typ zpráv (popsaný v 3.3.2.4) neobsahuje časovou značku udávající čas vykonání obchodu a tak byla hodnota nahrazena definovaným záporným číslem (viz tabulka 5.4). Informace o vykonaných obchodech je ale potřeba zobrazit ve webové aplikaci, či zaslat odpovědí na požadavek API. Program tedy tyto vykonané obchody sdružuje s první následující kladnou časovou značkou ve sloupci *Trans*, ke které lze knihu objednávek zrekonstruovat. Pokud tedy algoritmus najde číslo řádku $\boldsymbol{b}$, ke kterému knihu

zrekonstruuje, musí také zjistit, zda nejsou na předcházejících řádcích tyto vykonané obchody, a případně je zahrnout do následného zpracování a vizualizace. Pro ohraničení těchto všech řádků, které se mají použít pro zpracování se používá právě první číslo intervalu ***a***. To určuje číslo řádku, na kterém je první odlišná kladná časová značka, a od kterého následují řádky potřebné pro zpracování. Do zpracování se nepočítá samotný řádek s číslem ***a***.

Na obrázku 5.1 lze vidět reálné situace, které mohou v souboru nastat (pro zjednodušení byla struktura souboru zkrácena).

1

i	Price	Q	Trans
1	p1	q1	t1
2	p2	q2	t2
3	p3	q3	t3
4	p4	q4	t4
5	p5	q5	t5
6	p6	q6	t6
7	p7	q7	t7

2

i	Price	Q	Trans
1	p1	q1	t1
2	p2	q2	t2
3	p3	q3	INT_MIN
4	p4	q4	INT_MIN
5	p5	q5	INT_MIN
6	p6	q6	t3
7	p7	q7	t4

3

i	Price	Q	Trans
1	p1	q1	t1
2	p2	q2	t2
3	p3	q3	t3
4	p4	q4	t3
5	p5	q5	t4
6	p6	q6	t5
7	p7	q7	t6

4

i	Price	Q	Trans
1	p1	q1	t1
2	p2	q2	t2
3	p3	q3	t2
4	p4	q4	INT_MIN
5	p5	q5	t3
6	p6	q6	t3
7	p7	q7	t4

Obrázek 5.1: Vizualizace různých intervalů rekonstrukce ve zkráceném zdrojovém souboru CSV (vlastní zpracování).

Tabulka číslo 1 zobrazuje nejběžnější situaci. Kniha objednávek má být zrekonstruována k času ***t4***. Tomuto řádku nepředchází žádný řádek se zápornou časovou značkou ve sloupci *Trans* (nedošlo k žádnému obchodu) a tak interval bude vypadat následovně: ***(3; 4)***.

Tabulka číslo 2 zobrazuje situaci, kdy má být kniha objednávek zrekonstruována k času ***t3***. Tomuto řádku předchází tři vykonané obchody (tři řádky se zápornou časovou značkou ve sloupci *Trans*). Tyto řádky je potřeba zaznamenat pro další zpracování, a tak výsledný interval bude vypadat následovně: ***(2; 6)***.

Tabulky číslo 3 a 4 zobrazují speciální situace, ke kterým v souboru může také dojít. Několik řádků za sebou (i více než dva) může mít stejnou časovou značku

ve sloupci *Trans* (v tabulkách se jedná o časovou značku ***t3***). V takovém případě je validní stav knihy objednávek v posledním řádku se stejnou časovou značkou. Pokud má tedy být kniha objednávek zrekonstruovaná k času ***t3***, najde se poslední řádek, který tuto značku obsahuje. Interval v takovém případě vypadá následovně: pro tabulku 3 ***(2; 4)***, pro tabulku 4 ***(3; 6)***.

Tyto intervaly jsou generovány metodou `get_time_seq`. Výstupem této metody je interval, který se použije jako vstup pro zmíněnou metodu `calc_order_book_state`. Ta jako první zkontroluje, zda jsou v předaném intervalu nějaké vykonané obchody a případně je uloží do proměnné `__executes`. Následuje samotná rekonstrukce knihy objednávek a nastavení proměnných `__bookA` a `__bookB`.

Webová aplikace. Jak již bylo zmíněno, aplikace je dostupná na adrese a portu specifikovaných v konfiguračních proměnných `addr` a `port`. Po spuštění je zobrazena výchozí kniha objednávek definovaná v hlavní funkci `main`. Aplikace zobrazuje pro jaký instrument, produkt, datum a čas je kniha objednávek aktuálně zrekonstruována. Umožňuje náhled na knihu objednávek (všechny nabídkové i poptávkové objednávky a jejich množství), na vykonané obchody a na graf vykreslující obchodované množství pro každou cenu.

