Engineer

ETL

♻️ å flytte data mellom systemer og omforme dem underveis

Hva det er

Bokstavene i ETL står for extract, transform, load, på norsk uttrekk, transformering og lasting. Sammen beskriver de arbeidet med å kopiere data ut av systemene som skaper dem, gi dem den formen og betydningen et annet system forventer, og skrive dem inn der, som oftest i et datavarehus. En nær variant, ELT, beholder de tre stegene, men laster rådataene først og transformerer dem inne i målsystemet, og mange team skriver ETL/ELT for å dekke begge.

Hver bokstav er en egen jobb. Uttrekket leser fra en kilde: en hel tabell, en eksportfil, et API, eller bare radene som er endret siden forrige kjøring, funnet gjennom et tidsstempel som `updated_at` eller gjennom change data capture (CDC), som leser databasens egen oversikt over hver endring. Transformeringen er der dataene renses og avstemmes: tekst blir til datoer og tall, enheter og tidssoner gjøres ensartede, duplikater fjernes, koder slås opp mot referansetabeller, og resultatet formes om til modellen målsystemet bruker, f.eks. fakta- og dimensjonstabeller. Lastingen skriver resultatet, enten som en masseinnsetting i en tom tabell eller som en sammenslåing (merge) som oppdaterer radene som allerede finnes, og legger til de nye.

Rekkefølgen på de to siste stegene avgjør hvor arbeidet gjøres. Klassisk ETL transformerer på en egen server før noe når målsystemet, og det ga mening den gang datalagring og regnekraft i et datavarehus var knappe og dyre. ELT legger rådataene i mellomtabeller (staging-tabeller) inne i målsystemet og kjører transformeringen der, i SQL. Råkopien blir værende, så når en regel endres, kan transformeringen kjøres på nytt over hele historikken uten et nytt uttrekk. Prisen er at alt kilden sendte, også kolonner ingen ba om, nå ligger i datavarehuset og må forvaltes der.

PostgreSQL dekker hvert steg med egne verktøy. `COPY` flytter rader mellom en tabell og en fil i tekst-, CSV- eller binærformat og er den raskeste veien inn. For å lese en fil på selve serveren må du være superbruker eller ha rollen `pg_read_server_files`, mens psql-kommandoen `\copy` i stedet leser filen på klienten. `INSERT ... ON CONFLICT` og `MERGE` fletter nye rader inn i dem som allerede finnes. Logisk dekoding gjør write-ahead-loggen (WAL) om til en strøm av radendringer for CDC, og en prosedyre i PL/pgSQL kan holde transformeringen tett ved dataene den endrer.

Bruk begrepet i en beslutning

Bruk «Hva er ETL?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Tall som kommer fram dit de trengs, til faste tider, uten at noen må kopiere et regneark ved månedsslutt.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Én infrastrukturplan for 14 avdelinger — 14 forskningsavdelinger på én felles infrastrukturplan

Én infrastrukturplan for 14 avdelinger Ta kontakt →

Hvorfor det er viktig

De fleste spørsmål som er verdt å stille om en virksomhet, krever data fra mer enn ett system. Ordrene ligger i én database, betalingene i en eksport fra en leverandør, kundeopplysningene i et kundesystem (CRM). Inntil noe flytter dem til ett sted under ett sett regler, blir hvert svar satt sammen for hånd, og fremgangsmåten varierer med personen som gjør jobben. ETL er det som flytter dem, og et datavarehus er bare så riktig som lastingene som fyller det.

Transformeringssteget er også der en virksomhet skriver ned hva dataene betyr. Hvilke rader som teller som et salg, hvilken dag en betaling ved midnatt hører til, hvilken post som vinner når to kunder ser ut til å være den samme: avgjort én gang i lastingen gjelder svaret for hver rapport lenger ut i kjeden. Overlatt til hver enkelt rapport glir svarene fra hverandre, og to dashbord viser to forskjellige summer.

