

بسم الله الرحمن الرحيم



پروژه‌ی پایگاه داده

Azure پروژه‌ی

محمد زارع ۴۰۱۱۰۶۰۰۶

علی قاسم زاده ۴۰۱۱۰۶۳۳۹

سید محمد میرزادی ۴۰۱۱۱۰۱۰۴

امیررضا اینانلو ۴۰۱۱۰۵۶۶۷

۱ مقدمه

در سال‌های اخیر، رایانش ابری به عنوان یکی از مهم‌ترین زیرساخت‌های فناوری اطلاعات، نقشی کلیدی در توسعه سامانه‌های نوین نرم‌افزاری ایفا کرده است. در این میان، معماری‌های بدون سرور (Serverless) به دلیل کاهش پیچیدگی مدیریت زیرساخت، مقیاس‌پذیری خودکار و پرداخت مبتنی بر مصرف، به شدت مورد توجه قرار گرفته‌اند. یکی از سرویس‌های پرکاربرد در این حوزه، Azure Functions از شرکت مایکروسافت است که به توسعه‌دهنده‌گان اجراه می‌دهد منطق کسب‌وکار را در قالب توابع کوچک و مقیاس‌پذیر پیاده‌سازی کنند، بدون نیاز به مدیریت مستقیم سرورها.

برای ارزیابی دقیق چنین سامانه‌هایی، استفاده از ردیابی‌های واقعی بار کاری اهمیت ویژه‌ای دارد. دیتابست Azure Functions Trace 2019 (گردآوری شده در جولای ۲۰۱۹) زیرمجموعه‌ای از داده‌هایی است که در مقاله‌ی USENIX ATC 2020 با عنوان Serverless in the Wild تحلیل شده‌اند. نمونه‌برداری این دیتابست در سطح برنامه انجام شده و هر جا برنامه‌ای وارد نمونه شده، تمام توابعش نیز حضور دارد. در Functions، «برنامه» واحد تخصیص منبع است؛ بنابراین تصمیم‌های warm-up و اندازه‌گیری حافظه در سطح برنامه انجام می‌شود. شناسه‌ها (HashApp، HashOwner) با HMAC-HashFunction با SHA256 و نمک‌های متفاوت ناشناسی شده‌اند تا هم‌بستگی امن بین فایل‌ها حفظ شود. بر این اساس، تحلیل این ردیاب می‌تواند الگوهای مصرف، ناهنجاری‌ها و فرصت‌های بهینه‌سازی را در محیطی نزدیک به تولید آشکار کند.

۲ معرفی پروژه

این پروژه بر اساس دیتابست Azure Functions 2019 طراحی شده است؛ مجموعه‌ای از فایل‌ها که بازه‌ی ۱۴ روزه‌ای از بار کاری را پوشش می‌دهند و سه خانواده داده‌ی اصلی را شامل می‌شوند:

- **شمار فراخوانی تابع و محرک‌ها** (Function invocation counts and triggers): برای هر روز یک فایل که ۱۴۴۰ فیلد دقیقه‌ای (دقیقه‌های ۱ تا ۱۴۴۰ شباه روز) دارد. محرک‌ها در ۷ گروه کلی (مانند storage، queue، timer، http، ...) دسته‌بندی شده‌اند و پس از اجرای تابع شمارش می‌شوند.
 - **توزیع مدت زمان اجرای تابع** (Function execution durations): برای هر روز آماری مانند p0/p1/p25/p50/p75/p99/p100 و صدک‌های وزن دار Maximum، Minimum، Count، Average (بر پایه‌ی ثبت‌های ۳۳ ثانیه‌ای از میانگین‌های بازه‌ای) ارائه می‌شود؛ لذا ممکن است در موارد نادر با جمع فراخوانی‌های فایل دقیقه‌ای کمی اختلاف داشته باشد.
 - **توزیع حافظه‌ی برنامه‌ها** (Application memory): نمونه‌برداری حافظه هر ۵s انجام و هر دقیقه میانگین می‌شود؛ فایل‌های روزانه صدک‌های AverageAllocatedMb را گزارش می‌کنند. (در این ردیاب ۱۲ فایل نخست موجود است و دو روز آخر حافظه غایب است).
- اهداف پروژه عبارت‌اند از: پیش‌پردازش و نرمال‌سازی داده‌ها، طراحی مدل ذخیره‌سازی کارا، و اجرای کوئری‌های تحلیلی برای استخراج شاخص‌هایی مانند تاخیر (Latency)، توان عملیاتی (Throughput) و الگوهای مصرف حافظه. برای کاراتر شدن ذخیره‌سازی و تحلیل:

• داده‌های ۱۴ روز در سطح جدول مرج شده و ستون day به جداول واقعیت افزوده می‌شود تا فیلتر/تجمعیع روزانه ساده شود.

• بهدلیل بزرگی HashFunction و فراوانی رکوردها، بهجای نگهداشتن مستقیم آن در فکت‌ها، یک کلید جانشین عددی function_id تعریف و نگاشت function_id ↔ hash_function در جدول مجزا ذخیره می‌گردد.

• شمار فراخوانی‌های دقیقه‌ای هر روز بهجای ۱۴۴۰ ستون، در یک آرایه‌ی SMALLINT[1440] نگهداری می‌شود تا هم فضای ذخیره‌سازی کم شود و هم بازیابی کل روز برای یک تابع با یک دسترسی انجام شود.

در گام‌های بعدی، این داده‌ها در پایگاهداده (رابطه‌ای یا سری‌زمانی) بارگذاری می‌شوند، مدل داده و ایندکس‌ها پیاده‌سازی می‌گردد و با اجرای پرس‌وجوهای تحلیلی، رفتار توابع و برنامه‌ها بررسی شده و پیشنهادهای بهینه‌سازی ارائه می‌شود. دیتابست تحت مجوز CC-BY منتشر شده و در صورت استفاده علمی، ارجاع به مقاله‌ی USENIX ATC 2020 الزامی است.

