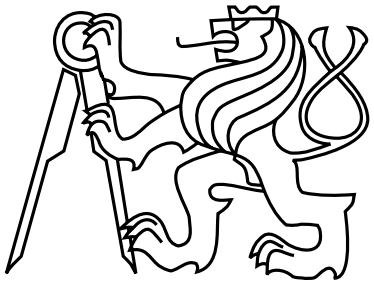
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

Lexikální analyzátor dialektu dotazovacího jazyka databáze MySQL pro zachytávání změn v databázi

Bc. Roman Kuchár

Vedoucí práce: Ing. Jiří Pechanec

Studijní program: Otevřená informatika, Magisterský

Obor: Softwarové inženýrství

22. února 2018

Obsah

Seznam obrázků

Seznam tabulek

Seznam příkladů

Kapitola 1

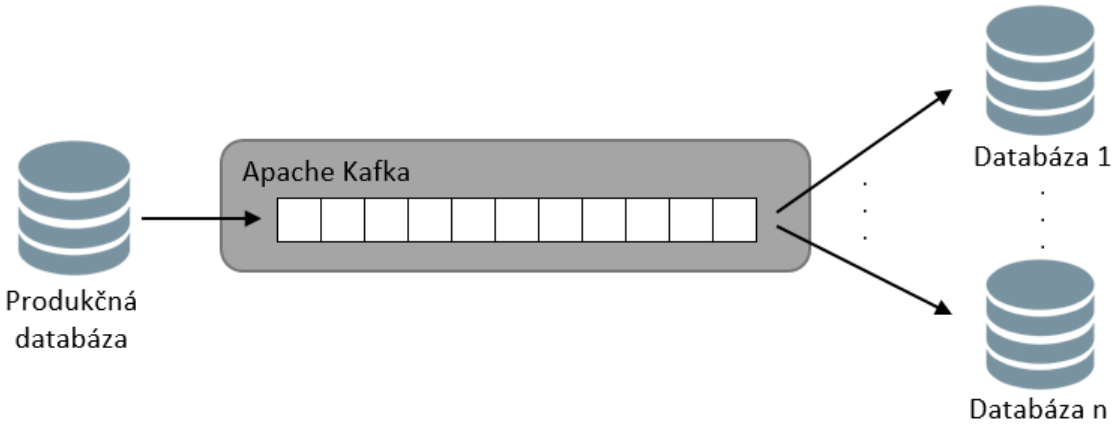
Debezium

Debezium[1] je projekt, ktorý slúži k zaznámenavaniu zmien v databázi analýzov udalostí v transakčnom logu. Jednou z podporovaných databázi je taktiež MySQL. Na správnu funkcionalitu Debezium potrebuje metadáta popisujúce štruktúru databáze v závislosti na čase. Pre MySQL je to možné dosiahnuť tak, že príkazy, ktoré vytvárajú alebo upravujú štruktúru databáze sú zachytávané, analyzované a na ich základe je upravený model v pamäti, ktorý popisuje štruktúru databáze. Stávajúci lexikálny analyzátor je ručne napísaný, veľmi jednoduchý a zďaleka nepostihuje všetky nuance SQL jazyka, čím sa stáva nachylným k chybám. Novo implementovaný strojovo generovaný lexikálny analyzátor nahradí aktuálne riešenie v projekte Debezium, čím sa zníži pravdepodobnoť vzniku chýb, a bude možné analyzátor upraviť jednoduchou zmenou v gramatike jazyka nad ktorým bude pracovať.

TODO: vykuchat do uvodu

1.1 Zachytávanie zmenených dát

Hlavnou myšlienkou zachytávania zmenených dát, anglicky Change Data Capture (CDC), je vytvárať sled událostí, ktoré reprezentujú všetky zmeny v tabulkách danej databáze. To znamená, že pre každý *insert*, každý *update*, každý *delete* dotaz sa vytvorí jedena odpovedajúca udalosť, ktorá bude odoslaná a následne dostupná pre konzumentov tohto sledu viď obrázok 1.1. V projekte Debezium sa na sprostredkovanie sledu události využíva Apache Kafka [3] infraštruktúra, no myšlienka CDC nie je viazaná na Kafku.



