Engineer

Verbindungspool

offene Datenbankverbindungen wiederverwenden und begrenzen

Was es ist

Ein Verbindungspool ist ein Satz offener Datenbankverbindungen, die eine Anwendung aufbewahrt und erneut ausleiht, statt für jede Arbeit eine neue Verbindung zu öffnen und danach zu schließen. Code entnimmt dem Pool eine Verbindung, führt seine Anweisungen darüber aus und gibt sie zurück. Der nächste Aufrufer erhält dieselbe bereits offene Verbindung. SQLAlchemys Dokumentation beschreibt Pooling als „ein Standardverfahren, um langlebige Verbindungen für eine effiziente Wiederverwendung im Speicher zu halten“. Dieselbe Seite gibt dem Pool eine zweite Aufgabe: Er verwaltet die Zahl gleichzeitig von der Anwendung nutzbarer Verbindungen. Eine Verkehrsspitze wartet deshalb auf freie Verbindungen, statt weitere zu öffnen.

Ein Pool kann an zwei Orten liegen. Innerhalb der Anwendung gehört er zur Datenbankbibliothek: Jede SQLAlchemy-`Engine` bringt einen mit, und Django kann für PostgreSQL einen in jedem Prozess halten. Außerhalb ist er ein separates Programm zwischen Clients und Datenbankserver. PgBouncer ist ein solcher Pooler für PostgreSQL. Anwendungen verbinden sich damit wie mit der Datenbank, und er leiht ihnen Serververbindungen aus eigenen Pools. Beides lässt sich kombinieren. Djangos Datenbankreferenz hat einen Abschnitt zum Betrieb hinter einem Transaktionspooler wie PgBouncer.

Warum sich das Offenhalten lohnt, zeigt die Serverseite. PostgreSQL gibt jedem Client einen eigenen Prozess. Ein überwachender Prozess namens Postmaster lauscht auf eingehende Verbindungen und startet für jede Verbindungsanfrage einen neuen Serverprozess. Jeder Client ist mit genau einem dieser Prozesse verbunden. Eine Verbindung zu öffnen heißt deshalb, einen Prozess auf dem Datenbankserver zu starten. Ein Pool bezahlt diesen Preis einmal und verwendet das Ergebnis danach wieder.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Verbindungspool?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Die Abfragen und fehlenden Indizes hinter einem von Jahr zu Jahr schwerfälligeren System finden und die entscheidenden Probleme beheben.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 30 Tage Stromdaten in 180 ms — 180 ms für die Aggregation eines Monats an Zählerständen

30 Tage Stromdaten in 180 ms Kontakt aufnehmen →

Warum es wichtig ist

Der erste Grund sind die Verbindungskosten. Djangos Dokumentation begründet persistente Verbindungen klar: Sie vermeiden den Aufwand einer neuen Verbindung bei jeder HTTP-Anfrage. Der Standard tut allerdings das Gegenteil. `CONN_MAX_AGE` ist 0 und schließt die Verbindung am Ende jeder Anfrage. Eine positive Sekundenzahl hält sie zur Wiederverwendung offen, `None` ohne Zeitlimit. Das ist die einfachste Wiederverwendung: eine Verbindung pro Thread, die die Anfrage überdauert. Ein Pool ist ihre allgemeine Form.

Der zweite Grund ist das Serverlimit. PostgreSQLs `max_connections` hat typischerweise den Standardwert 100. Bei unzureichenden Kerneleinstellungen kann er niedriger sein. Ändern lässt er sich nur beim Serverstart. Drei Plätze sind standardmäßig für Superuser reserviert. Auch eine Erhöhung kostet Ressourcen: PostgreSQL bemisst bestimmte Ressourcen, darunter gemeinsamen Speicher, direkt anhand von `max_connections`. Ohne Pool hält jeder Worker eine eigene Verbindung. Djangos Dokumentation nennt die Folge: Weil jeder Thread eine eigene Verbindung behält, muss die Datenbank mindestens so viele gleichzeitige Verbindungen wie Workerthreads annehmen. Ein Pool macht daraus eine vom Team gewählte und summierbare Zahl.

Der dritte Grund ist Hygiene. Zwischen zwei Ausleihen durchläuft jede Verbindung den Pool. Dort wird sie bereinigt und geprüft. Wird eine Verbindung zurückgegeben, ruft SQLAlchemys Pool `rollback()` auf ihr auf. Das setzt den gesamten Transaktionszustand zurück und gibt auch Sperren frei, damit der nächste Nutzer keine halbfertige Transaktion erbt. In PgBouncers standardmäßigem Session-Pooling-Modus wird seine `server_reset_query`, standardmäßig `DISCARD ALL`, ausgeführt, wenn eine Serververbindung freigegeben wird und bevor ein anderer Client sie erhält. Ein Pool kann eine Verbindung außerdem vor dem Ausleihen testen. Das ist bei längere Zeit ungenutzten Verbindungen wichtig.