Måten en lasting henter ut data på, avgjør hvordan den vokser. En full omlasting kopierer alt hver gang. Den er enkel og retter seg selv, siden gårsdagens feil blir overskrevet i natt, men kostnaden stiger med den samlede datamengden. En inkrementell lasting flytter bare det som er endret, så der stiger kostnaden i stedet med endringstakten. Den må huske hvor forrige kjøring stoppet, merke

seg rader som er slettet i kilden, og ta imot rader som kommer sent med gamle tidsstempler. De fleste systemer begynner med fulle omlastinger og bytter over, tabell for tabell, når natten ikke lenger er lang nok.

Den løpende kostnaden er at lastinger er kode som kjører uten tilsyn, som regel om natten, mot kilder som endres uten forvarsel. De trenger logging som forteller hva som kjørte og hvor mange rader som ble flyttet, et driftsvarsel når en kjøring stopper, og framfor alt én egenskap: en kjøring som gjentas med de samme inndataene, skal etterlate målet nøyaktig slik én kjøring ville gjort. En lasting med den egenskapen er idempotent, og den kan kjøres på nytt etter hvilken som helst feil uten at noen må sjekke noe først.

Slik ser det ut i praksis

Fragmenterte pipeliner forsinket forretningskritisk rapportering på tvers av flere utviklings- og produksjonsmiljøer i PostgreSQL, og 600 ETL/ELT-pipeliner ble skrevet i PL/pgSQL for å erstatte dem: ETL i en skala på flere hundre rutinemessige lastinger. Primær- og fremmednøkkelbegrensninger ivaretok referanseintegriteten under transformasjonene, unike og sammensatte indekser støttet join-operasjonene, og unntakshåndtering med transaksjonskontroll rullet en kjøring tilbake når den gikk galt. Inkrementelle lastinger brukte CDC og tidsstempelbaserte deltaer, og logging og revisjon av kjøringer støttet sporing og feilsøking. Leveransen omfatter transformasjonslogikk, integritetsregler, inkrementell lasting og kjøringslogging; dokumentasjonen fastslår ingen sammenlignende ytelsesmåling eller beregning av feilrate.

En migrering mellom databaser er ETL som kjører én gang. Å flytte en kartapplikasjon fra Microsoft SQL Server til PostgreSQL med PostGIS betydde å føre over 50 millioner romlige poster gjennom ETL-pipeliner med transformasjonssteg som gjorde om hver eneste koordinat til referansesystemet EPSG:4326. Romlige spørringer fikk ti ganger bedre ytelse av migreringen som helhet, der ETL-jobben virket sammen med GiST- og SP-GiST-indekser og en justert konfigurasjon på den nye siden: svartiden, i snitt 2,5 sekunder før, havnet under 250 millisekunder, og CPU-belastningen i backend falt med 65 %.

Over en nettverksgrense viser seks ETL/ELT-pipeliner alle bokstavene i navnet i arbeid. Uttrekket besto av planlagte nedlastinger av komprimerte arkiver fra en server som bare kunne nås gjennom et IPsec-VPN, med Bash- og Python-skript som sorterte Parquet-, CSV- og .bak-filene i dem etter type og skjema, og .bak-sikkerhetskopiene ble gjenopprettet i en lokal MSSQL Server-instans. Dataene

ble lastet inn i staging-skjemaer i MSSQL og PostgreSQL med bcp, psql eller SSIS, alt etter formatet, så renset, normalisert og beriket av prosedyrer i PL/pgSQL og T-SQL, og til slutt komprimert med gzip og fraktet gjennom en Google Cloud Storage-bucket inn i Google BigQuery. Arbeid som hadde tatt minst tre timer for hånd, kjørte på under 20 minutter fra ende til ende, og synkroniseringen gikk fra ukentlig til daglig.