Obrázek 1.1: Koncept distribúcie zmenených dát[2]

1.1.1 Replikácia dát

Jedným z využití CDC je replikácia dát do iných databáz napríklad v zmysle vytvorenia zálohy dát, ale taktiež je možné CDC využíť pri implementácií zaujímavých analytických požiadavkov. Predstavme si, že máme produkčnú databázu a specializovaný analytický systém na ktorom chceme spustiť analýzu. V tomto prípade je nutné dostať dáta z produkčnej databáze do analytického systému a CDC je možnosť, ktorá nám to umožní. Ďalším využitím môže byť prísun dát ostatným týmom, ktoré na základe nich múžu napríklad vypočítavať a smerovať svoju marketingovú kampaňn napríklad na užívateľov, ktorý si objednali istý konkrétny produkt. Nakoľko necheme aby sa takýto výpočet vykonával nad produkčnou databázov ale skôr nad nejakou separovanou databázov, tak opäť je možné využiť CDC na propagáciu dát do separovanej databáze, kde si už marketingový tým môže vykonávať akokoľvek náročné výpočty.

1.1.2 Microservice Architecture

Ďalšie využitie CDC je vhodné pri použití Microservice architecture, kde je doména rozdelená na niekoľko služieb, ktoré potrebujú mdzi sebou interagovať. Pre príklad máme tri micro služby: objednávaciu aplikácia na spracovávanie užívateľských objednávok, produktovú službu, ktorá sa stará o produktový katalóg, a nakoniec máme skaldovú službu, ktorá kontroluje reále množstvo produktových vecí na sklade. Je zretelné, že na správne fungovanie bude objednáva aplikácia vyžadovať dáta od produktovej a skladovej služby. Jednou z možnosí je, že objdnávacia aplikácia bude priamo komunikovať z ostatnými službami napríklad pomocou REST api[1], čím ale bude úzko spojená a závislá na chode danej služby. Ak by takáto služba zlyhala/spadla tak nebude fungovat celá aplikácia. Druhou možnosťou je práve využiť CDC, kde produktová a skladová sužba budú poskytovať sled zmenených dát a objednávacia aplikácia ich bude zachytávať a udržiavať kópiu časti týchto dát, ktoré ju

[1] <https://cs.wikipedia.org/wiki/Representational_State_Transfer>

zaujímjú, vo vlastnej lokálnej databáze. Ak by v takomto prípade niektorá zo služieb zlyhala, tak objednávacia aplikácia môže naďalej fungovať.

1.1.3 Ostané

Bežnou praxou vo väčších aplikáciacch je používanie cache pre rýchly prístup k dátam na základe špecifických dotazov. V takýchto prípdoch je potrebné riešiť problémy updatu cache alebo jej invalidaácie, pokiaľ sa isté dáta zmenia.

Riešenie fulltextového vyhľadávania pomocou databáze nie je veľmi vhodné a namiesto toho sa používa SOLR[2] alebo Elasticsearch[3], čo sú systémy, ktoré potrebujú byť synchronizované z dátami v primárnej databáze.

1.2 Odchytávanie zmien v databázi

Každý databázový systém má svoj log súbor, ktorý používa na zotavenie sa po páde a rollbacknutie transakcií, ktoré ešte neboli commitnuté alebo na replikáciu dát voči sekundárným databázam alebo inej funkcionalite. Či už to sú transakčné, binárne alebo replikačné logy, vždy v sebe udržujú všetky transakcie, ktoré boli úspešne vykonané nad databázou, a preto sú vhodné na odchytávanie zien v databázach pre projekt Debezium. Konkrétne v MySQL databáze sa volá **binlog** (2.1.1). Nakoľko sú tieto logy plne transparentné voči aplikácií, ktorá do databáze zapisuje, výkon aplikácie nebude nijako ovplyvnený čítaním týchto logov.

1.2.1 Infraštruktúra správ pomocou Apache Kafka

Apache Kafka[3] poskytuje semantické pravidlá, ktoré dobre vyhovujú potrebám projektu Debezium. Prvým z nich je, že všetky správy v Kafke majú kľúč a hodnotu. Táto vlastnosť sa využíva na zjednotenie správ, ktoré spolu súvisia a to konkrétne tak, že na základe primárneho kľúča v tabuľke, ktorej zmena sa zmena týka je možné štruktúrovať kľúč správy a hodnota správy bude reprezentovať konkrétnu zmenu.