Uživatel může v aplikaci změnit zdrojový soubor zadáním nového názvu instrumentu, produktu a obchodního dne. O vykonání se stará metoda `change_data_df`, a pokud pro zadané hodnoty neexistuje datový soubor, zůstane aplikace v původním stavu.

Aplikace také umožňuje měnit čas, ke kterému má být kniha objednávek zobrazena. Uživatel může zadat konkrétní čas (ve formátu `hh:mm:ss.nnnnnnnnn` nebo `hh:mm:ss`), v takovém případě je volána funkce `get_time_seq`, nebo může aktuálně zvolený čas posunout o skok dopředu či dozadu. Skok je v tomto případě následující či předcházející časová značka ve zdrojovém souboru. O tuto funkci se starají metody `get_next_time_seq` a `get_prev_time_seq`, které opět naleznou požadovaný interval řádků podle popsaných pravidel.

API. Rozhraní poskytuje uživatelům funkci získání stavu knihy objednávek vzdáleně. Ve výchozím stavu je dostupné na adrese: `http://addr:port/api/`. Adresu lze konfigurovat v návěští `@server.route` funkce `api_get_data`. Rozhraní přijímá požadavky a odesílá odpovědi ve formátu JSON ([definice]).

S příchodem prvního požadavku se inicializuje instance knihy objednávek `api_OB`. Ta poté uchovává stav knihy objednávek po posledním zpracovaném požadavku. Požadavek musí specifikovat čtyři parametry: název instrumentu, číslo produktu, datum a čas. Pokud jsou zadané hodnoty validní, je podle nich zrekonstruována kniha objednávek a její stav zaslán v odpovědi. Pokud jsou hodnoty nevalidní, nebo

pro ně chybí datový soubor, je zaslána prázdná odpověď. V odpovědi je zaslána časová značka, ke které rekonstrukce proběhla, všechny nabídkové a poptávkové objednávky s množstvím, a vykonané obchody.

Zdrojový kód 5.5: Ukázka požadavku na API (vlastní kód).

```
url = 'http://127.0.0.1:1234/api/'
params = {"Instrument":"FGBL",
      "Security":"4128839",
      "Date":"20191202",
      "Time":"15:15:04.077796640"}
response = requests.get(url=url,params=params)
```

Zdrojový kód 5.6: Ukázka zkrácené odpovědi z API (vlastní kód).

```
{ "Asks": [
        {"Price": 171.95, "Qty": 146},
        {"Price": 171.96, "Qty": 224}],
    "Bids": [
        {"Price": 171.94, "Qty": 29},
        {"Price": 171.93, "Qty": 140}],
    "Executes": [],
    "Time": "54904077796640"
}
```

5.5 Testování serveru pro rekonstrukci

Jako první byl prováděn test správnosti rekonstrukce knihy objednávek. Korektnost byla testována pomocí dalšího z řady systémů, které společnost Deutsche Börse nabízí. Jejich analytická platforma **A7** poskytuje přístup k tržním datům objednávek na burzách **Eurex** a **Xetra**. Umožňuje také uživatelům detailní náhled na knihy objednávek a nabízí pokročilé nástroje analýzy tržní situace. Tato platforma také poskytuje rozhraní, které umožňuje na základě zadaných parametrů stáhnout kompletní stav knihy objednávek.

V rámci této práce byly zrekonstruovány dvě různé knihy objednávek pro tyto parametry:

1. `instrument` = *"FGBL"*, `security` = "4128839", `date` = "20191202",
2. `instrument` = *"FGBX"*, `security` = "5315926", `date` = "20210104".

Pro stejné parametry byly z platformy **A7** staženy informace o daných knihách objednávek. Aby bylo porovnání správnosti co nejpřesnější, byl pro každou knihu objednávek uložen její stav pro každou časovou značku v daném dni, kterou platforma

dovolila nastavit. To samé bylo provedeno i pro lokálně zrekonstruované knihy objednávek. Pro každou časovou značku, kterou zdrojový soubor pro rekonstrukci obsahoval, se vygeneroval stav knihy objednávek a uložil do souboru.

