Engineer

ETL

🔄 at flytte data mellem systemer og omforme dem undervejs

Hvad det er

ETL, en forkortelse af extract, transform, load, er arbejdet med at kopiere data ud af de systemer, der skaber dem, give dem den form og betydning, et andet system forventer, og skrive dem dertil, som regel ind i et datawarehouse. En nær variant, ELT, beholder de tre trin, men indlæser rådata først og transformerer dem inde i målet, og mange skriver ETL/ELT for at dække begge.

Hvert bogstav er et selvstændigt stykke arbejde. Udtrækket læser fra en kilde: en hel tabel, en eksportfil, et API eller kun de rækker, der er ændret siden sidste kørsel, fundet via et `updated_at` -tidsstempel eller via change data capture (CDC), som læser databasens egen log over hver ændring. Transformationen er der, hvor data renses og afstemmes: tekst bliver til datoer og tal, enheder og tidszoner gøres ensartede, dubletter fjernes, koder slås op i referencetabeller, og resultatet formes om til den model, målet bruger, for eksempel fakta- og dimensionstabeller. Indlæsningen skriver resultatet, enten som en masseindsættelse i en tom tabel eller som en fletning, der opdaterer de rækker, der allerede findes, og tilføjer de nye.

Rækkefølgen af de to sidste trin afgør, hvor arbejdet foregår. Klassisk ETL transformerer på en separat server, før noget når målet, og det gav mening, dengang lagerplads og regnekraft i et datawarehouse var knappe og dyre. ELT lægger rådata i staging-tabeller inde i målet og kører transformationen dér i SQL. Råkopien bliver liggende, så når en regel ændres, kan transformationen køres igen over hele historikken uden et nyt udtræk. Prisen er, at alt, hvad kilden sendte, også kolonner, ingen har bedt om, nu ligger i datawarehouset og skal styres dér.

PostgreSQL har et værktøj til hvert trin. `COPY` flytter rækker mellem en tabel og en fil i tekst-, CSV- eller binært format og er den hurtigste vej ind. At læse en fil på serveren kræver superbrugerrettigheder eller rollen `pg_read_server_files`, mens psql-kommandoen `\copy` læser filen på klienten i stedet. `INSERT ... ON CONFLICT` og `MERGE` fletter nye rækker ind i de eksisterende. Logisk dekodning gør write-ahead-loggen til en strøm af rækkeændringer til CDC, og en PL/pgSQL-procedure kan holde transformationen tæt på de data, den ændrer.

Brug begrebet i en beslutning

Brug »Hvad er ETL?« 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

De fleste spørgsmål, der er værd at stille i en virksomhed, kræver data fra mere end ét system. Ordrerne ligger i én database, betalingerne i en udbyders eksport, kunderne i et CRM. Indtil noget samler dem ét sted under ét sæt regler, bliver hvert svar stykket sammen i hånden, og svaret afhænger af, hvem der stykker det sammen. ETL er det, der samler dem, og en fejl i indlæsningen går videre til hver rapport, der bygger på den.

Transformationen er også stedet, hvor en virksomhed skriver ned, hvad dens data betyder. Hvilke rækker tæller som et salg, hvilken dag en betaling ved midnat hører til, hvilken post der vinder, når to kunder ligner hinanden: afgøres det én gang i indlæsningen, gælder svaret for hver eneste rapport bagefter. Overlades det til hver rapport, glider svarene fra hinanden, og to dashboards viser to totaler.

Måden, en indlæsning udtrækker på, bestemmer, hvordan den vokser. En fuld genindlæsning kopierer alt hver gang. Den er enkel og retter sig selv, fordi gårsdagens fejl bliver overskrevet i nat, men prisen stiger med den samlede datamængde. En inkrementel indlæsning flytter kun det, der er ændret, så prisen i stedet stiger med ændringstakten. Den skal huske, hvor sidste kørsel stoppede,

opdage rækker, der er slettet i kilden, og tage imod rækker, der kommer sent med gamle tidsstempler. De fleste systemer begynder med fulde genindlæsninger og skifter tabel for tabel, når natten ikke længere er lang nok.

Den løbende pris er, at indlæsninger er kode, der kører uden opsyn, som regel om natten, mod kilder, der ændrer sig uden varsel. De har brug for logning, der viser, hvad der kørte, og hvor mange rækker der flyttede sig, alarmer, når en kørsel stopper, og frem for alt én egenskab: en kørsel, der gentages med samme input, skal efterlade målet præcis, som én kørsel ville. En indlæsning med den egenskab kan startes igen efter enhver fejl, uden at nogen behøver at tjekke først.

Sådan ser det ud i praksis

Fragmenterede pipelines forsinkede forretningskritisk rapportering på tværs af flere PostgreSQL-udviklings- og produktionsmiljøer, og 600 ETL/ELT-pipelines blev udviklet i PL/pgSQL for at afløse dem: ETL i en skala af hundredvis af rutinemæssige indlæsninger. Primary- og foreign key-constraints bevarede den referentielle integritet under transformationer, unikke og sammensatte indeks understøttede joins, og exception handling med transaktionskontrol rullede en kørsel tilbage, når noget gik galt. Inkrementelle indlæsninger brugte CDC og tidsstempelbaserede deltaer, og logning og revision af kørsler understøttede sporing og fejlsøgning. Det leverede arbejde omfatter transformationslogik, constraints, inkrementel indlæsning og logning af kørsler; beskrivelsen dokumenterer ikke et sammenlignende benchmark eller en beregning af fejlprocenten.