Kafka taktiež garantuje poradie správ metodou FIFO[4], čím sa zabezpečí správne poradie zmien, ktoré bude konzument príjmať. Táto vlastnosť je veľmi dôležitá nakoľko ak by nastala situácia *insert* a následne *update* alebo dve *update* akcie za sebou, tak musí byť zabezpečené aby sa ku konzumentovi dostali v správnom poradí inač by mohla nastať nekonzistencia voči dátam v primárnej databáze a dátam s ktoré si udržiava konzument.

Kafka je pull-based systém, čo znamená, že konzument je sám sebe pánom a drží si informáciu o tom, ktoré správy z konkrétneho topiku už prečítal resp. kde chce začať čítanie ďalších správ. Takto môže sledovať aktuálne pribúdajúce správy, ale taktiež sa môže zaujímat aj o správy z minulosti.

Zmien v databázach môže byť veľmi veľa, čo spôsobí veľké množstvo údálosti, a preto je nutné spomnúť ďalšiu výhodu kafky a to jej škálovateľnosť. Kafka podporuje horizontálnu

[2] <http://lucene.apache.org/solr/>
[3] <https://www.elastic.co/>
[4] First in First out

škálovateľnosť a jednotivé topiky môžu byť rozdelené na viacero partícií. Je ale nutné si uvedomiť, že poradie zmien je garantované iba na konkrétnej partícii. Kafka zabezpeí, že všetky správy z rovnakým kľúčom budú na rovnakej partícii, čím sa garantuje ich správne poradie, ale môže nasstať situácia, že událosť s iným kľúčom, ktorá reálne nastala neskôr, môže byt kozumentom spracovávana skôr, čo môže, ale aj nemusí vadiť v závisloti na konkrétnej funkcionalite konzumenta.

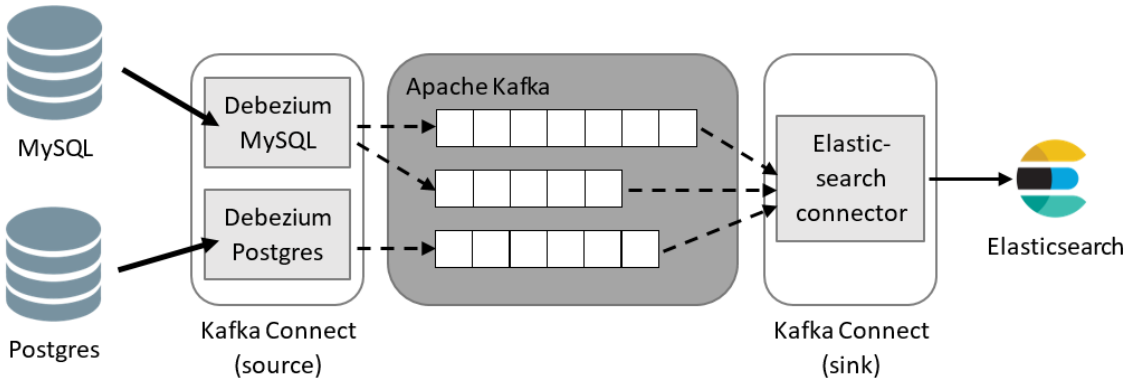
1.2.2 Kafka connect

Kafka connect je framework, ktorý umožnuje jednoduchú implementáciu dátových spojení z Kafkou. Tieto konektory majú nastarosti dáta, ktoré vstupjú alebo vystupujú z Kafky. Nazývajú sa *source* konektory alebo *sink* konektory na základe toho, o čo sa starajú. Debeziové konektory majú nastarosti naplňovanie Kafky, takže sa používa *source* konektor. Kafka connect ponúka možnosť na riešenie ofsetu. To znamená, že keď konektor číta eventy z logu udržuje si zároveň pozíciu logu, z ktorej naposledy čítal. Môže nastať situácia, že konektor spadne a bude musieť byť reštartovaný. V takomto prípade konektor potrebuje vedieť ako ďaleko v čítani logu bol a kde má s čítaním pokračovať. Použitím Kafka connect je zabezpečené, že po každom spracovaní údalosti konektor commitne svoj ofset a ak by konektor musel byť reštartovaný tak môže zistit posledný comutnutý ofset a pokračovať v čítaní logu z danej pozície.