Wie es in der Praxis aussieht

In SQLAlchemy 2.1 erhält jede `Engine` standardmäßig einen `QueuePool`, ausgenommen SQLite-Datenbanken im Arbeitsspeicher. Sein `pool_size` ist 5, `max_overflow` ist 10. Eine Engine kann deshalb gleichzeitig bis zu 15 Verbindungen halten. Sind alle belegt, wartet ein Aufrufer die durch `pool_timeout` festgelegte Dauer, standardmäßig 30 Sekunden, bevor er aufgibt. Die Einstellungen sind Schlüsselwortargumente von `create_engine()`:

```
from sqlalchemy import create_engine

# At most ten connections for this engine: ten kept open, none beyond.
engine = create_engine(
    "postgresql+psycopg2://me@localhost/mydb",
    pool_size=10,
    max_overflow=0,
    pool_timeout=10,  # wait ten seconds for a free connection, then give up
    pool_pre_ping=True,  # test each connection as it is checked out
    pool_recycle=3600,  # replace connections that are over an hour old
)
```

Mit `pool_pre_ping` prüft der Pool beim Entnehmen, ob die Verbindung noch nutzbar ist: über den Ping des Treibers oder eine Anweisung, die `SELECT 1` entspricht. `pool_recycle` ist für Server gedacht, die eine bestimmte Zeit ungenutzte Verbindungen schließen. Bei 3.600 wird jede seit mehr als einer Stunde offene Verbindung bei ihrer nächsten Entnahme ersetzt.

Django 6.1 hat einen eigenen PostgreSQL-Pool. Er funktioniert nur mit dem psycopg-Treiber, braucht das Paket `psycopg[pool]` und wird mit `"OPTIONS": {"pool": True}` in den Datenbankeinstellungen eingeschaltet. Alternativ steht dort ein Dictionary mit Optionen für psycopgs `ConnectionPool`. Der Pool verwendet Verbindungen über Threads hinweg wieder. Jeder Prozess hält einen separaten Pool pro Datenbankalias. `CONN_MAX_AGE` muss bei aktivem Pooling 0 bleiben. Die Lebensdauer setzt stattdessen die Pooloption `max_lifetime`. Unter ASGI empfiehlt Djangos Dokumentation, persistente Verbindungen auszuschalten und diesen eingebauten Pool zu verwenden.

PgBouncers `pool_mode` entscheidet, wann eine Serververbindung an einen anderen Client gehen kann. Im standardmäßigen Sitzungsmodus behält der Client sie bis zur Trennung. Im Transaktionsmodus besitzt er nur während einer Transaktion eine Serververbindung. Wenige Dutzend Serververbindungen

können dadurch viel mehr Clients bedienen. Der Anweisungsmodus gibt die Verbindung nach jeder Abfrage frei und erlaubt keine über mehrere Anweisungen reichenden Transaktionen. Jeder Pool hält bis zu `default_pool_size` Serververbindungen pro Paar aus Nutzer und Datenbank, standardmäßig 20. PgBouncer nimmt bis zu `max_client_conn` Clientverbindungen an, standardmäßig 100.

Die eigenen Notizen des Unternehmens ordnen den Pool früh in der Arbeit ein. Die Datenbank vor dem Sharding optimieren lässt Leser die Zahl der aktiven Sitzungen in `pg_stat_activity` mit der Zahl der CPU-Kerne vergleichen. Sind um ein Vielfaches mehr aktiv als Kerne vorhanden, verbringt der Server seine Zeit mit dem Wechseln dazwischen. Ein Verbindungspool behebt das Problem schon vor einer Hardwareaufrüstung. Das letzte Ticket genau einmal verkaufen wendet dieselbe Idee auf einen Blitzverkauf an: Hunderttausend Käufer sollten über PgBouncer zu wenigen Dutzend Verbindungen werden, nicht zu Tausenden Serverprozessen, die auf eine Sperre warten.