۳ تکنولوژی‌های لازم

۱.۳ پریپراسس

تکنولوژی‌های مهمی که در مرحله پیش‌پردازش داده استفاده کردیم به شرح زیر هستند:

• **DuckDB** : موتور ستونی in-process برای آنالیتیکس. مستقیم روی فایل‌های CSV کار می‌کند، دستورهای SQL قوی مانند JOIN و ARRAY_AGG UNPIVOT را پشتیبانی می‌کند و نیازی به سرور جداگانه ندارد. انتخابی عالی برای پردازش دسته‌ای و سریع داده‌های حجمی روی لپ‌تاپ یا سرور سبک است.

• **PostgreSQL** : مقصد نهایی داده‌ها. یک پایگاهداده رابطه‌ای قدرتمند با پشتیبانی از نرم‌مال‌سازی، ایندکس‌های پیشرفته و کوئری‌های تحلیلی. داده‌های تمیز و نرم‌مال‌شده در آن ذخیره می‌شوند تا برای تحلیل‌های چندبعدی (مانند رابطه App □ Owner □ Function) آماده باشند.

• **CSV** : فرمت میانی و انتقالی. ساده، سبک و سازگار با همه ابزارها. در این پروژه به عنوان مرحله واسطه برای فایل‌های روزانه و همچنین برای خروجی نهایی استفاده شد.

۲.۳ لود دیتا

برای بارگذاری داده‌های پیش‌پردازش شده در پایگاهداده از ابزارها و کتابخانه‌های زیر استفاده شده است:

• **psycopg2** : کتابخانه استاندارد پایتون برای اتصال به PostgreSQL این ابزار امکان اجرای دستورات SQL، مدیریت تراکنش‌ها (commit/rollback) و درج داده‌های دسته‌ای را فراهم می‌کند. استفاده از آن باعث می‌شود فرآیند لود داده قابل اعتماد و اتمیک باشد.

• **PostgreSQL** : داده‌ها بعد از پیش‌پردازش در جداول این دیتابیس ذخیره می‌شوند. طراحی نرم‌الشده و استفاده از کلیدهای خارجی باعث تضمین یکپارچگی داده و آماده‌شدن آن برای کوئری‌های تحلیلی می‌گردد.

• **CSV** : داده‌های آماده قبل از بارگذاری در قالب فایل CSV ذخیره شده‌اند. این کار باعث سادگی پردازش دسته‌ای و انتقال داده بین DuckDB و PostgreSQL می‌شود.

۳.۳ دریافت اطلاعات بر خط

برای دریافت داده آنلاین و درج مستقیم در پایگاهداده، از ترکیب فناوری‌های زیر استفاده شد:

• **Flask** : فریمورک مینیمال پایتون برای ساخت API REST مسیرهای /upload/invocations به صورت JSON (برای ارسال برنامه‌ای) و /upload/memory به صورت CSV (برای پذیرنده‌ی پردازش دسته‌ای) پشتیبانی می‌شوند. این انعطاف دو سناریو را پوشش می‌دهد: بارگذاری توسعه و استقرار سریع سرویس را ممکن می‌سازد.

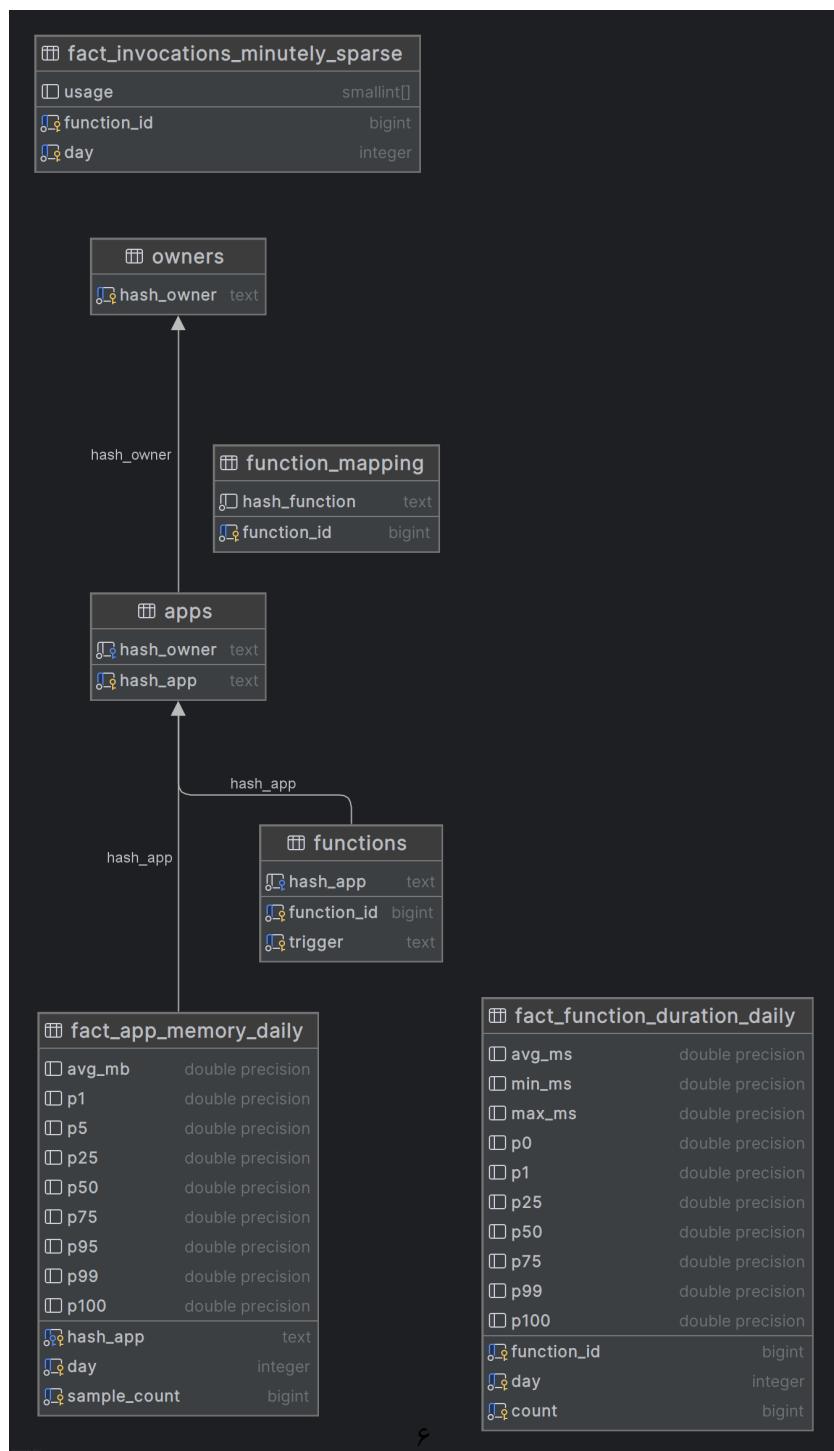
• **HTTP + JSON/CSV** : داده‌ها هم به صورت JSON (برای ارسال برنامه‌ای) و هم به صورت CSV (برای بارگذاری دسته‌ای) پشتیبانی می‌شوند. این انعطاف دو سناریو را پوشش می‌دهد: بارگذاری آنلاین و پردازش دسته‌ای.