En migrering er ETL, der kører én gang. At flytte en kortapplikation fra Microsoft SQL Server til PostgreSQL med PostGIS betød, at over 50 millioner spatiale records gik gennem ETL-pipelines, hvis transformationstrin konverterede hver koordinat til referencesystemet EPSG:4326. Migreringen som helhed, med GiST- og SP-GiST-indeks og en tunet konfiguration på den nye side, gjorde de spatiale forespørgsler ti gange hurtigere: hvor et svar før tog 2,5 sekunder i gennemsnit, kom det nu på under 250 millisekunder, og backendens CPU-belastning faldt med 65 %.

På tværs af en netværksgrænse er hvert bogstav i navnet i brug i seks ETL/ELT-pipelines. Udtrækket bestod af planlagte downloads af komprimerede arkiver fra en server, der kun kunne nås gennem en IPsec VPN, hvor Bash- og Python-scripts pakkede Parquet-, CSV- og .bak-filerne ud og klassificerede dem, og .bak-filerne

blev gendannet i en lokal MSSQL Server-instans. Dataene blev lagt i staging-skemaer i MSSQL og PostgreSQL med bcp, psql og SSIS, renset og beriget af modulære PL/pgSQL- og T-SQL-procedurer og til sidst lagt op i Google BigQuery med bq CLI og Python-baseret indtagelse, der også tog sig af skemamappingen. Det, der havde været mindst tre timers manuelt arbejde, tog under 20 minutter fra ende til anden, og synkroniseringen gik fra ugentlig til daglig.

Hvor det går galt

En kørsel, der startes igen, indlæser de samme rækker to gange. Et job indsætter med almindelig `INSERT`, stopper halvvejs på en netværksfejl, og scheduleren kører det igen fra begyndelsen. De rækker, der nåede ind første gang, ligger der nu to gange, og hver total bygget på dem er for høj. Giv målet en nøgle, der identificerer en række i forretningens termer, og flet batchen ind med `INSERT ... ON CONFLICT DO UPDATE`, som PostgreSQL dokumenterer som en atomisk indsæt-eller-opdatér selv ved høj samtidighed, eller med `MERGE`. Der er én hage: en enkelt `ON CONFLICT DO UPDATE`-sætning nægter at ændre den samme række to gange og melder en cardinality violation, så en batch med gentagne nøgler først skal skæres ned til én række pr. nøgle.

```
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 masseindlæsning stopper tæt på slutningen og efterlader disken mere fyldt end før. `COPY FROM` skriver rækkerne ind i tabellen undervejs. Stopper den på en dårlig række, bliver de allerede skrevne rækker liggende i slettet tilstand, usynlige for forespørgsler, men stadig på disken, indtil `VACUUM` frigør pladsen, og et stop sent i en stor fil kan koste meget plads. PostgreSQL 17 tilføjede `ON_ERROR ignore`, der springer rækker over i tekst- eller CSV-input, hvis værdier ikke lader sig konvertere til kolonnens type, og PostgreSQL 18 tilføjede `REJECT_LIMIT`, der sætter et loft over, hvor mange der må springes over. Det dækker kun typekonvertering; en række, der bryder en constraint, stopper stadig kommandoen. Indlæs i en staging-tabel først, tjek den dér, og flyt de gode rækker over i den rigtige tabel i én transaktion, så holdes en dårlig fil væk fra de data, folk læser.

Change data capture fylder disken. Logisk dekodning læser ændringer gennem en replikeringsslot, og en slot gemmer hvert stykke write-ahead-log, som dens forbruger stadig mangler, uanset om en forbruger er forbundet. Holder læseren op, hober loggen sig op, til disken er fuld, og i ekstreme tilfælde lukker serveren ned for at forhindre transaction ID wraparound. PostgreSQL-dokumentationen siger det ligeud: en slot, der ikke længere er brug for, skal droppes.
Efter et nedbrud kan en slot også sende de seneste ændringer en gang til, så forbrugeren skal kunne tåle dubletter, og det er den samme genkørselsegenskab igen.

Morgenens første rapport er langsom efter en stor indlæsning. Planneren vælger, hvordan en forespørgsel skal køres, ud fra statistik om hver tabel, og en masseindlæsning ændrer dataene under statistikken. Autovacuum kan opdatere den af sig selv, men ikke altid før den første rapport kører. At køre `ANALYZE` som sidste trin i indlæsningen løser det. Ved meget store indlæsninger i en eksisterende tabel tilføjer PostgreSQL-vejledningen om at fylde en database, at det kan betale sig at droppe indeks og fremmednøgler og oprette dem igen bagefter, fordi de bygges hurtigere samlet end række for række. Til gengæld forsvinder fejlkontrollen, mens de er væk, så et unikt indeks fortjener en ekstra overvejelse, før det fjernes.

Læs mere

- COPY — kommandoen til masseindlæsning, dens formater og indstillinger, og hvad en afbrudt indlæsning efterlader.
- Populating a Database — PostgreSQL's råd om store indlæsninger, fra én transaktion og `COPY` til at genopbygge indeks og køre `ANALYZE` bagefter.
- MERGE — indsæt, opdatér og slet i én sætning, styret af, hvordan hver kilderække matcher målet.
- INSERT — `ON CONFLICT` -klausulen, dens atomiske upsert og dens regel mod at røre den samme række to gange.
- Logical Decoding Concepts — hvordan ændringer læses fra write-ahead-loggen, og hvorfor en ubrugt replikeringsslot skal væk.



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 ETL?« 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/etl/