

Engineer

Forbindelsespulje

et lager af åbne databaseforbindelser til genbrug



Hvad det er

En forbindelsespulje er et sæt åbne databaseforbindelser, som en applikation holder på og låner ud igen, i stedet for at åbne en ny forbindelse til hvert stykke arbejde og lukke den bagefter. Koden låner en forbindelse fra puljen, kører sine sætninger på den og giver den tilbage, og den næste, der skal bruge en, får den samme forbindelse, allerede åben. SQLAlchemys dokumentation beskriver pooling som en standardteknik til at holde langlivede forbindelser i hukommelsen, så de kan genbruges effektivt. Samme side giver puljen en opgave mere: den styrer, hvor mange forbindelser applikationen må bruge på samme tid, så et pludseligt tryk venter på en ledig forbindelse i stedet for at åbne flere.

En pulje kan ligge to steder. Inde i applikationen er den en del af databasebiblioteket: hver `Engine` i SQLAlchemy har en, og Django kan holde en i hver proces til PostgreSQL. Uden for applikationen er den et selvstændigt program, der står mellem klienterne og databaseserveren. PgBouncer er sådan en pooler til PostgreSQL: applikationerne forbinder sig til den, som om den var databasen, og den låner dem serverforbindelser fra sine egne puljer. De to kan kombineres, og Djangos databasereference har et afsnit om at køre bag en transaktionspooler som PgBouncer.

Hvorfor en forbindelse er værd at beholde, ses på serversiden. PostgreSQL giver hver klient sin egen proces: en overordnet proces, postmaster, lytter efter indkommende forbindelser og starter en ny serverproces for hver anmodning om forbindelse, og hver klient er forbundet til præcis én af de processer. At åbne en forbindelse betyder altså at starte en proces på databaseserveren, og en pulje betaler den pris én gang og genbruger så resultatet.

Brug begrebet i en beslutning

Brug »Hvad er en forbindelsespulje?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. At finde de forespørgsler og manglende indeks bag et system, der er blevet tungere hvert år, og rette dem, der betyder noget.

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. 30 dages eldata på 180 ms — 180 ms til at aggregere en måneds måleraflæsninger

30 dages eldata på 180 ms Kontakt os →

Hvorfor det betyder noget

Den første grund er prisen for at forbinde. Djangos dokumentation siger det ligeud: en vedvarende forbindelse sparer arbejdet med at oprette en ny i hver HTTP-forespørgsel. Standardindstillingen gør dog det modsatte. `CONN_MAX_AGE` er 0, som lukker forbindelsen ved slutningen af hver forespørgsel; et positivt antal sekunder holder den åben til genbrug, og `None` holder den åben uden grænse. Det er den enkleste form for genbrug, én forbindelse pr. tråd, som lever længere end forespørgslen, og en pulje er den generelle form.

Den anden grund er grænsen på serveren. PostgreSQLs `max_connections` er typisk 100 som standard, kan være lavere, hvis kernens indstillinger ikke kan bære det, og kan kun ændres ved serverstart. Tre af pladserne er som standard reserveret til superbrugere. Det er heller ikke gratis at hæve tallet: PostgreSQL dimensionerer visse ressourcer, blandt andet delt hukommelse, direkte efter `max_connections`. Uden en pulje holder hver worker sin egen forbindelse, og Djangos dokumentation siger, hvad det fører til: fordi hver tråd har sin egen forbindelse, skal databasen kunne tage imod mindst lige så mange samtidige forbindelser, som der er worker-tråde. En pulje gør det til et tal, holdet selv vælger og kan lægge sammen.