• **PostgreSQL + psycopg2** : درج مستقیم داده در جداول اصلی. استفاده از تراکنش‌ها و دستورهای NOTHING DO CONFLICT ON باعث می‌شود داده‌های تکراری مدیریت شوند و درج ایدم پوتنت باشد. این همانگی با مدل نرم‌الشده (function_id) به جای hash_function سرعت و یکپارچگی داده را تضمین می‌کند.

• **Environment Variables** : پیکربندی اتصال به پایگاهداده (هاست، پورت، یوزر، پسورد) از محیط خوانده می‌شود. این باعث می‌شود استقرار در محیط‌های مختلف (Dev/Prod/Docker/Kubernetes) بدون تغییر کد امکان‌پذیر باشد.

Δ

۴ نرم‌السازی



شکل ۱ : a

در این پروژه ابتدا بهمنظور یکپارچه‌سازی بازه‌ی زمانی، یک ستون `day` به جداول واقعیت (فکت) اضافه کردیم و داده‌های ۱۴ روز را به صورت `merged` در همان جداول ذخیره نمودیم. مقدار `day` یک شناسه‌ی عددی روز (مثلاً `YYYYMMDD` یا یک ایندکس ترتیبی) است تا بتوانیم تحلیل‌های روزانه/میان‌روزی را با یک کلید ساده فیلتر کنیم. توجه شود که موجودیت‌های هویتی مانند `apps` و `owners` `ذاتاً` وابسته به روز نیستند، بنابراین برای این جداول ستون `day` تعریف نشده است. در مقابل، جداولی که آمار روزانه یا دقیقه‌ای را نگه می‌دارند (فکت‌ها)، `day` را به عنوان بخشی از کلید یا ایندکس دریافت کرده‌اند.

پس از یکپارچه‌سازی زمانی، نرم‌السازی داده‌ها را به‌ نحوی انجام دادیم که **کلیدهای سنگین و پر تکرار** از مسیر پرس‌وجوه‌ای پرترافیک کنار گذاشته شوند و حجم ذخیره‌سازی و هزینه‌ی ایندکس گذاری کاهش یابد. مهم‌ترین تصمیم طراحی این بود که به جای نگهداشت مستقیم `hash_function` (که خیلی بزرگ است و تعداد بسیار زیادی رکورد دارد) در جداول پر تکرار، یک **کلید جانشین عددی** به نام `function_id` به ستون `hash_function` بگذشت آن را در یک جدول مجزا نگه داریم. به این ترتیب، بیشترین حجم که ناشی از ستون `hash_function` بود از فکت‌ها خارج شد و اندازه‌ی جداول و ایندکس‌ها به‌طور محسوسی کوچک‌تر گردید. در ادامه، ساختار جداول و استدلال طراحی هر کدام به صورت خلاصه آمده است:

• **owners(hash_owner)** : فهرست همه‌ی مالک‌ها با شناسه‌ی ناشناس‌سازی شده. این موجودیت وابسته به روز نیست، لذا ستون `day` ندارد. نگهداری مالکان در یک جدول جدا، نرم‌السازی را حفظ و از تکرار `hash_owner` در سایر جداول جلوگیری می‌کند.

• **apps(hash_app, hash_owner)** : هر `app` دقیقاً متعلق به یک `owner` است؛ بنابراین با کلید اصلی `hash_app` و کلید خارجی به `owners` مدل شد. چون نسبت به روز ثابت است، ستون `day` ندارد. این چینش رابطه‌ی سلسه‌مراتبی `App → Owner` را شفاف نگه می‌دارد.

• **functions(function_id, hash_app, trigger)** : هر تابع به یک `app` مربوط است. برای پشتیبانی از تحلیل تفکیکی بر مبنای نوع `trigger`، کلید اصلی به صورت مرکب `(function_id, trigger)` تعریف شد تا در صورت تفاوت رفتار آمار بر حسب `trigger` بتوان آن‌ها را جداگانه مشاهده کرد.

• **function_mapping(function_id, hash_function)** : بدلیل بزرگ بودن `hash_function` و فراوانی رکوردها، حجم عمده‌ی ذخیره‌سازی مربوط به همین ستون بود. برای کاهش شدید حجم فکت‌ها و ایندکس‌ها و بهبود سرعت `function_id GROUP BY JOIN` را به عنوان کلید جانشین تعریف کردیم و `hash_function` را فقط در این جدول نگه داشتیم. نتیجه: دیتا است کوچک‌تر، ایندکس‌های سبک‌تر، و پردازش سریع‌تر.

