Engineer

ETL

🔄 att flytta data mellan system och omforma dem på vägen

Vad det är

ETL, kort för extract, transform, load, alltså extrahera, transformera och ladda, är arbetet med att kopiera data ut ur de system som skapar dem, ge dem den form och betydelse som ett annat system förväntar sig och skriva in dem där, oftast i ett analysdatalager. En nära variant, ELT, behåller de tre stegen men laddar först rådata och transformerar dem inuti målsystemet, och många team skriver ETL/ELT för att täcka båda.

Varje bokstav är ett eget jobb. Extraheringen läser från en källa: en hel tabell, en exportfil, ett API, eller bara de rader som har ändrats sedan förra körningen, som hittas genom en tidsstämpel i `updated_at` eller genom change data capture (CDC), som läser databasens egen förteckning över varje ändring. Transformeringen är där data rensas och stäms av: text görs om till datum och tal, enheter och tidszoner görs enhetliga, dubbletter tas bort, koder slås upp mot referenstabeller, och resultatet formas om till den modell som målet använder, t.ex. fakta- och dimensionstabeller. Laddningen skriver resultatet, antingen som en massinsättning i en tom tabell eller som en sammanslagning som uppdaterar de rader som redan finns och lägger till de nya.

Ordningen mellan de två sista stegen avgör var arbetet körs. Klassisk ETL transformerar på en separat server innan något når målet, vilket var rimligt när lagring och beräkningskraft i analysdatalagret var knappa och dyra. ELT lägger rådata i mellanlagringstabeller, så kallade staging-tabeller, inuti målet och kör transformeringen där i SQL. Råkopian finns kvar, så när en regel ändras kan transformeringen köras om över historiken utan att den behöver extraheras en gång till. Priset är att allt som källan skickade, även kolumner som ingen bad om, nu bor i analysdatalagret och måste förvaltas där.

PostgreSQL täcker varje steg med egna verktyg. `COPY` flyttar rader mellan en tabell och en fil i text-, CSV- eller binärformat och är den snabbaste vägen in. Att läsa en fil på servern kräver superanvändarrättigheter eller rollen `pg_read_server_files`, medan psql-kommandot `\copy` i stället läser filen på klienten. `INSERT ... ON CONFLICT` och `MERGE` fogar in nya rader i befintliga. Logisk avkodning gör om transaktionsloggen, write-ahead log, till en ström av radändringar för CDC, och en procedur i PL/pgSQL kan hålla transformeringen nära de data den ändrar.

Använd begreppet i ett beslut

Använd ”Vad är ETL?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Siffror som kommer fram dit de behövs, på fasta tider, utan att någon kopierar ett kalkylblad i slutet av månaden.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. En infrastrukturplan för 14 avdelningar — 14 forskningsavdelningar under en och samma infrastrukturplan

En infrastrukturplan för 14 avdelningar Hör av er →

Varför det spelar roll

De flesta frågor som är värda att ställa till en verksamhet behöver data från mer än ett system. Ordrar ligger i en databas, betalningar i en export från en leverantör, kunduppgifter i ett CRM-system. Tills något flyttar dem till ett och samma ställe under ett och samma regelverk sätts varje svar ihop för hand, och tillvägagångssättet varierar med personen som gör jobbet. ETL är det där något, och ett analysdatalager är bara så korrekt som de laddningar som fyller det.

Transformeringssteget är också där en verksamhet skriver ned vad dess data betyder. Vilka rader som räknas som en försäljning, vilken dag en betalning vid midnatt hör till, vilken post som vinner när två kunder matchar: avgjort en gång i laddningen gäller svaret för varje rapport längre ned i kedjan. Lämnas frågan åt varje rapport glider svaren isär, och två dashboards visar två olika summor.

Hur en laddning extraherar avgör hur den växer. En full omladdning kopierar allt varje gång. Den är enkel och rättar sig själv, eftersom gårdagens misstag skrivs över i natt, men kostnaden växer med den totala datamängden. En inkrementell laddning flyttar bara det som har ändrats, så kostnaden växer i stället med ändringstakten. Den måste komma ihåg var förra körningen slutade, märka rader

som har raderats i källan och ta emot rader som kommer sent med gamla tidsstämplar. De flesta system börjar med fulla omladdningar och byter tabell för tabell när natten inte längre räcker till.

Den löpande kostnaden är att laddningar är kod som körs utan tillsyn, oftast på natten, mot källor som ändras utan förvarning. De behöver loggning som säger vad som kördes och hur många rader som flyttades, larm när en körning stannar, och framför allt en egenskap: en körning som upprepas med samma indata ska lämna målet exakt som en enda körning skulle ha gjort. En laddning med den egenskapen kan köras om efter vilket avbrott som helst utan att någon först behöver undersöka läget.

Så ser det ut i praktiken

600 pipelines, en migrering som kördes en enda gång och sex laddningar genom ett VPN: samma tre steg återkommer här i mycket olika skala.

Fragmenterade pipelines försenade affärskritisk rapportering i flera utvecklings- och produktionsmiljöer för PostgreSQL, och 600 ETL/ELT-pipelines skrevs i PL/pgSQL för att ersätta dem: ETL i en skala av hundratals rutinmässiga laddningar. Villkor för primärnycklar och främmande nycklar skyddade referensintegriteten under transformeringarna, unika och sammansatta index stödde JOIN-operationerna, och undantagshantering med transaktionsstyrning rullade tillbaka en körning när något gick snett. Inkrementella laddningar använde CDC och tidsstämpelbaserade deltan, och loggning och granskning av körningar stödde spårning och felsökning. Det levererade arbetet omfattar transformeringslogik, integritetsvillkor, inkrementell laddning och loggning av körningar; beskrivningen fastställer inte något jämförande prestandatest eller någon beräkning av felfrekvensen.