Ďalším prínosom je možnosť konfigurácie schémy správ. Kafka connect má svoj typovací systým, ktorý umožnuje popísať štruktúru kľúčov a hodnôt v správach. Bližšie popísané v kapitole 1.2.3.

Kafka connect je clustrovateľná takže je možné v závislosti na špecifikácií rozdeliť konektor a jeho tasky medzi viacero uzlov. Taktiež ponúka bohatý eko-systém konektorov. Na stránkach Confluent[5] je možné si stiahnuť rôzne typy či už *sink* alebo *source* konektorov. Príklad CDC topológie s použitím Kafka connect je na obrázku 1.2. Zámerom v danom obrázku je zdieľať dáta dvoch tabuliek z MySQL databáze a jednej tabuľky z Postgres databáze. Každá monitorovaná tabuľka je vyjadrená jedným topikom v Kafke. Prvým krokom je nastavenie clusterov v Apache Kafka, pričom to môže bežat na jednom alebo viacerých clustroch. Ďalším krokom je nastavenie Kafka Connect, ktorá je oddelená od Apache Kafka a beží v separátnych procesoch alebo clustroch, a ktorá mi bude spravovať spojenie z Apache Kafka. Následne musím deploynut instnacie Debezium konektorov do Kafka connect a to konkrétne MySQL a Postgres konektory, nakoľko sledujem dáta v týchto databázových systémoch. Posledným krokom je konfigurácia aspoň jedného *sink* konektoru, ktorý bude spracovávat dané topiky v Apache Kafka a odosielať ich inému systému (konzumentovi). Na konrétnom príklade je použitý Elasticsearch konektor nakoľko nasím konzumentom je Elasticsearch.

[5] <https://www.confluent.io/product/connectors/>



Obrázek 1.2: CDC topológia z Kafka connect[2]

1.2.3 Štruktúra správy

Ako už bolo spomenuté, správy v Kafke obsahujú kľúč, v prípade Debezia je primárny kľúč v tabuľke, a hodnotu, ktorá má komplexnejšiu štruktúru skladajúcu sa z:

- **before** stavu, ktorý v sebe nesie predchádzajúci stav data, ktoré sa mení. V prípade, že nastane *insert* event, táto hodnota bude prázdna, nakoľko práve vzniká a nemá žiadny predchádrzjúci stav.
- **after** stavu, ktorý v sebe nesie nový stav data. Táto hodnota môže byť opäť prázdna a to v prípade *delete* eventu.
- **source** informácie, ktoré v obsahujú metadáta o pôvode danej zmeny. Skladá sa napríklad z informácií ako meno databázového servru, názvu logovacieho súboru, z ktorého číta a pozíciu v ňom, názvu databáze a tabuľky, timestamp a pod.

Příklad 1.1: Hodnota správy odosielanej Debeziom

```
{
        "schema" : {
                ...
        },
        "payload" : {
                "before" : null,
                "after" : {
                        "id": 352,
                        "name" : "Janko",
                        "surename" : "Hrasko",
                        "email" : "janko@hrasko.sk"
                },
                "source" : {
                        "name" : "dbserver1",
```

```
                    "serer_id" : 0,
                    "ts_sec" : 0,
                    "file" : "mysql-bin.000001",
                    "pos" : 12,
                    "row" : 0,
                    "snapshot" : true,
                    "db" : "todo_list",
                    "table" : "users"
            },
            "op" : "c",
            "ts_ms" : 1517152654614
        }
}
```


Kafka dokáže spracovávať akýkoľvek druh binárných dát, takže jej na tejto logickej štruktúre nezáleží. Na odosielanie správ sa používajú konventory, ktoré prevádzajú správu do formy v ktorej bude odosielaná. Vďaka použitiu Kafka connect je opäť možnosť využit konvertory, ktoré poskytuje a pre Debezium to sú:

- **JSON**, do ktorého je možnosť zahrnúť informácie o schéme dát, na základe ktorej môžu konzumenti správne interpretovať prijatú správu. Tento formát je výhodné používať počas vývoja aplikácie nakoľko je čitateľný pre človeka.
- **Avro**, ktorý má veľmi efektívnu a kompaktntnú reprezentáciu vhodnú na prdukčné účely. Takáto správa nieje čitateľná, nakoľko je to binárna reprezentácia správy. V týchto správach sa nenachádza informácia o schémy tabuľky, ale iba idenetifikátor na danú schému a jej verziu, ktorú je možné získať pomocou registru schémat, čo je ďalšia časť ekosystému Kafky. Konzument môže získať konkrétnu schému z registrov a na základe nej interpretovat binárne dáta, ktoré dostal.