Pro každou knihu objednávek tak vznikly dva soubory, jeden s daty z lokálně zrekonstruované knihy, a druhý z platfromy **A7**. Jako jednotný formát dat v obou souborech se použil ***LOBSTER*** (Limit Order Book System – The Efficient Reconstructor) formát. Definice formátu je dostupná zde: [definice]. Tento formát dat je speciálně navržen pro práci s daty limitních objednávkových knih. Definuje předpis každého řádku jako sekvenci cen a množství, pro jednotlivé úrovně objednávek, a pro danou časovou značku. Úroveň objednávek představuje dvojici nabídkové a poptávkové objednávky, které mají v knize objednávek stejnou prioritu (nabídková objednávka s nejvyšší cenou a poptávková objednávka s nejnižší cenou, druhá nejvyšší nabídková a druhá nejnižší poptávková, atd.).

Zdrojový kód 5.7: Ukázka zkráceného formátu LOBSTER (vlastní zpracování).

```
1 Time,Ask Price 1,Ask Volume 1,Bid Price 1,Bid Volume 1
2 904158156918,173.01000000,0.00010000,172.95000000,0.00040000
3 904243590539,173.01000000,0.00010000,172.95000000,0.00040000
```

Pro kontrolu bylo stanoveno, že se bude porovnávat prvních třicet úrovní (na ukázce 5.7 lze vidět pouze jedna úroveň). Oba soubory tedy pro každou časovou značku obsahovaly `Ask Price 1` až `Ask Price 30` a `Bid Price 1` až `Bid Price 30`.

Kontrola byla provedena skriptem napsaném v jazyce **Python**, který oba soubory postupně procházel a porovnával hodnoty úrovní pro shodné časové značky. Jako správné hodnoty byly považovány ty ze souboru s daty ze systému **A7**. Dosažené výsledky byly následující:

V knize **20191202-FGBL-4128839** bylo porovnáno 731915 časových značek a shoda byla **99,966%**.

V knize **20210104-FGBX-5315926** bylo porovnáno 2048243 časových značek a shoda byla **99,982%**.

Zátěžový test API. Další test měl za cíl zjistit průměrný čas získání dat přes API. Testování probíhalo celkem na šesti počítačích. Na všech byl spuštěn operační systém **MS Windows**. Na jednom počítači byl spuštěn program `reconstruction.py` a představoval tak server běžící aplikace. Na zbylých pěti počítačích běžel skript požadující data z API.

Skript byl napsán v jazyce **Python**. Obsahoval adresu běžícího serveru a definoval parametry požadavku, který se na server zasílal. Ukázka požadavku lze vidět v kódu 5.5. Parametry byly nastaveny tak, aby se požadavek dotazoval na stav knihy objednávek **20191202-FGBL-4128839**. Čas, ke kterému měl být získán stav knihy objednávek byl jediným parametrem, který se v požadavku měnil. Tento čas, ve for-

mátu `hh:mm:ss` byl pro každý požadavek náhodně generován v rozsahu 00:00:00 až 23:59:59. Skript vygeneroval a zaslal celkem 10000 požadavků, a měřil čas od zaslání požadavku do zíkání odpovědi.

Tento skript byl v jeden moment spuštěn na všech pěti počítačích. Výsledný průměrný čas získání dat z API byl **0,1392 sekund**.

Zhodnocení. Díky těmto provedeným testům byla ověřena přesnost rekonstrukce a také stabilita naprogramovaného serveru. Přesnosti, kterých se dosáhlo na obou knihách objednávek byly pro tuto práci dostačující. Implementovaný systém by měl být schopen s podobnou přesností zrekonstruovat i další knihy, pro které budou poskytnuta příslušná data. Zátěžový test nejen prokázal rychlou rekonstrukci knihy objednávek k náhodnému času, ale také stabilitu serveru, který se během testu nedostal do nevalidního stavu, nebo nezaslal chybná data.

Důvodů, proč není u testu přesnosti shoda knih objednávek úplná může být několik. Jeden z důvodů byl odhalen při bližším zkoumání stavu knihy při konkrétních časových značkách, ve kterých nebyla shoda. V několika málo případech algoritmus řazení zpráv zařadil zprávy odlišně, než jak byly zařazeny v systému **A7**. Většinou se ale jednalo pouze o vzájemné prohození dvou zpráv. Konkrétní situace byla například, když algoritmus řazení dle definovaného pravidla umístil nejprve zprávu **Partial Order Execution** a poté zprávu **Order Delete**. Stejná situace byla poté vyhledána ve webové aplikaci systému **A7**, kde byly zprávy provedeny v opačném pořadí. Tyto anomálie mohou být zapříčiněny ztrátou datagramu zaslaného **T7 EOBI** a přijetím znovuposlaného opravného datagramu, kde ale zpráva má již jinou hodnotu parametru *PARENT_ID*.