• **fact_invocations_minutely_sparse(function_id, day, usage[1440])** : برای هر تابع و هر روز، شمار فراخوانی‌های هر دقیقه را نگه می‌داریم. به جای ۱۴۴۰ ستون مجزا، همه‌ی دقایق روز را در یک آرایه `[1440]` ذخیره کردیم؛ به‌ نحوی که درایه‌ی $i - 1$ تعداد کال‌های دقیقه‌ی i را نگه می‌دارد (ایندکس صفر-مبنی). این کار، همان «مرج ستون‌های دقیقه‌ای» در قالب آرایه است و مزیتش کاهش شدید تعداد ستون‌ها، کوچک شدن متادیتای کاتالوگ و خواندن کارآمد کل روز با یک دسترسی است.

fact_function_duration_daily(function_id, day, avg_ms, min_ms, . . .)

توزیع مدت زمان اجرای توابع را در سطح روز نگه می دارد. وجود count در کلید اصلی علاوه بر بکتا کردن ردیف، سازگاری بین آمار و اندازه هی نمونه را تضمین می کند.

fact_app_memory_daily(hash_app, day, sample_count, avg_mb, p1, p5, . . .)

آمار مصرف حافظه های برنامه ها در سطح روز. مشابه منطق قبلي، sample_count در کلید اصلی برای تضمین سازگاری آمار با اندازه هی نمونه و upsert-پسندی افزوده شده است.

۵ پیش پردازش دیتا ها

در این بخش به نحوه ای ساخت و کدی که برای استخراج و ساخت تیبل های بخش نرم افزار سازی زده بودیم می پردازیم :

: owners(...)

این بخش از کد مسئول استخراج شناسه های یکتای مالکان از میان فایل های داده است. ابتدا تابع sorted_files(prefix) موجود در پوشه را که با پیشوند مشخصی آغاز می شوند بر می گرداند و آن ها را بر اساس شماره موجود در انتهای نام فایل (با الگوی $d<number>.csv$) مرتب می کند تا ترتیب پردازش پایدار باشد. سپس یک اتصال درون حافظه ای به پایگاه داده DuckDB ایجاد می شود و جدولی موقتی با نام `owners_stg` برای جمع آوری مقادیر ستون HashOwner ساخته می گردد.

کد سه خانواده فایل را پردازش می کند: `invocations_per_function`, `app_memory_percentiles` و `function_durations_percentiles`. در هر خانواده تقریباً بین ۱۲ تا ۱۴ فایل وجود دارد و در مجموع چند ده فایل بررسی می شوند. برای هر فایل فقط ستون HashOwner خوانده می شود و رکوردهای آن در جدول موقت درج می گردند. نکته مهم این است که از دستور DISTINCT SELECT استفاده شده تا از ورود مقادیر تکراری جلوگیری شود؛ ابتدا در سطح هر فایل و سپس یک بار دیگر در پایان کار در سطح کل داده ها.

در گام پایانی، مجموعه یکتای مالکان در جدول `owners_final` ذخیره می شود و با استفاده از دستور COPY به صورت یک فایل خروجی CSV با نام `owners.csv` صادر می گردد. این طراحی باعث می شود تنها شناسه های منحصر به فرد مالکان ذخیره شوند و هیچ داده تکراری در خروجی باقی نماند.

: apps(...)

در این مرحله، هدف استخراج شناسه یکتای برنامه ها و صاحبان آن ها از فایل های مختلف دیتابست است. برای این منظور، ابتدا سه خانواده اصلی از فایل ها شامل `func-invocations_per_function`, `app-memory_percentiles` و `function_durations_percentiles` مورد پردازش قرار می گیرند.

تمام فایل‌های مرتبط با این خانواده‌ها به ترتیب شماره موجود در نام فایل مرتب شده و یکی پس از دیگری خوانده می‌شوند. از هر فایل، مقادیر HashOwner و HashApp استخراج شده و تنها رکوردهایی که دارای مقادیر معتبر هستند در جدول موقت apps_stg ذخیره می‌گردند.

پس از اسکن کامل همه فایل‌ها، رکوردهای تکراری حذف شده و جدول نهایی apps_final ساخته می‌شود که شامل ترکیب یکتا برنامه و صاحب آن است. در پایان، این جدول به صورت یک فایل CSV با نام apps.csv ذخیره شده و تعداد رکوردهای تولیدی گزارش می‌شود. این فایل به عنوان مرجع برنامه‌ها در مراحل بعدی پردازش مورد استفاده قرار می‌گیرد.

• **function_mapping(...)**

در ابتدای مرحله پیش‌پردازش، لازم است میان هر HashFunction و یک شناسه یکتا ارتباط برقرار شود. هدف از این کار، یکسان‌سازی ارجاع به توابعی است که در فایل‌های مختلف دیتابست ظاهر می‌شوند. برای این منظور، ابتدا تمام مقادیر متایز HashFunction از دو منبع داده شامل فایل‌های فراخوانی توابع (invocations_per_function) و فایل‌های زمان اجرای توابع (function_durations_percentiles) استخراج می‌شوند. سپس توابع تکراری حذف شده و فهرستی از توابع یکتا به دست می‌آید.

با استفاده از دستور row_number() در SQL به هر تابع یک شناسه عددی (function_id) اختصاص داده می‌شود. این شناسه‌ها به ترتیب مرتب‌سازی الفبایی مقدار هش تولید می‌شوند تا نظم ثابتی در تمامی مراحل پردازش برقرار باشد.

در نهایت، جدول حاصل به صورت یک فایل CSV با نام function_mapping.csv ذخیره می‌شود. این فایل نقش مرجع اصلی را در ادامه پردازش‌ها ایفا می‌کند و تضمین می‌نماید که تمام مراحل بعدی بتوانند با استفاده از یک شناسه یکتا به هر تابع اشاره کنند.