Kapitola 2

Aktuálna implementácia MySQL konektoru

Projekt Debezium sa zkladá z viacerých častí. Hlavnou časťou je jardo systému, ktoré je spoločné pre všetky typy konektorov podporovaných Debeziom. Jadro systému zaobstaráva základnú funckionalitu spojenú s CDC, ktorú tento systém podpruje. Definuje model sledovaných dát, na základe ktorých si systém udržuje aktuálne schémata tabuliek a ich dátový stav v pamäti. Jedným z týchto podporovaných konektorov je aj konektor pre MySQL databázu 2.1.

TODO co viac k tomu ?

2.1 MySQL konektor

Minimálnou podporovanou verziou MySQL je aktálne verzia 5.6. MySQL konektor Debezia dokáže sledovať zmeny v databázy na úrovni jednotlivých riadkov v tabuľkách pomocou čítania databázového binlogu (2.1.1). Pri prvom pripojení na MySQL server si konetor vytvorí aktuálny obraz všekých tabuliek (2.1.2) a následne sleduje všetky komitnuté zmeny, na základe ktorých vytvára jednotlivé *insert*, *update* a *delete* eventy. Pre každú tabuľku je vytvorený separátny topik v Kafke v ktorom sa ukladajú eventy spojené z danou tabuľkou. Týmto spôsobom je zabezpečený štart s konzistentným obrazom všetkých dát.

Konektor je taktiež veľmi tolerantný vočí chybám. Zároveň s čítaním udalostí z binlogu si konektor ukladá ich pozíciu. Ak by nastala akákolvek situácia pri ktorej by konektor prestal pracovať a bol by nutný jeho reštart, tak jednoducho začne čítanie binlogu na pozícii na ktorej skončil pred pádom. Konektor sa bude rovnko správať aj keby chyba a jeho pád nastali počas prvotného vytvárania aktuálneho obrazu.

2.1.1 Binárny log

v MySQL je možné implementovať CDC na základe sledovania binárneho logu v skratke nazývaného binlog[4]. Binlog obsahuje všetky události, ktoré popisujú zmeny vykonané nad

MySQL databázou ako napríklad vytváranie tabuliek alebo zmena dát. Poradie týchto udalosti je zachované voči reálnemu poradiu ako boli SQL dotazý vykonávané. Toto binárne logovanie sa využíva na dva základne účely:

- Pre **replikáciu**, kde binlog na master replikačnom servri sprostredkuváva záznamy o zmenách, ktoré majú byť odoslané slave serverom. Master server odošle udalosti nachádzajúce sa v binlogu slave servri, ktorý tieto udalosti vykoná u seba za účelom udržania rovnakého dátového stavu ako je na master replikačnom servri.
- Pre **obnovu systému z chybového stavu** anlicky nazývanú **recovery**. Po nahraní zálohy databáze sú znouspustené udalosti zaznamenané v binlogu, ktoré nastali po vytvorení zálohy, čím sa zabezpečí konzistentný stav databázových dát z dátami v dobe zlyhania databázového servri.

Binárny log neobsahuje udalosti, ktoré nemajú žiadny efket na dáta ako napríklad SELECT alebo SHOW. Udalosti môžu byť do logu zapisované v rôznych formátoch, na základe ktorých sa mení aj spôsob replikácie dát. Tieto formáty logovania sú:

- **Statement-based** logovanie, v ktorom udalosti obsahujú SQL dotazy, ktoré produkujú zmeny v dátach (INSERT, UPDATE, DELETE). V rámci tohto logovanie môže taktiež obsahovať dotazy, ktoré môžu iba potencionálne meniť dáta napríklad DELETE dotaz, ktorý sa nespáruje so žiadnymi dátami. Pri replikácií slave server číta binlog a zaradom vykonáva SQL dotazy, ktoré obsahujú jednotlivé udalosti.
- **Row-based** logovanie, v ktorom udalosti popisujú zmeny pre jednotlivé riadky v tabuľkách. Pri replikácií sa kopírujú eventy, ktoré reprezentujú zmeny riadkov v tabuľkách na slave servri.