En migrering är ETL som körs en gång. Att flytta en kartapplikation från Microsoft SQL Server till PostgreSQL med PostGIS innebar att föra över 50 miljoner spatiala poster genom ETL-pipelines vars transformeringssteg räknade om varje koordinat till referenssystemet EPSG:4326. På den nya sidan fanns index av typerna GiST och SP-GiST och en finjusterad konfiguration. Migreringen som helhet gav tio gånger bättre prestanda för spatiala databasfrågor och 65 % lägre processorbelastning i backend, och i genomsnitt kom svaren på under 250 millisekunder, mot 2,5 sekunder före flytten.

Över en nätverksgräns visar sex ETL/ELT-pipelines varje bokstav i namnet i arbete. Extraheringen bestod av schemalagda nedladdningar av komprimerade arkiv från en server som bara gick att nå genom ett IPsec-VPN, där skript i Bash och Python sorterade filerna i Parquet, CSV och .bak efter typ och schema, och säkerhetskopiorna i .bak återställdes till en lokal MSSQL-instans. Data lästes in i staging-scheman i MSSQL och PostgreSQL med bcp, psql eller SSIS beroende på formatet, rensades, normaliserades och berikades av procedurer i PL/pgSQL och T-SQL, komprimerades sedan med gzip och fördes via en bucket i Google Cloud Storage till Google BigQuery. Arbete som hade tagit minst tre timmar för hand gick på under 20 minuter från början till slut, och synkroniseringen gick från veckovis till daglig.

Var det går fel

En omkörning laddar samma rader två gånger. Ett jobb gör vanliga `INSERT`, stannar halvvägs på ett nätverksfel, och schemaläggaren kör det igen från början. Rader som kom in första gången finns nu två gånger, och varje summa som bygger på dem blir för hög. Ge målet en nyckel som identifierar en rad i verksamhetens termer och foga in batchen med `INSERT ... ON CONFLICT DO UPDATE`, som PostgreSQL dokumenterar som en atomisk infogning eller uppdatering även vid hög samtidighet, eller med `MERGE`. En hake finns: en enskild sats med `ON CONFLICT DO UPDATE` vägrar ändra samma rad två gånger och ger ett kardinalitetsfel, så en batch med upprepade nycklar måste först reduceras till en rad per nyckel.

```
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 massladdning stannar nära slutet och lämnar disken fullare än förut. `COPY FROM` skriver rader till tabellen allteftersom. När kommandot stannar på en dålig rad blir de raderna kvar i raderat tillstånd, osynliga för databasfrågor men kvar på disken tills `VACUUM` återvinner dem, och ett avbrott sent i en stor fil kan slösa mycket utrymme. PostgreSQL 17 lade till `ON_ERROR ignore`, som hoppar över rader i text- eller CSV-indata vars värden inte går att konvertera till kolumnens typ, och PostgreSQL 18 lade till `REJECT_LIMIT` för att begränsa hur många rader som får hoppas över. Det täcker bara typkonvertering; en rad som bryter mot ett

villkor stoppar fortfarande kommandot. Att först ladda in i en staging-tabell, kontrollera den där och flytta de bra raderna till den riktiga tabellen i en enda transaktion håller en dålig fil borta från de data som folk läser.

Change data capture fyller disken. Logisk avkodning läser ändringar genom en replikeringsslot, och en slot behåller varje del av transaktionsloggen som dess konsument fortfarande behöver, oavsett om en konsument är ansluten eller inte. När läsaren stannar hopar sig loggen tills disken är full, och i extrema fall stänger servern av sig själv för att förhindra wraparound av transaktions-id. Dokumentationen för PostgreSQL är tydlig: en slot som inte längre behövs ska tas bort. Efter en krasch kan en slot också skicka nyliga ändringar en gång till, så konsumenten måste tåla dubbletter, alltså samma krav på säkra omkörningar som ovan.

Morgonens första rapport är långsam efter en stor laddning. Planeraren väljer hur en databasfråga ska köras utifrån statistik om varje tabell, och en massladdning ändrar datan under statistiken. Autovacuum kan uppdatera statistiken på egen hand, men inte alltid innan den första rapporten körs. Att köra `ANALYZE` som laddningens sista steg löser det. För mycket stora laddningar till en befintlig tabell tillägger guiden från PostgreSQL om att fylla en databas att det kan löna sig att ta bort index och främmande nycklar och skapa dem igen efteråt, eftersom det går snabbare att bygga dem i ett svep än att uppdatera dem rad för rad. Under tiden kontrolleras villkoren inte, så ett unikt index förtjänar en extra tanke innan det tas bort.

Läs mer

- COPY — kommandot för massladdning, dess format och inställningar, och vad en misslyckad laddning lämnar efter sig.
- Populating a Database — råden från PostgreSQL för stora laddningar, från en enda transaktion och `COPY` till att bygga om index och köra `ANALYZE` efteråt.
- MERGE — infogning, uppdatering och borttagning i en och samma sats, styrd av hur varje rad i källan matchar målet.
- INSERT — satsdelen `ON CONFLICT`, dess atomiska upsert och regeln mot att röra samma rad två gånger.
- Logical Decoding Concepts — hur ändringar läses ur transaktionsloggen och varför en oanvänd replikeringsslot måste bort.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med ett representativt dataset, frågorna det ska besvara och dagens flaskhals. Vi kan avtala mått på kvalitet, genomströmning och återställning.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är ETL?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/etl/