• **fact_invocations_minutely_sparse(...)**

در این بخش، داده‌های مربوط به تعداد فراخوانی هر تابع در بازه‌های یک دقیقه‌ای استخراج و به فرمتی ساخت‌یافته تبدیل می‌شوند. فایل‌های ورودی از خانواده invocations_per_function خوانده شده و به ترتیب شماره روز (dXX.csv) مرتب‌سازی می‌گردند.

برای هر روز، ابتدا داده‌ها به کمک دستور UNPIVOT از حالت ستونی (1440 ستون متناظر با دقیقه‌های شباهه روز) به شکل ردیف‌های مجزا تبدیل می‌شوند. سپس شناسه هر تابع از طریق جدول function_mapping به داده‌ها افزوده می‌شود.

به منظور تکمیل مقادیر گمشده، یک جدول مرجع شامل تمام دقیقه‌های روز ساخته شده و برای هر تابع با عمل JOIN CROSS ترکیب می‌گردد؛ در نتیجه، اگر در دقیقه‌ای تعداد فراخوانی ثبت نشده باشد، مقدار آن با صفر جایگزین می‌شود.

در ادامه، داده‌های هر تابع به صورت یک آرایه (ARRAY_AGG) از مقادیر فراخوانی دقیقه‌ای در کل روز تجمعی می‌شوند. خروجی این مرحله شامل شناسه تابع، شماره روز و آرایه مصرف دقیقه‌ای است که برای هر روز در فایل جداگانه ذخیره می‌گردد.

در پایان، تمام فایل‌های روزانه به کمک تابع concat_csv_parts در یک فایل نهایی با نام fact_invocations_minutely_sparse.csv ادغام شده و تعداد کل رکوردهای خروجی گزارش می‌شود.

این فایل نهایی مبنای اصلی تحلیل‌های بعدی بر اساس رفتار فراخوانی توابع در سطح دقیقه‌ای خواهد بود.

• **fact_function_duration_daily(...)** : همان‌طور که در بخش قبلی گفته‌یم برای سبک‌سازی فکت‌ها، function_id را به جای hash_function نگه می‌داریم. کد این بخش دقیقاً همین نگاشت را اعمال می‌کند و خروجی روزمحور می‌سازد: ابتدا فایل function_mapping.csv باز می‌شود تا نگاشت (hash_function → function_id) در یک جدول موقت در DuckDB در دسترس باشد؛ سپس تمام فایل‌های روزانه‌ی خروجی (dNN) از نام فایل استخراج و به صورت ستون day به خروجی تزریق خوانده می‌شوند، شماره‌ی روز (dNN) از نام JOIN INNER m.hash_function = src.HashFunction تا می‌گردد. روی JOIN INNER m.hash_function = src.HashFunction به جای هش سنگین، کمینه‌بیشینه، صدک‌ها و شمارش) برداشته می‌شود و نتیجه‌ی همان روز آماری موردنیاز (میانگین، کمینه‌بیشینه، صدک‌ها و شمارش) به صورت CSV موقت ذخیره می‌گردد. در انتهای، همه‌ی پارت‌های روزانه مرج شده و فایل نهایی به صورت fact_function_duration_daily.csv بنابراین هم ستون day برای تحلیل‌های روزانه داریم و هم به جای function_id از hash_function استفاده می‌کنیم.

• **fact_app_memory_daily(...)** : همان‌طور که در بخش نرم‌السازی گفته‌یم، برای سبک‌سازی فکت‌ها به جای hash_function از کلید عددی function_id استفاده می‌کنیم. این اسکریپت دقیقاً همین را پیاده می‌کند: ابتدا نگاشت (function_id → hash_function) از function_mapping.csv در DuckDB بارگذاری می‌شود؛ سپس همه فایل‌های روزانه (.csv). خوانده می‌شوند، شماره روز (dNN) از نام فایل استخراج و به صورت ستون day تزریق می‌شود. روی JOIN INNER HashFunction با جدول نگاشت می‌زنیم تا جایگزین شود و برای هر function_id در همان روز یک ردیف تجمعی می‌سازیم: avg_ms از min_ms/max_ms Average، p0, p1, p25, p50, p75, p99, p100 (به صورت میانگین Minimum/Maximum) صدک‌ها (p0, p1, p25, p50, p75, p99, p100) روزانه، و count به صورت جمع. خروجی هر روز به صورت CSV در tmp_dur در _tmp_dur نوشته می‌شود و در پایان، همه روزها مرج شده و فایل نهایی fact_function_duration_daily.csv تولید می‌گردد؛ نتیجه: فکت روزمحور با day و شناسه سبک function_id آماده تحلیل.

۶ لود کردن فایل‌های پیش‌پردازش شده

همان‌طور که در بخش بالا گفته شد ابتدا دیتا‌ها را پیش‌پردازش می‌کنیم به فرمت تیبل‌ها در فایل‌های csv سیو می‌کنیم و در ادامه با استفاده از کد زیر آنها را داخل تیبل‌های SQL می‌ریزیم:

Listing : ۱ Load preprocessed CSV to SQL

```
1 import csv  
2 import psycopg2  
3 from psycopg2 import sql  
4
```

```

5
6 DB_HOST = "1.0.0.127"
7 DB_PORT = 5434
8 DB_NAME = "azuredatabase"
9 DB_USER = "azureuser"
10 DB_PASSWORD = "azurepass"
11
12
13 conn = psycopg2.connect(
14     host=DB_HOST,
15     port=DB_PORT,
16     dbname=DB_NAME,
17     user=DB_USER,
18     password=DB_PASSWORD
19 )
20 print("connected!")
21 cur = conn.cursor()
22
23
24 csv_file = "./data/fact_invocations_minutely_sparse_pg.csv"
25 i = 0
26 with open(csv_file, newline='') as f:
27     reader = csv.reader(f)
28     headers = next(reader)
29
30     for row in reader:
31         function_id = int(row[0])
32         day = int(row[1])
33         usage_array = row[2].strip("{}").split(",")
34
35         for minute_index, count_str in enumerate(usage_array):
36             minute = minute_index + 1
37             count = int(count_str)
38
39             if count > 0:
40                 cur.execute(
41                     sql.SQL(
42                         "INSERT INTO raw_invocations(function_id,
43                         day, minute, count) VALUES (%s, %s, %s, %s)"
44                         ),

```

```

44             (function_id, day, minute, count)
45         )
46
47 conn.commit()
48 cur.close()
49 conn.close()
50 print("CSV data inserted into raw_invocations successfully.")