Závěr 6

Tato práce se nejprve zaměřila na pochopení fungování finančních trhů. Rozebrány byly různé typy objednávek, které mohou účastníci trhů využívat pro své obchodní strategie. Dále bylo obecně popsáno, jak se z těchto objednávek tvoří objednávkové knihy a za jakým účelem je finanční instituce používají. Následovala analýza konkrétního burzovního systému **T7 EOBI** německé finanční společnosti Deutsche Börse. Popsáno bylo interní fungování toho systému, jeho funkční charakteristiky a typ burzovních dat, který tento systém zprostředkovává svým zákazníkům.

Praktická část práce se zaměřovala na zpracování dat získaných ze zmíněného systému **T7 EOBI**. Nejprve byl zkoumán vhodný způsob uložení dat. Testovány byly databáze **InfluxDB**, **MariaDB** a **SQLite**, ale výsledným řešením bylo uložení dat do souborů CSV. Dále byl navrhnut systém předzpracování dat do vhodného formátu a z něj následné rekonstrukce knihy objednávek. Navrhnutý systém byl poté implementován a jeho správnost byla otestována vůči analytickému systému **A7** stejné společnosti.

Implementovaný systém úspěšně rekonstruuje knihu objednávek a poskytuje požadované funkce jako vizualizaci stavu a rozhraní pro zasílání požadavků. Zadání bylo splněno v celém rozsahu. Kniha objednávek však zobrazuje pouze kumulativní obchodované množství pro každou cenu. Možným pokračováním by bylo rozšířit implementaci tak, aby kniha zobrazovala jednotlivé konkrétní objednávky od obchodníků, ze kterých je kumulativní množství tvořené. Následujícím rozšířením by také mohlo být zpracování dalších typů zpráv, které **T7 EOBI** zprostředkovává.

Uživatelská příručka

V této příloze bude popsáno použití vytvořeného programu včetně jeho možných konfigurací.

Softwarové požadavky. Přiložené programy byly napsány v jazyce **Python**, který je multiplatformní. Mělo by je tedy být možné spustit na operačních systémech **MS Windows**, **Linux** i **macOS**. Je ale potřeba mít na daném operačním systému nainstalovaný jazyk **Python 3**. Programy byly psány ve verzi **Python 3.8.8**, a je tedy doporučena jako minimální verze pro úspěšné spuštění. Dále je zapotřebí mít nainstalované následující knihovny v uvedených verzích (nebo novějších):

- **NumPy** verze 1.20.3 (ke stažení zde: numpy),
- **Pandas** verze 1.3.1 (ke stažení zde: pandas),
- **Flask** verze 1.1.2 (ke stažení zde: flask),
- **Dash** verze 2.16.0 (ke stažení zde: dash),
- **Dash Bootstrap Components** verze 1.5.0 (ke stažení zde: dash-bootstrap-components),
- **Fontawesome** verze 5.10.1 (ke stažení zde: fontawesome).

Spuštění a konfigurace programu merge.py. Program je spouštěn bez jakýchkoliv parametrů příkazem `...\>python merge.py`.

Konfigurační parametry se nachází v hlavní funkci `main`. Jejich význam je popsán v kapitole 5.2. Výchozí hodnoty těchto parametrů jsou:

- `delim = ",";`
- `path = "raw/";`
- `output_format_params = ["i", "PARENT_ID", "ID", "TrdRegTSTimeIn", "TrdRegTSTimePriority", "Side", "Price", "DisplayQty", "op", "Trans", "Prio"];`

- `qty_rounder = 10000;`
- `outfile = "merged/"+date+"-"+instrument+"-"+security+".csv"`.

Výchozí hodnoty parametru `data_files` jsou nastaveny pro zpracování dat knihy objednávek 20191202 – *FGBL* – 4128839, a lze je vidět v kódu A.1. Parametr představuje datový typ slovník, kde klíčem je název souboru a hodnotou typ zprávy. Typ zprávy se poté používá při vytváření instancí třídy `Tagged_line` jako parametr `source_op`, který dále v hlavní `while` smyčce udává typ zpracování pro daný řádek. Pokud je tedy potřeba přidat nový typ zprávy pro zpracování, musí se nejprve přidat nový název souboru a typ zprávy do parametru `data_files`, a poté pro daný typ zprávy přidat podmínku do hlavní `while` smyčky, kde bude definováno, jak se má zpráva zpracovat.