Den tredje grund er hygiejne. En pulje er det ene sted, hver forbindelse passerer mellem to lånere, så det er her, en forbindelse bliver ryddet op og testet. Når en forbindelse kommer tilbage, kalder SQLAlchemys pulje `rollback()` på den, hvilket frigiver al transaktionstilstand, låse medregnet, så den næste låner ikke arver en halvfærdig transaktion. I PgBouncers standardtilstand med sessionspuljer køres dens `server_reset_query`, som standard `DISCARD ALL`, når en serverforbindelse frigives, og før en anden klient får den. Og en pulje kan teste en forbindelse, før den låner den ud, hvilket betyder noget for forbindelser, der har ligget ubrugt et stykke tid.

Sådan ser det ud i praksis

I SQLAlchemy 2.1 får hver `Engine` som standard en `QueuePool`, med SQLites database i hukommelsen som undtagelse. Dens `pool_size` er 5, og dens `max_overflow` er 10, så én engine kan have op til 15 forbindelser på én gang, og en kalder, der finder dem alle i brug, venter `pool_timeout`, som standard 30 sekunder, før den giver op. Indstillingerne er nøgleordsargumenter til `create_engine()`:

```
from sqlalchemy import create_engine

# Højst ti forbindelser til denne engine: ti holdes åbne, ingen ud over dem.
engine = create_engine(
    "postgresql+psycopg2://me@localhost/mydb",
    pool_size=10,
    max_overflow=0,
    pool_timeout=10,  # vent ti sekunder på en ledig forbindelse, giv så op
    pool_pre_ping=True,  # test hver forbindelse, når den lånes ud
    pool_recycle=3600,  # udskift forbindelser, der er over en time gamle
)
```

Med `pool_pre_ping` tjekker puljen ved udlån, at forbindelsen stadig kan bruges, via driverens eget ping eller en sætning svarende til `SELECT 1`. `pool_recycle` er beregnet til servere, der lukker forbindelser, som har været ubrugt et vist stykke tid: med 3600 bliver enhver forbindelse, der har været åben i mere end en time, udskiftet ved næste udlån.

Django 6.1 har sin egen pulje til PostgreSQL. Den virker kun med driveren psycopg og kræver pakken `psycopg[pool]`, og den slås til med `"OPTIONS": {"pool": True}` i databaseindstillingerne eller med en ordbog af indstillinger til psycopgs `ConnectionPool`. Puljen genbruger forbindelser på tværs af tråde, og hver proces har en separat pulje for hvert databasealias. `CONN_MAX_AGE` skal blive på 0, når pooling er slået til; levetiden for en forbindelse i puljen sættes i stedet med puljens indstilling `max_lifetime`. Under ASGI råder Djangos dokumentation til at slå vedvarende forbindelser fra og bruge denne indbyggede pulje i stedet.

PgBouncers `pool_mode` afgør, hvornår en serverforbindelse kan gå videre til en anden klient. I sessionstilstand, som er standard, beholder klienten den, til den afbryder forbindelsen. I transaktionstilstand har en klient kun en serverforbindelse, mens en transaktion kører, så et par dusin serverforbindelser kan betjene langt flere klienter. Sætningstilstand frigiver forbindelsen efter hver

forespørgsel og tillader ikke transaktioner, der strækker sig over flere sætninger. Hver pulje rummer op til `default_pool_size` serverforbindelser pr. par af bruger og database, som standard 20, og PgBouncer tager imod op til `max_client_conn` klientforbindelser, som standard 100.

Virksomhedens egne noter sætter puljen tidligt i rækkefølgen. Optimér databasen, før du sharder den får læseren til at sammenligne de aktive sessioner i `pg_stat_activity` med antallet af CPU-kerner: når mange gange flere er aktive, end der er kerner, bruger serveren sin tid på at skifte mellem dem, og en forbindelsespulje løser det, før nogen hardware gør. At sælge den sidste billet præcis én gang bruger samme tanke på et flash-salg: hundrede tusind købere bør blive til et par dusin forbindelser gennem PgBouncer, ikke tusindvis af serverprocesser, der venter på én lås.