Hvor det går galt

En kjøring som startes på nytt, laster de samme radene to ganger. En jobb setter inn med vanlig `INSERT`, stopper halvveis på en nettverksfeil, og jobbplanleggeren kjører den på nytt fra starten. Radene som kom inn første gang, ligger nå der to ganger, og hver sum som bygger på dem, blir for høy. Gi målet en nøkkel som identifiserer en rad i forretningstermer, og flett inn partiet med `INSERT ... ON CONFLICT DO UPDATE`, som PostgreSQL dokumenterer som en atomisk innsetting eller oppdatering også ved høy samtidighet, eller med `MERGE`. Én hake finnes: en enkelt `ON CONFLICT DO UPDATE`-setning nekter å endre den samme raden to ganger og gir et kardinalitetsbrudd (cardinality violation), så et parti med gjentatte nøkler må først reduseres til én rad per nøkkel.

```
INSERT INTO meter_reading (meter_id, read_at, kwh)
SELECT DISTINCT ON (meter_id, read_at) meter_id, read_at, kwh
FROM staging_meter_reading
ORDER BY meter_id, read_at, received_at DESC
ON CONFLICT (meter_id, read_at)
DO UPDATE SET kwh = EXCLUDED.kwh;
```

En masselasting stopper nær slutten og etterlater disken fullere enn før. `COPY FROM` skriver rader inn i tabellen etter hvert som den leser dem. Når den stopper på en dårlig rad, blir radene den allerede har skrevet, liggende igjen i slettet tilstand, usynlige for spørringer, men fortsatt på disken til `VACUUM` tar plassen tilbake, og en feil sent i en stor fil kan sløse bort mye plass. PostgreSQL 17 la til `ON_ERROR ignore`, som hopper over rader i tekst- eller CSV-inndata der verdiene ikke lar seg konvertere til kolonnens type, og PostgreSQL 18 la til `REJECT_LIMIT` for å sette et tak på hvor mange rader den kan hoppe over. Det dekker bare typekonvertering; en rad som bryter en begrensning, stopper fortsatt kommandoen. Last inn i en mellomtabell først, sjekk dataene der, og flytt de gode radene inn i den virkelige tabellen i én transaksjon, så holdes en dårlig fil unna dataene folk leser.

Change data capture fyller disken. Logisk dekoding leser endringer gjennom et replikeringsspor (replication slot), og et slikt spor holder på hver bit av write-ahead-loggen som mottakeren fortsatt trenger, enten en mottaker er koblet til eller ikke. Når leseren stopper, hoper loggen seg opp til disken er full, og i ytterste konsekvens slår serveren seg av for å hindre omløp av transaksjons-ID-ene (transaction ID wraparound). PostgreSQL-dokumentasjonen sier det rett ut: et spor som ikke lenger trengs, skal fjernes. Etter et krasj kan et spor også sende nylige endringer én gang til, så mottakeren må tåle duplikater, og det er den samme egenskapen ved nye kjøringer igjen.

Morgenens første rapport er treg etter en stor lasting. Spørringsplanleggeren velger hvordan en spørring skal kjøres, ut fra statistikk om hver tabell, og en masselasting endrer dataene under den statistikken. Autovacuum kan oppdatere statistikken på egen hånd, men ikke alltid før den første rapporten kjører. Å kjøre `ANALYZE` som siste steg i lastingen løser det. For svært store lastinger inn i en eksisterende tabell legger PostgreSQL-veiledningen om å fylle en database til at det kan lønne seg å fjerne indekser og fremmednøkler og opprette dem på nytt etterpå, fordi det går raskere å bygge dem samlet enn å oppdatere dem rad for rad. Da gir du fra deg feilsjekkingen mens de er borte, så en unik indeks fortjener en ekstra tanke før den fjernes.

Les mer

- COPY — kommandoen for masselasting, formatene og opsjonene, og hva en mislykket lasting etterlater seg.
- Populating a Database — rådene fra PostgreSQL for store lastinger, fra én transaksjon og `COPY` til å bygge indekser på nytt og kjøre `ANALYZE` etterpå.
- MERGE — innsetting, oppdatering og sletting i én setning, styrt av hvordan hver kilderad passer med målet.
- INSERT — leddet `ON CONFLICT`, den atomiske upserten og regelen mot å røre den samme raden to ganger.
- Logical Decoding Concepts — hvordan endringer leses fra write-ahead-loggen, og hvorfor et replikeringsspor som ikke brukes, må fjernes.

Ordliste

Velg ut fra deres faktiske rammer

Ta med et representativt datasett, spørsmålene det skal besvare og dagens flaskehals. Vi kan avtale målinger av kvalitet, gjennomstrømning og gjenoppretting.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er ETL?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/etl/