Zdrojový kód A.1: Výchozí hodnoty parametru `data_files` programu `merge.py` (vlastní kód).

```
data_files = {
  "2350_00_D_03_A_20191202.OrderAdd_FGBL_4128839.csv":"A",
  "2350_00_D_03_A_20191202.OrderModify_FGBL_4128839.csv":"M",
  "2350_00_D_03_A_20191202.OrderDelete_FGBL_4128839.csv":"D",
  "2350_00_D_03_A_20191202.OrderModifySamePrio_FGBL_4128839.
   csv":"MS",
  "2350_00_D_03_A_20191202.FullOrderExecution_FGBL_4128839.
   csv":"FE",
  "2350_00_D_03_A_20191202.PartialOrderExecution_FGBL_4128839
   .csv":"PE",
  "2350_00_D_03_A_20191202.ExecutionSummary_FGBL_4128839.csv"
   :"ES"
}
```

Spuštění a konfigurace programu preprocess.py. Program je spouštěn bez jakýchkoliv parametrů příkazem `...\>python preprocess.py`.

Konfigurační parametry se nachází na začátku souboru. Jejich význam je popsán v kapitole 5.3. Výchozí hodnoty těchto parametrů jsou nastaveny pro zpracování dat knihy objednávek 20191202 – *FGBL* – 4128839. Konkrétně pro zpracování výstupního souboru programu `merge.py` pro tuto knihu. Hodnoty parametrů jsou:

- `delim = ",";`
- `infile = "merged/20191202-FGBL-4128839.csv";`
- `incols = ["PARENT_ID", "ID", "TrdRegTSTimeIn", "TrdRegTSTimePriority", "Side", "Price", "DisplayQty", "op", "Trans", "Prio"];`

- `outfile = "data/20191202-FGBL-4128839-ob.csv";`
- `outcols = ["i", "Price", "DisplayQty", "Q", "from", "to", "Trans", "Prio"];`
- `ops_to_include = ["A", "D", "XXX", "YYY", "E", "MS", "PE"]`.

Hodnota parametru `infile` definuje cestu a název vstupního souboru, který byl vytvořen programem `merge.py`. Musí se tedy shodovat s formátem názvů a umístěním těchto souborů definovaných parametrem `outfile` souboru `merge.py`. To stejné platí pro parametr `incols`, který popisuje názvy sloupců těchto vstupních souborů. Tento parametr musí obsahovat podmnožinu názvů sloupců definovaných parametrem `output_format_params` souboru `merge.py`. Parametr `ops_to_include` musí obsahovat podmnožinu hodnot sloupce *op* vstupního souboru. Pokud tedy byl do zpracování přidán nový typ zprávy, je třeba doplnit hodnotu sloupce *op* této zprávy do parametru `ops_to_include`.

Spuštění a konfigurace programu reconstruction.py. Program je spouštěn bez jakýchkoliv parametrů příkazem `...\>python reconstruction.py`.

Konfigurační parametry se nachází ve třídě `Config`. Jejich význam je popsán v kapitole 5.4. Výchozí hodnoty těchto parametrů jsou:

- `addr = "127.0.0.1";`
- `port = 1234;`
- `delim = ",";`
- `path = "data/";`
- `df_cols = ["Price", "DisplayQty", "Q", "from", "to", "Trans", "Prio"];`
- `df_cols_type = {"Price": np.float64, "DisplayQty": np.int64, "Q": np.int64, "from": np.int64, "to": np.int64, "Trans": np.int64, "Prio": str}`.

Program pracuje se soubory vytvořenými programem `preprocess.py`. Program předpokládá adresář a formát pojmenování souborů stejný, jako je definováno v parametru `outfile` programu `preprocess.py`. Změna adresáře je možná nastavit v parametru `path`, a změnu formátu pojmenování je nutné změnit v metodě `change_data_df` třídy `OB`. Parametr `df_cols` musí obsahovat podmnožinu hodnot parametru `outcols` programu `preprocess.py`.

Webová aplikace má nastavené výchozí hodnoty pro zobrazení knihy objednávek 20191202 – *FGBL* – 4128839. Tyto hodnoty lze změnit v hlavní funkci `main` v příkazu: `dash_OB = OB("FGBL", "4128839", "20191202")`. Proměnné `dash_time`, `dash_book`, `dash_figure` a `dash_OB` uchovávají kompletní stav webové aplikace a jsou používány pro vizualizaci a výpočty nad knihou objednávek. Interakce webové aplikace lze měnit ve funkci `update_orderbook`. Vzhled aplikace lze měnit v hlavní funkci `main` v parametru `app.layout`.

