Engineer

Change data capture

at aflæse ændringerne i en database og sende dem videre

Hvad det er

Change data capture, som regel forkortet CDC, betyder at aflæse ændringerne i en database og sende dem videre til andre systemer, efterhånden som de sker. Hver indsættelse, opdatering og sletning bliver til en hændelse, som noget andet kan reagere på: en rapportkopi, der holder sig ajour, et søgeindeks, en anden tjeneste, der skal vide, at en ordre er afgivet. Databasen, hvor ændringen sker, er fortsat den autoritative kilde, og de andre systemer følger efter.

Der er groft sagt to måder at finde ændringerne på. Den enkle er at spørge: med få minutters mellemrum kører en forespørgsel efter rækker, hvis tidsstempel for seneste opdatering er nyere end sidste kørsel. Den anden er at læse databasens egen optegnelse over, hvad den har ændret. PostgreSQL fører den optegnelse i sin write-ahead-log (WAL), og dokumentationen beskriver princippet bag: ændringer i datafilerne skrives først, når de poster, der beskriver dem, ligger sikkert på permanent lager. Fordi loggen allerede rummer hver ændring i den rækkefølge, den skete, giver den en komplet historik, uden at tabellerne skal forespørges overhovedet. Det er den anden fremgangsmåde, logbaseret CDC, man som regel mener med udtrykket i dag.

I PostgreSQL hedder mekanismen logisk dekodning. Dokumentationen beskriver den som den proces, der trækker alle varige ændringer i en databases tabeller ud i et sammenhængende format, som kan læses uden kendskab til databasens indre tilstand. Tre dele arbejder sammen. Serveren skal skrive detaljer nok i loggen, og det styres af indstillingen `wal_level` : standardværdien, `replica` , rækker til at kopiere data mellem servere, mens `logical` tilføjer det, dekodningen har brug for. En replikeringsslot står for én strøm af ændringer fra én database, som afspilles for en klient i den rækkefølge, de blev foretaget, og den husker, hvor langt klienten er nået. Et output-plugin omsætter loggens interne poster til det format, klienten ønsker.

De færreste skriver selv klienten. Debezium beskriver sig selv som en samling kildekonnektorer til Apache Kafka Connect, hvor hver enkelt læser ændringer fra en bestemt database gennem databasens egne CDC-funktioner. Konnektoren til PostgreSQL læser enten gennem `pgoutput`, som Debeziums dokumentation omtaler som det indbyggede output-plugin i PostgreSQL, eller gennem `decoderbufs`, et plugin, som Debezium-fællesskabet vedligeholder. Debezium kan også køre som et bibliotek i en JVM-applikation eller som Debezium Server, der sender ændringshændelserne videre til systemer som Amazon Kinesis, Google Cloud Pub/Sub og Apache Pulsar.

Brug begrebet i en beslutning

Brug »Hvad er change data capture?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Tal, der lander, hvor der er brug for dem, efter en fast plan, uden at nogen kopierer et regneark ved månedens slutning.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Én infrastrukturplan til 14 afdelinger — 14 forskningsafdelinger på én infrastrukturplan

Én infrastrukturplan til 14 afdelinger Kontakt os →

Hvorfor det betyder noget

Polling ser billigere ud, og for en lille tabel, der sjældent ændres, er det ofte rigtigt. Debeziums dokumentation opregner, hvad læsning af loggen kan, som polling ikke kan. Den fanger hver ændring, hvor en forespørgsel kun ser rækkens tilstand i det øjeblik, den kører, så en række, der blev ændret tre gange mellem to kørsler, kun dukker op én gang. Den fanger sletninger, hvor en slettet række bare mangler i næste resultat. Den kræver ingen ekstra kolonne i datamodellen, som et »Last Updated«-felt, som al kode, der skriver til tabellen, skal huske at udfylde. Den kan tage rækkens tidligere tilstand med og metadata, for eksempel hvilken transaktion der foretog ændringen. Og den leverer ændringerne med meget lille forsinkelse uden den CPU-belastning, som hyppig polling giver: for PostgreSQL og MySQL ligger forsinkelsen ifølge Debezium i millisekundområdet.