Pre účely CDC v Debeziu je používané row-based logovanie, nakoľko zalogované udalosti obsahujú zmeny pre konkrétne riadky v tabuľkách a tým pádom nieje nutné dopočítavať dáta, ktoré by boli na základe daného dotazu zmenené. Master server ukladá do binlogu iba kompletné a vykonané transakcie, čo znamená, že sa tam nemôžu objaviť syntakticky nevalidné dotazy. V MySQL konektoru teda nie je nutné sledovať korektnosť parsovaných dotazov.

Pomocou SQL dotazu `SHOW BINARY LOGS` je možné vylistovať aktuálne existujúce binárne logy na servri. Následne dotazom `SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]` je možné sledovať informácie o všetkých udalostiach obsiahnutých v danom binlogu ako sú napríklad typ udalosti, jeho začiatočná a jeho konečná pozícia v logu. Na čítanie a spracovávanie binlogu ponúka MySQL nástroj *mysqlbinlog*, ktorý je možné spustiť príkazom `mysqlbinlog [options] log_file`. Prvý riadok udalosti vždy obsahuje prefix *# at* za ktorým následuje číslo reprezentujúce pozíciu udalosti v binlogu. Podľa základného nastavenia MySQL zobrazuje mysqlbinlog udalosti týkajúce sa zmien na úrovní riadkov zakódované ako base-64[1] použitím interného príkazu *BINLOG*. Aby bolo možné vidieť tento pseudokód je možné použiť prepínač `--verbose` alebo `-v`. Na výstupe bude možné vidiet tento pseudokód na riadkoch, ktoré budú začínať prefixom *###*. Použitím prepínača

[1] typ kódovania, ktorý prevádza binárne dáta na postupnost znakov

`--verbose` alebo `-v` dvakrát, môžeme nastaviť aj zobrazovanie dátových typov a iných metadát pre každý stĺpec. Aby sa v logu nezobrazoval interný príkaz *BINLLOG* a zakódovaná hodnota udalostije možné použiť prepínač `--base64-output=DECODE-ROWS`. Kombináciou týchto prepínačov získame možnosť pohodlne sledovat obsah udalosti týkajúcich sa zmien v dátach.

Pre Debezium sú dôležité udalosti typu:

- **Query**, v ktorom sa objavujú dotazy na zmenu štruktúry databáze (DDL) ako je možné vidiet na poslednom riadku príkladu udalosti 2.1.
- **Table_map**, pomocou ktorého binlog mapuje konkrétne tabuľky na identifikátor, ktorým sa následne tieto tabuľky referencuje. Príklad tejto udalosti je možné vidieť v príklade 2.2 na pozícii 552 a jeho následné použitie pre udalosť na pozícii 621.
- **Update_rows**, ktorý obsahuje informácie o zmene dát na úrovni riadkov, ako je možné vidiet na udalosti v príklade 2.2 na pozícii 621.
- **Write_rows**, ktorý obsahuje infomrácie o novo vzniknutých dátach.
- **Delete_rows**, ktorý obsahuje informácie o zmazaných dátach.

Příklad 2.1: Query udalost z binárneho logu MySQL

```
# at 219
#180213 9:59:15 server id 223344 end_log_pos 408 CRC32 0x19237396 Query thread_id
    ↪ =15 exec_time=0 error_code=0
use ‘inventory‘/*!*/;
SET TIMESTAMP=1518515955/*!*/;
SET @@session.pseudo_thread_id=15/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.
    ↪ unique_checks=1, @@session.autocommit=1/*!*/;
SET @@session.sql_mode=1436549152/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
/*!\C utf8 *//*!*/;
SET @@session.character_set_client=33,@@session.collation_connection=33,@@session.
    ↪ collation_server=8/*!*/;
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
/* ApplicationName=IntelliJ IDEA 2017.2 */ alter TABLE customers add column
    ↪ phone_number varchar(15) NULL
```