```

توضیح کوتاه: این اسکریپت با کتابخانه psycopg2 به پایگاهداده PostgreSQL متصل می‌شود، فایل CSV پیش‌پردازش شده را می‌خواند و ستون آرایهای [٤٤٠] usage را به ردیفهای دقیقه‌ای تبدیل می‌کند (از امین دقیقه در ایندکس 1-i). سپس فقط رکوردهای با شمارش غیرصفر را در جدول raw_invocations(function_id, day, minute, count) درج می‌کند تا حجم ذخیره‌سازی کاهش یابد. در پایان تراکنش commit و اتصال بسته می‌شود.

۷ کار با داده‌های ذخیره‌شده

- ۱۰ تابعی که بیشترین اوج فراخوانی در یک دقیقه را داشتند (شامل مشخص کردن دقیقه اوج).

```

1 SELECT fm.function_id,
2     f.day,
3     u.minute_of_day,
4     u.calls
5 FROM fact_invocations_minutely_sparse f
6 CROSS JOIN LATERAL (
7     -- unnest ordinality
8     SELECT cnt::int AS calls, ord AS minute_of_day
9     FROM unnest(f.usage) WITH ORDINALITY AS t(cnt, ord)
10    ORDER BY cnt DESC
11    LIMIT 1
12 ) AS u
13 JOIN function_mapping fm ON fm.function_id = f.function_id
14 ORDER BY u.calls DESC
15 LIMIT 10;
16

```

نمونه خروجی کوئری PostgreSQL

function_id	day	minute_of_day	calls
-------------	-----	---------------	-------

```

3   72069 | 14 |           1 | 32767
4   27032 | 13 |           1 | 32767
5   27032 | 14 |           1 | 32767
6   42388 | 14 |           1 | 32767
7   42388 | 12 |           1 | 32767
8   27032 | 12 |           1 | 32767
9   42388 | 13 |           1 | 32767
10  72069 | 13 |           1 | 32767
11  72069 | 11 |           1 | 32767
12  27032 | 1 |            1 | 32767
13 (10 rows)
14

```

این کوئری برای هر جفت (function_id, day) آرایه usage را با ORDER BY ORDINALITY به ردیف‌ها باز می‌کند تا شماره دقیقه حفظ شود و با

LIMIT 1 DESC دقیقه دارای بیشترین فراخوانی را پیدا می‌کند.

سپس خروجی با JOIN function_mapping می‌شود و همه این اوج‌ها بین تمام روزها/تابع بر اساس calls به صورت نزولی مرتب می‌شوند تا ۱۰ رکورد برتر همراه با minute_of_day و day باشند.

۲. شناسایی توابعی که مستقر شده‌اند ولی حداقل به مدت ۱۲ ساعت متواتی هیچ فراخوانی نداشتند (نشانه‌ای از هدررفت منابع).

```

1 WITH candidates AS (
2   SELECT
3     fim.function_id,
4     fim.day,
5     s.start_minute,
6     s.end_minute,
7     s.duration_minutes
8   FROM fact_invocations_minutely_sparse AS fim
9   CROSS JOIN LATERAL (
10    SELECT
11      MIN(i) AS start_minute,
12      MAX(i) AS end_minute,
13      COUNT(*) AS duration_minutes
14    FROM (
15      SELECT
16        i,
17        i - ROW_NUMBER() OVER (ORDER BY i) AS grp

```

```

18      FROM generate_subscripts(fim."usage", 1) AS i
19      WHERE fim."usage"[i] = 0
20      ) z
21      GROUP BY grp
22      HAVING COUNT(*) >= 720 -- at least 12 hours of
consecutive zeros
23      ORDER BY MIN(i)
24      LIMIT 1
25      ) AS s
26      WHERE 0 = ANY (fim."usage")
27 )
28 SELECT
29     fm.hash_function,
30     c.function_id,
31     fa.hash_app,
32     a.hash_owner,
33     c.day,
34     c.start_minute,
35     c.end_minute,
36     c.duration_minutes
37 FROM candidates c
38 LEFT JOIN function_mapping fm USING (function_id)
39 LEFT JOIN functions fa USING (function_id)
40 LEFT JOIN apps a USING (hash_app)
41 ORDER BY c.day, c.function_id;
42