Det andet argument er, at databasen og de beskeder, andre systemer modtager, siger det samme. Debezium nævner dobbeltskrivning (dual writes) som den anden fremgangsmåde, den sammenligner sig med: applikationen gemmer en ændring i sin database og sender bagefter en besked om den. Stopper processen mellem de to trin, er databasen og beskedstrømmen uenige, og intet registrerer, hvilken af dem der tager fejl. CDC fjerner den anden skrivning. Applikationen gemmer én gang, og beskeden udledes af det, der faktisk blev gemt. Outbox-mønstret går et skridt videre: applikationen skriver selve beskeden i en outbox-

tabel i samme transaktion som ændringen, og CDC udgiver rækkerne i den tabel. Debeziums Outbox Event Router beskriver formålet som at undgå uoverensstemmelser mellem en tjenestes egen tilstand og de hændelser, som andre tjenester modtager.

Det tredje argument er rækkefølgen. En replikeringsslot udleverer ændringerne i den rækkefølge, de blev foretaget på kildeserveren, samlet efter den transaktion, der foretog dem, som eksemplet nedenfor viser. En forbruger, der anvender dem i den rækkefølge, genskaber den samme tilstand trin for trin. For en ETL-indlæsning gør det den inkrementelle indlæsning ærlig: et natligt job kan tage præcis de ændringer, der er kommet siden sidste kørsel, sletninger medregnet, i stedet for at genindlæse hele tabeller eller stole på en tidsstempelkolonne, som én kodesti glemte at opdatere.

Sådan ser det ud i praksis

Det mindste eksempel, der virker, bruger kun PostgreSQL. Det kræver `wal_level = logical` i serverens konfiguration, hvilket først træder i kraft efter en genstart, og en ledig replikeringsslot; `max_replication_slots` er som standard 10. Sessionen nedenfor kører som superbruger, ligesom dokumentationens eget eksempel, og bruger `test_decoding`, output-pluginnet fra det eksempel, der skriver hver ændring ud som en tekstlinje.

```
-- postgresql.conf: wal_level = logical (træder i kraft efter en genstart)
SELECT * FROM pg_create_logical_replication_slot('shop_cdc', 'test_decoding');

BEGIN;
INSERT INTO orders (id, status) VALUES (42, 'pending');
INSERT INTO outbox (id, event) VALUES (7, 'ordre 42 afgivet');
COMMIT;

SELECT data FROM pg_logical_slot_peek_changes('shop_cdc', NULL, NULL);
-- BEGIN 10301
-- table public.orders: INSERT: id[integer]:42 status[text]:'pending'
-- table public.outbox: INSERT: id[integer]:7 event[text]:'ordre 42 afgivet'
-- COMMIT 10301

SELECT data FROM pg_logical_slot_get_changes('shop_cdc', NULL, NULL);
-- de samme fire rækker; kaldet bruger dem op, så et nyt kald giver ingen

SELECT pg_drop_replication_slot('shop_cdc');
```

Ordren og dens outbox-besked kommer ud sammen, inden for den transaktion, der skrev dem. `pg_logical_slot_peek_changes` læser uden at flytte slotten og giver derfor de samme rækker ved næste kald, mens `pg_logical_slot_get_changes` bruger dem op. En `CREATE TABLE` ville kun give et tomt `BEGIN` og `COMMIT`, fordi logisk dekodning ikke tager DDL med. Den sidste linje er den vigtigste: en slot, der bliver glemt, bliver ved med at holde på serverens ressourcer.

I produktion er det Debezium, der læser. Første gang konnektoren til PostgreSQL starter, tager den et konsistent øjebliksbillede (snapshot) af de tabeller, den opfanger ændringer fra, og streamer derefter ændringer fra det punkt, hvor øjebliksbilledet blev taget. Standardværdien for `snapshot.mode`, `initial`, tager kun øjebliksbilledet, når der ikke findes gemte offsets, så en genstart fortsætter fra den senest registrerede position. Hver hændelse har et `before`- og et `after`-billede af rækken, en `source`-blok, et tidsstempel og en `op`-kode: `c` for oprettelse, `u` for opdatering, `d` for sletning, `r` for en række læst under et øjebliksbillede og `t` for truncate. Efter en sletning sender konnektoren desuden en tombstone, en hændelse med samme nøgle og værdien null, så Kafkas log compaction kan fjerne de tidligere beskeder med den nøgle.

Begge mønstre fra denne artikel går igen på virksomhedens egne sider. Mellem 2022 og 2024 udviklede grundlæggeren 600 ETL/ELT-pipelines i PL/pgSQL på tværs af flere udviklings- og produktionsmiljøer med PostgreSQL, herunder inkrementelle indlæsninger med CDC og tidsstempelbaserede deltaer. Noten At sælge den sidste billet præcis én gang bygger et lynudsalg op omkring en outbox: hvert trin lægger sin besked i outboxen i samme transaktion som ændringen, change data capture udgiver outboxen, og fordi en besked kan komme mere end én gang, er hver forbruger idempotent, med ordren som nøgle.