Příklad 2.2: Table_map a Update_rows události z binárneho logu MySQL

```
# at 552
#180219 11:51:22 server id 223344 end_log_pos 621 CRC32 0x622e3e17 Table_map: ‘
    ↪ inventory‘.‘customers‘ mapped to number 109
# at 621
```

```
#180219 11:51:22 server id 223344 end_log_pos 736 CRC32 0x8ec54ac4 Update_rows:
    ↪ table id 109 flags: STMT_END_F
### UPDATE `inventory`.`customers`
### WHERE
### @1=1001 /* INT meta=0 nullable=0 is_null=0 */
### @2='Sally' /* VARSTRING(255) meta=255 nullable=0 is_null=0 */
### @3='Thomas' /* VARSTRING(255) meta=255 nullable=0 is_null=0 */
### @4='sally.thomas@acme.com' /* VARSTRING(255) meta=255 nullable=0 is_null
    ↪ =0 */
### @5=NULL /* VARSTRING(15) meta=15 nullable=1 is_null=1 */
### SET
### @1=1001 /* INT meta=0 nullable=0 is_null=0 */
### @2='John' /* VARSTRING(255) meta=255 nullable=0 is_null=0 */
### @3='Thomas' /* VARSTRING(255) meta=255 nullable=0 is_null=0 */
### @4='sally.thomas@acme.com' /* VARSTRING(255) meta=255 nullable=0 is_null
    ↪ =0 */
### @5=NULL /* VARSTRING(15) meta=15 nullable=1 is_null=1 */
```

2.1.2 Aktuálny obraz tabuliek

Po nakonfigurovaní a prvom spustení MySQL konektoru sa podľa základného nastavenia spustí tvorba aktuálneho obrazu tabuliek sledovanej databáze. Vo väčšine prípadoch už MySQL binlog neobsahuje kompletnú históriu databáze a preto je tento mód v základnom nastavení.

Pri každom vytváraní aktuálneho obrazu, konektor postupuje podľa týchto krokov:

1. Aktivuje globálny zámok čítania (read lock) aby zabránil ostatným databázovým klientom v zapisovaní.
2. Spustí transakciu s izoláciou na opakované čítanie (repeatable read)[2], aby všetky nasledujúce čítania v rámci tejto transakcie boli vočí jednému konzistentnému obrazu.
3. Prečíta aktuálnu pozíciu binlogu.
4. Prečíta schéma databází a tabuliek na základe konfigurácie konektoru.
5. Uvolní globálny zámok, aby ostaný databázový klienti mohli znovu zapisovať do databáze.
6. Voliteľne zapíše zmeny DDL do Kafka topiku vrátane všetkých potrebných SQL dotazov.
7. Oskenuje všetky databázové tabuľky a vygeneruje príslušné *create* události Kafka topiky pre jednotlivé riadky v tabuľkách.

[2] stupeň izolácie založený na používaní *read* a *write* zámkoch, ktorý ale nezabráni prítomnosti fantómov vznikajúcich v stiuácii, keď v jednej transakcii podľa rovnakého dotazu čítame dáta 2x z rôznymi výsledkami, pretože v medzičase stihla iná transakcia vytvoriť alebo zmazať časť týchto dát.

8. Komitne transakciu.

9. Do konektorového ofsetu zaznamená, že úspešne ukončil vytváranie obrazu.

Transakcia vytvorená v druhom kroku nezabráni ostatným klientom upravovať dáta, ale poskytne konektoru konzistentný a nemenný pohľad na dáta v tabuľkách. Nakoľko transakcia nezabráni klientom aplikovať DDL zmeny, ktoré by mohli vadiť konektoru pri čítaní pozície a schémat v binlogu, je nutné v prvom kroku použiť globálny zámok na čítanie k zamedzeniu tohto problému. Tento zámok je udžiavaný na veľmi krátku dobu potrebnú pre konektor na vykonanie krokov tri a štyri. V piatom kroku je tento zámok uvolnený predtým, než konektor vykoná väčšinu práce pri kopírovaní údajov.

Note: moze byt popisane este viac ak by bolo potrebne nahrabat strany :)

Literatura

[1] Debezium Community. *Debezium* [online]. 2018. [cit. 28. 1. 2018]. Dostupné z: <http://debezium.io/>.

[2] MORLING, G. Streaming Database Changes with Debezium. https://www.youtube.com/watch?v=I0Z2Um6e430, publikované 9. 11. 2017.

[3] NARKHEDE, N. – SHAPIRA, G. – PALINO, T. *Kafka: The Definitive Guide: Real-time data and stream processing at scale.* O'Reilly UK Ltd., 2017. ISBN 978-1-491-99065-0.

[4] *MySQL 5.7 Reference Manual.* Oracle Corporation and/or its affiliates, 2018. https://dev.mysql.com/doc/refman/5.7/en/
.