```

	hash_function	function_id	day	start_minute	end_minute	duration_minutes	hash_app	
	hash_owner							
0007dfbf83f4e2675f84b8429b256d312f8d7f6a9264b4494f2ce3f8a79a	13 +48d8f6d7e04ed313c22161682f35900166ac9eedd9a59165b3fd993938	13 1	1440 1439					
1dp f0417c085bd1c51561867420b66623bc7885e27669a11bc7fa964dd3d3e4eb9	13 +48d8f6d7e04ed313c22161682f35900166ac9eedd9a59165b3fd993938	2 1440 1439						
0007dfbf83f4e2675f84b8429b256d312f8d7f6a9264b4494f2ce3f8a79a	13 +48d8f6d7e04ed313c22161682f35900166ac9eedd9a59165b3fd993938	13 1440 1439						
1dp f0417c085bd1c51561867420b66623bc7885e27669a11bc7fa964dd3d3e4eb9	13 +48d8f6d7e04ed313c22161682f35900166ac9eedd9a59165b3fd993938	2 1440 1439						
8dp 033aa19183aba2f215c7b1df32196e9fe9c0b7c78f10197efca25d08e9e0e9	1 25 +49d7269c7179d427979a 1440 1439							
000f2da683015e9d906123fadcab750a58335a0fb3ea0a9e41464524d01c1	25 +49d7269c7179d427979a 1440 1439							
8dp 033aa19183aba2f215c7b1df32196e9fe9c0b7c78f10197efca25d08e9e0e9	1 25 +49d7269c7179d427979a 1440 1439							
000f2da683015e9d906123fadcab750a58335a0fb3ea0a9e41464524d01c1	25 +49d7269c7179d427979a 1440 1439							
001742d12a73d7f78b75f224ca06323d74699a74143976156e28de3e8b1770	36 +74af26da2e0a960896dfe638e563f351fa83c100df623f1727f3deb00b2 1 1440 1439							
07c 2dadf437791419aa2dfeacfa66a3a22bhd558fc3b5c5e61c4a84f4ea3bc	36 +74af26da2e0a960896dfe638e563f351fa83c100df623f1727f3deb00b2 2 1440 1439							
07c 2dadf437791419aa2dfeacfa66a3a22bhd558fc3b5c5e61c4a84f4ea3bc	1 1440 1439							
001b49279bcc6e1d125a9705270bfef90bd1e95312b78c2db639f0a511bd3543	42 8cf907fc4ccffef3e3c91e1e2a8a5e7c3b4a3821ad0bc0795b6c2b81f8671c 1 1440 1439							
356 d363ba9b8308a52dc4f91fc8adb456a612aae9a263f31c6d4357f197814	42 8cf907fc4ccffef3e3c91e1e2a8a5e7c3b4a3821ad0bc0795b6c2b81f8671c 2 1440 1439							
002278142f117a513471795d25c6662311dc01bca7645677131643533f397814	54 92fe75fe48c7e67e7e843633a31f774d1c1bd57e99296e434489648f222b 1 1440 1439							
gec b989e67ae44cd3488b71aaeffce32a99b66dc711adba75dcf2a82fc6a7c	54 92fe75fe48c7e67e7e843633a31f774d1c1bd57e99296e434489648f222b 2 1440 1439							
0022707a024f1f74f37f19bf5672586aaa71291c4c01bee7a96ba0dbbbf98	54 92fe75fe48c7e67e7e843633a31f774d1c1bd57e99296e434489648f222b 1 1440 1439							
00240134db8cde2edfe516/e7e3a5a0cc6d5697339a896de72ad3369bb6b5b79a	55 b9166a9b20963289b67396c7195e95ca392832e14a3684ddc25b10548c 1 1440 1439							
045 0134db8cde2edfe516/e7e3a5a0cc6d5697339a896de72ad3369bb6b5b79a	55 b9166a9b20963289b67396c7195e95ca392832e14a3684ddc25b10548c 2 1440 1439							
d45 ral129678e995947ab5d7763ad9ef8799e99d6e104f7c88caf095e7dbb51f3	56 b9166a9b20963289b67396c7195e95ca392832e14a3684ddc25b10548c 1 1440 1439							
0027781b9643f1a7ea30f0d6827669a1e6b0bf97e44d07be91db0b3bea49e	61 e06889f7b16a37751c627f2f0d423d92a12e7834c25dd1b79156ae4 1 1440 1439							
* 011 5752ccb03d7fe673686d997d74236952a97063c373e1ba9b9206e09f755	61 2 1440 1439							
us 0027781b9643f1a7ea30f0d6827669a1e6b0bf97e44d07be91db0b3bea49e	61 e06889f7b16a37751c627f2f0d423d92a12e7834c25dd1b79156ae4 1 1440 1439							
011 0026666d5086611a8e311dc5b5c3978a85f3b08d6675156e08765979	70 804ba87943fe6915b86efca99ff45e877b0554260755a4bea6828d83ba 1 1440 1439							
016 8339aa7fa19868a592c44ef23de072bb693e1ca85ab8db22de230a999d433	70 804ba87943fe6915b86efca99ff45e877b0554260755a4bea6828d83ba 2 1440 1439							
022d6666d5086611a8e311dc5b5c3978a85f3b08d6675156e08765979	70 804ba87943fe6915b86efca99ff45e877b0554260755a4bea6828d83ba 1 1440 1439							
016 8339aa7fa19868a592c44ef23de072bb693e1ca85ab8db22de230a999d433	70 804ba87943fe6915b86efca99ff45e877b0554260755a4bea6828d83ba 2 1440 1439							
023113d23d4ef0530f9929161aac4208a59b5e79b35e77f8b7479	75 7a9fe1b403894679a54ff6186c767e626f9328d43140d061bd05a466							

شکل ۲: بخشی از خروجی کوئری ۲

CROSS JOIN LATERAL (function_id, day) با candidates آرایه usage[i]=0 را به اندیس‌ها باز می‌کند، فقط دقایقی که usage[i] ≠ 0 هستند را می‌گیرد و با ترفند row_number() - بازه‌های صفر بیوسته را گروه‌بندی می‌کند؛ سپس نخستین بازه‌ای را که طول اش end_minute/start_minute MAX(i) و MIN(i) بطور مترادف است با ≥ 720 ساعت) این کوئری بر می‌گرداند.

بعد، همین نامزدها با JOIN apps و functions .function_mapping می‌شوند تا function_id را مطابق شود و hash_app و hash_function را می‌ضمیمه کنند، اما day و hash_owner را با اساس start_minute/end_minute مرتباً مرتب گردد. شرط ANY(usage)=0 هم فقط ردیف‌هایی را نگه می‌دارد که دست کم یک صفر دارند.