Hvor det går galt

Intet sætter loft over, hvad en slot må gemme. En slot holder på hvert stykke write-ahead-log, som dens forbruger ikke har bekræftet, og standardindstillingerne i PostgreSQL sætter ingen grænse for det. `max_slot_wal_keep_size` er `-1`, så slots må beholde en ubegrænset mængde log, og `idle_replication_slot_timeout` er `0`, hvilket slår ugyldiggørelsen af ubrugte slots fra. En konnektor, der gik i stå under en mislykket udrulning, kan derfor fylde disken længe efter, at alle har glemt den. Hold øje med `pg_replication_slots` for slots, der ikke længere er aktive, og sæt `max_slot_wal_keep_size` til det, disken kan bære, vel vidende at en forbruger, der kommer længere bagud end det, måske ikke kan fortsætte. Vejen tilbage går

gennem en ny slot, og en ny slot begynder i nuet: Debezium advarer om, at en slot, der slettes og oprettes igen, i det stille kan springe hændelserne imellem over, så tag et nyt øjebliksbillede bagefter.

En stille tabel holder loggen tilbage. Debeziums dokumentation beskriver en mere lumsk variant. En slot hører til én database, og konnektoren rykker kun sin bekræftede position frem, når den ser ændringer, den opfanger. Ændres de opfangede tabeller sjældent, mens andre tabeller eller andre databaser på samme server skriver meget, flytter positionen sig næsten ikke, og loggen bag den bliver ved med at vokse. Løsningen er et hjerteslag (heartbeat): `heartbeat.interval.ms` får konnektoren til at sende heartbeat-hændelser med et fast interval, og `heartbeat.action.query` lader den skrive jævnligt til en lille tabel i den opfangede database. Tabellen skal være med i den publikation (publication), som konnektoren læser, så slotten rykker frem, også når der ellers ikke sker noget.

Forbrugerne går ud fra, at hver ændring kommer én gang. En logisk slot udsender hver ændring én gang under normal drift, men dens position gemmes kun ved checkpoints, så efter et nedbrud kan den vende tilbage til et tidligere punkt og sende de seneste ændringer igen. PostgreSQL-dokumentationen overlader følgerne til klienten. En forbruger, der lægger hvert beløb til en løbende sum, tæller de ændringer to gange. Gør hver forbruger idempotent: anvend hver ændring som en upsert med rækkens primærnøgle som nøgle, eller registrér, hvilke hændelser der allerede er behandlet, så en ny levering ikke ændrer noget.

Opdateringer og sletninger kommer uden de gamle værdier. Hvad en hændelse for en opdatering eller sletning fortæller om den tidligere række, afhænger af tabellens replica identity. Med `REPLICA IDENTITY DEFAULT` indeholder Debeziums hændelser højst de tidligere værdier af primærnøglen, og for en tabel uden primærnøgle sender konnektoren slet ingen hændelser for opdateringer eller sletninger. `REPLICA IDENTITY FULL` lægger alle tidligere kolonner i `before`, men det koster: PostgreSQL-dokumentationen bemærker, at det øger mængden af WAL, især når der køres mange `UPDATE`- og `DELETE`-sætninger. Giv hver opfanget tabel en primærnøgle, og skift kun til `FULL` på de tabeller, hvis forbrugere har brug for de gamle værdier.

Læs mere

- Debezium Features — hvad logbaseret CDC fanger, som polling og dobbeltskrivning overser, plus øjebliksbilleder, filtre og maskering.
- Debezium connector for PostgreSQL — output-plugins, øjebliksbilleder, hændelsernes format, replica identity og diskplads til WAL.
- Logical Decoding Concepts — replikeringsslots, output-plugins, og hvorfor en slot, som ingen bruger, skal slettes.
- Logical Decoding Examples — at oprette en slot, kigge på og bruge ændringer og slette slotten igen.
- Replication (server configuration) — `max_replication_slots`, `max_slot_wal_keep_size` og `idle_replication_slot_timeout`.
- Outbox Event Router — outbox-mønstret, og hvordan Debezium sender outbox-rækker videre til topics.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring et repræsentativt datasæt, de spørgsmål det skal besvare, og den aktuelle flaskehals. Vi kan aftale målinger af kvalitet, gennemløb og gendannelse.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er change data capture?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/change-data-capture/