Hvor det går galt

Puljerne kræver tilsammen flere forbindelser, end serveren tillader. Hver pulje tæller kun sine egne forbindelser. Med SQLAlchemys standardværdier kan én engine nå op på 15, så tolv processer kan bede om 180 fra en server, hvis `max_connections` typisk er 100 som standard, med tre af dem reserveret til superbrugere, og Django har en separat pulje for hvert databasealias i hver proces. Læg det højeste antal for hver pulje, der peger på databasen, sammen på tværs af alle processer i alle applikationer og alle planlagte job, og hold summen under `max_connections` minus de reserverede pladser. Når det ikke kan lade sig gøre, så sæt PgBouncer foran og lad den styre antallet af serverforbindelser.

En forbindelse dør, mens den venter i puljen. En forbindelse, der har ligget ubrugt, kan blive lukket i den anden ende, uden at puljen ved det, og den næste låner fejler så på sin første sætning. Slå `pool_pre_ping` til i SQLAlchemy eller `CONN_HEALTH_CHECKS` i Django, som tester en genbrugt forbindelse én gang pr. forespørgsel, og sæt `pool_recycle` eller psycopgs `max_lifetime` lavere end den tid, hvorefter serveren lukker ubrugte forbindelser. Ingen af dem dækker en forbindelse, der falder midt i en transaktion: SQLAlchemys dokumentation siger klart, at pre-ping ikke håndterer det tilfælde, så applikationen må selv prøve arbejdet igen.

En worker oprettet med fork arver forælderens forbindelser. En webserver eller en jobkører, der opretter sin engine og derefter starter sine workers med fork, efterlader hvert barn med de samme puljeforbindelser som forælderen. SQLAlchemys dokumentation kalder det kritisk, at puljeforbindelser ikke deles

med en proces oprettet med fork, fordi begge processer så ville tale over den samme netværksforbindelse. Følg den anbefalede fremgangsmåde og kald `engine.dispose(close=False)` i hvert barns initializer, så barnet starter med en tom pulje og lader forælderens forbindelser være. Alternativerne er `NullPool`, som åbner og lukker en forbindelse ved hver brug, eller at kalde `engine.dispose()` før fork-kaldet.

Transaktionspooling ødelægger sessionstilstand. I PgBouncers transaktionstilstand kan to transaktioner fra samme klient køre på forskellige serverforbindelser, så intet af det, en klient sætter op for sin session, er sikkert at finde i dens næste transaktion. PgBouncers funktionsliste markerer `SET`, `LISTEN`, cursorer erklæret med `WITH HOLD`, `PREPARE` på SQL-niveau og advisory locks på sessionsniveau som aldrig understøttet i den tilstand og lægger ikke skjul på, at tilstanden i sagens natur bryder nogle af de antagelser, klienter gør sig om serveren. Hold sessionstilstand ude af kode, der går gennem en transaktionspooler, og send de klienter, der har brug for den, til en pulje i sessionstilstand. I Django skal `DISABLE_SERVER_SIDE_CURSORS` sættes til `True` for en database bag en transaktionspooler, ellers skal forespørgslerne køre inde i `atomic()`.

Læs mere

- Connection Pooling — `QueuePool` og dens standardværdier, håndtering af afbrudte forbindelser, nulstilling ved returnering og puljer ved fork i SQLAlchemy 2.1.
- PgBouncer features — sessions-, transaktions- og sætningstilstand, og hvilke PostgreSQL-funktioner hver af dem understøtter.
- PgBouncer configuration — `pool_mode`, `default_pool_size`, `max_client_conn` og timeouts med deres standardværdier.
- How Connections Are Established — PostgreSQLs model med én proces pr. bruger og postmaster-processen.
- Connections and Authentication — `max_connections` og de forbindelsespladser, der er reserveret til superbrugere og andre roller.
- Databases — vedvarende forbindelser, PostgreSQL-puljen og transaktionspooling i Djangos reference.

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 en forbindelsespulje?« 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/connection-pool/