Rozhraní je konfigurovatelné ve funkci `api_get_data`. Výchozí adresa rozhraní je `"http://addr:port/api/"` a výchozí metoda zasílání je `GET`. Tyto hodnoty lze nastavit v návěští `@server.route("/api/", methods=["GET"])` zmíněné funkce. Formát odpovědi je definován v proměnné `payload`.

Bibliografie

[Abe+16] ABERGEL, Frederic et al. *Limit Order Books*. Cambridge University Press, 2016. Dostupné také z: https://shorturl.at/ouP12.

[Ama] AMAZON WEB SERVICES. *What is NoSQL?* Dostupné také z: https://aws.amazon.com/nosql/.

[Deu20] DEUTSCHE BÖRSE GROUP. *Enhanced Order Book Interface v8.1.1*. 2020-07-23. Dostupné také z: https://shorturl.at/pJLNR.

[Dix] DIX, Paul. *Time series database (TSDB) explained*. Influxdata. Dostupné také z: https://www.influxdata.com/time-series-database/.

[Fla] FLASK. *Flask documentation*. Dostupné také z: https://flask.palletsprojects.com/en/3.0.x/.

[Gee23] GEEKSFORGEEKS. *Difference between SQLite and MariaDB*. 2023-05-03. Dostupné také z: https://www.geeksforgeeks.org/difference-between-sqlite-and-mariadb/.

[Gou+13] GOULD, Martin D. et al. *Limit order books*. University of Oxford, 2013-04-16. Dostupné také z: https://www.math.ucla.edu/~mason/papers/gould-qf-final.pdf.

[Har02] HARRIS, Larry. *Trading and Exchanges: Market Microstructure for Practitioners*. Oxford University Press, 2002. Dostupné také z: https://shorturl.at/nqxy5.

[IBM] IBM. *What is a relational database?* Dostupné také z: https://www.ibm.com/topics/relational-databases.

[Infa] INFLUXDATA. *InfluxDB key concepts*. Dostupné také z: https://docs.influxdata.com/influxdb/v1/concepts/key_concepts/.

[Infb] INFLUXDATA. *InfluxDB storage engine*. Dostupné také z: https://docs.influxdata.com/influxdb/v2/reference/internals/storage-engine/.

[JSO] JSON. *Introducing JSON*. Dostupné také z: https://www.json.org/json-en.html.

[Mad23] MADUSHAN, Dhanushka. *How Database B-Tree Indexing Works*. Built In, 2023-02-28. Dostupné také z: https://builtin.com/data-science/b-tree-index.

[Mar] MARIADB. *Choosing the Right Storage Engine*. Dostupné také z: https://mariadb.com/kb/en/choosing-the-right-storage-engine/.

[MyS] MYSQL. *Introduction to InnoDB*. Dostupné také z: https://dev.mysql.com/doc/refman/8.3/en/innodb-introduction.html.

[Num] NUMPY. *NumPy documentation*. Dostupné také z: https://numpy.org/doc/stable/.

[Pan] PANDAS. *Pandas documentation*. Dostupné také z: https://pandas.pydata.org/docs/.

[Plo] PLOTLY. *Dash Python User Guide*. Dostupné také z: https://dash.plotly.com/.

[Pyt] PYTHON SOFTWARE FOUNDATION. *heapq — Heap queue algorithm*. Dostupné také z: https://docs.python.org/3/library/heapq.html.

[SEC17] SEC. *Investor Bulletin: Understanding Order Types*. investor.gov, 2017-07-12. Dostupné také z: https://shorturl.at/jkFZ5.

[Sot] SOTO, Carlota. *InfluxQL, Flux, and SQL: Which Query Language Is Best?* Timescale. Dostupné také z: https://www.timescale.com/learn/influxql-flux-sql-which-query-language-is-best-with-cheatsheet.

[SQL] SQLITE. *What Is SQLite?* Dostupné také z: https://www.sqlite.org/index.html.

[Tam23] TAMPLIN, True. *Limit order books*. financestrategists.com, 2023-09-07. Dostupné také z: https://shorturl.at/dkmuQ.

Seznam obrázků

Seznam tabulek

Seznam výpisů