Wo es schiefgeht

Die Pools summieren sich auf mehr, als der Server erlaubt. Jeder Pool zählt nur eigene Verbindungen. Mit SQLAlchemys Standardwerten erreicht eine Engine 15. Zwölf Prozesse können deshalb 180 von einem Server verlangen, dessen Standardwert für `max_connections` typischerweise 100 beträgt, wobei drei Plätze für Superuser reserviert sind. Django hält außerdem für jeden Datenbankalias in jedem Prozess einen eigenen Pool. Summieren Sie die Höchstzahlen aller Pools für diese Datenbank, über jeden Prozess jeder Anwendung und jeden geplanten Job hinweg. Halten Sie die Summe unter `max_connections` abzüglich reservierter Plätze. Passt das nicht, stellen Sie PgBouncer davor und lassen Sie ihn die Zahl der Serververbindungen verwalten.

Eine Verbindung stirbt beim Warten im Pool. Die Gegenseite kann eine ungenutzte Verbindung schließen, ohne dass der Pool es weiß. Der nächste Nutzer scheitert dann mit seiner ersten Anweisung. Aktivieren Sie `pool_pre_ping` in SQLAlchemy oder `CONN_HEALTH_CHECKS` in Django. Letzteres prüft eine wiederverwendete Verbindung einmal pro Anfrage. Setzen Sie `pool_recycle` oder psycopgs `max_lifetime` unter die Zeit, nach der der Server ungenutzte Verbindungen schließt. Keines hilft bei einem Abbruch mitten in einer Transaktion. SQLAlchemys Dokumentation sagt ausdrücklich, dass Pre-Ping diesen Fall nicht behandelt. Die Anwendung muss die Arbeit selbst wiederholen.

Ein geforkter Worker erbt die Verbindungen des Elternprozesses. Ein Webserver oder Job Runner, der erst die Engine erzeugt und danach Worker forkt, lässt jeden Kindprozess dieselben Poolverbindungen wie den Elternprozess halten. SQLAlchemys Dokumentation nennt es entscheidend, Poolverbindungen nicht mit geforkten Prozessen zu teilen. Sonst sprechen beide Prozesse über dieselbe Netzwerkverbindung. Folgen Sie dem empfohlenen Ansatz und rufen Sie `engine.dispose(close=False)` in der Initialisierung jedes Kindprozesses auf. Er beginnt dann mit leerem Pool und lässt die Elternverbindungen unberührt. Alternativen sind `NullPool`, der bei jeder Nutzung eine Verbindung öffnet und schließt, oder `engine.dispose()` vor dem Forken.

Bei Transaktionspooling ist Sitzungszustand nicht garantiert. Im Transaktionsmodus von PgBouncer können zwei Transaktionen desselben Clients über verschiedene Serververbindungen laufen. Was der Client für seine Sitzung einrichtet, ist deshalb bei der nächsten Transaktion nicht garantiert vorhanden. PgBouncers Funktionsliste kennzeichnet `SET`, `LISTEN`, mit `WITH HOLD` deklarierte Cursor, `PREPARE` auf SQL-Ebene und Advisory Locks auf Sitzungsebene als in diesem Modus nie unterstützt. Sie sagt offen, dass der Modus von Natur aus einige Clientannahmen über den Server verletzt. Halten Sie Sitzungszustand aus Code heraus, der einen Transaktionspooler durchläuft. Leiten Sie Clients, die ihn brauchen, an einen Pool im Sitzungsmodus. Setzen Sie in Django `DISABLE_SERVER_SIDE_CURSORS` für eine Datenbank hinter einem Transaktionspooler auf `True` oder führen Sie diese Abfragen in `atomic()` aus.

Weiterführende Lektüre

- Connection Pooling — `QueuePool` und seine Standardwerte, Behandlung getrennter Verbindungen, Zurücksetzen bei Rückgabe und Pools nach Forks in SQLAlchemy 2.1.
- PgBouncer-Funktionen — Sitzungs-, Transaktions- und Anweisungsmodus sowie die jeweils unterstützten PostgreSQL-Funktionen.
- PgBouncer-Konfiguration — `pool_mode`, `default_pool_size`, `max_client_conn` und Zeitlimits mit Standardwerten.
- Wie Verbindungen entstehen — PostgreSQLs Modell eines Prozesses pro Nutzer und der Postmaster.
- Verbindungen und Authentifizierung — `max_connections` und die für Superuser und andere Rollen reservierten Verbindungsplätze.
- Datenbanken — persistente Verbindungen, PostgreSQL-Pool und Transaktionspooling in Djangos Referenz.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Bringen Sie einen repräsentativen Datensatz, die Fragen, die er beantworten muss, und den aktuellen Engpass mit. Wir können vereinbaren, wie Qualität, Durchsatz und Wiederherstellung gemessen werden.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ein Verbindungspool?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/connection-pool/