۳. محاسبه ضریب تغییرات (انحراف معیار تقسیم بر میانگین) برای تعداد فراخوانی هرتابع در طول روز، برای یافتن توابع با الگوی استفاده ناپایدار.

```

1 WITH daily_invocations AS (
2   SELECT
3     function_id,
4     day,
5     CAST(SUM(unnested_value) AS BIGINT) as daily_total
6   FROM (
7     SELECT
8       function_id,
9       day,
10      unnest(usage) as unnested_value
11     FROM fact_invocations_minutely_sparse

```

```

12      ) t
13      GROUP BY function_id, day
14  ),
15  stats AS (
16      SELECT
17          function_id,
18          AVG(daily_total) as avg_invocations,
19          STDDEV(daily_total) as stddev_invocations,
20          COUNT(*) as days_count
21      FROM daily_invocations
22      GROUP BY function_id
23      HAVING COUNT(*) > 1 -- Need at least 2 days for
24      meaningful CV
25  )
26  SELECT
27      s.function_id,
28      fm.hash_function,
29      f.hash_app,
30      a.hash_owner,
31      s.avg_invocations,
32      s.stddev_invocations,
33      CASE
34          WHEN s.avg_invocations > 0 THEN s.stddev_invocations /
35          s.avg_invocations
36          ELSE 0
37      END as coefficient_of_variation,
38      s.days_count
39  FROM stats s
40  JOIN functions f ON f.function_id = s.function_id
41  JOIN apps a ON a.hash_app = f.hash_app
42  JOIN function_mapping fm ON fm.function_id = s.function_id
43  ORDER BY coefficient_of_variation DESC;

```

function_id	hash_function	hash_owner	avg_invocations	stddev_invocations	coefficient_of_variation	
on days_count						
51505 967add9bb4c8cb1690f258afa62cc30a7c79267fe63fcf254d75d774f9468ff9 f322aa1fc9c7203d63f6c47d428389cf84798e8a7abf9c3db00a2b4d4ac						
96 2934b89aab55befcd2cd21fc5f42bef4f12821b7330ce929f1a41c8c579eb1c9 43.21a2857142857143 156.794398269467 3.48945715003725						
29 51505 967add9bb4c8cb1690f258afa62cc30a7c79267fe63fcf254d75d774f9468ff9 f322aa1fc9c7203d63f6c47d428389cf84798e8a7abf9c3db00a2b4d4ac						
96 2934b89aab55befcd2cd21fc5f42bef4f12821b7330ce929f1a41c8c579eb1c9 43.21a2857142857143 156.794398269467 3.48945715003725						
c5 14						
26720 521b95083a0b26ca536eb7ef139967a44647da0f4fa69c9b528c681c2c5 24aa5cb4b4e5cd0a6c7fb3f7e1bba6298c761e65519efdf28b2717fa20a						
c5 14924858638a14f4fbdddc3c85453b7d8804fa0a032de9dcbb56dd5b6b70e6 52.2500000000000000 166.003901925664 3.17710817082610						
53 26720 521b95083a0b26ca536eb7ef139967a44647da0f4fa69c9b528c681c2c5 24aa5cb4b4e5cd0a6c7fb3f7e1bba6298c761e65519efdf28b2717fa20a						
c5 14924858638a14f4fbdddc3c85453b7d8804fa0a032de9dcbb56dd5b6b70e6 52.2500000000000000 166.003901925664 3.17710817082610						
53 12						
7175 15ecce5ea2ee3a2b295b2a5d758acb265f9d63967aa85f47a1d0aaeb750 3b1b1b15960026bfc1dd3edbcff43e29da117b23da073282591273532e						
44 7180414a5359a1fa192d7fd4e627fd0c20e4b738582e2407365f4a7610901 155.71a2857142857143 491.461310562917 3.15617355407377						
89 14						
7175 15ecce5ea2ee3a2b295b2a5d758acb265f9d63967aa85f47a1d0aaeb750 3b1b1b15960026bfc1dd3edbcff43e29da117b23da073282591273532e						
44 7180414a5359a1fa192d7fd4e627fd0c20e4b738582e2407365f4a7610901 155.71a2857142857143 491.461310562917 3.15617355407377						
53427 14						
c9 53427 146d77742f233c8527536a1c968dcfc1e23a532499b5f2d068862a277f bf415a17fee7b8fc798e6e4d43d5c0b7f8966ca3e5bbf06eb05fa8c7						
c9 196d34ee137a836cb00aaa3894038ef92b7271c6823ea3732355a70aaa77f 60.7857142857142857 191.781597966753 3.15504391484670						
04 14						
53427 146d77742f233c8527536a1c968dcfc1e23a532499b5f2d068862a277f bf415a17fee7b8fc798e6e4d43d5c0b7f8966ca3e5bbf06eb05fa8c7						
c9 196d34ee137a836cb00aaa3894038ef92b7271c6823ea3732355a70aaa77f 60.7857142857142857 191.781597966753 3.15504391484670						
04 14						
53427 146d77742f233c8527536a1c968dcfc1e23a532499b5f2d068862a277f bf415a17fee7b8fc798e6e4d43d5c0b7f8966ca3e5bbf06eb05fa8c7						
c9 196d34ee137a836cb00aaa3894038ef92b7271c6823ea3732355a70aaa77f 60.7857142857142857 191.781597966753 3.15504391484670						
04 14						
1201 03ab418d71c10ad8e8b26114bacff8a853f4104045bbd3aae9ec7ef8229 793b566c45d35997d39a08ebf487f047f736ca655058dfchddca23e8a52b						
29 3fed0dc8b84ef899583812a28d386ed5b1866761359b8c6fe0eb087eed06c 19.137692307697704047 3.03393948789043						
00 1201 03ab418d71c10ad8e8b26114bacff8a853f4104045bbd3aae9ec7ef8229 793b566c45d35997d39a08ebf487f047f736ca655058dfchddca23e8a52b						
29 3fed0dc8b84ef899583812a28d386ed5b1866761359b8c6fe0eb087eed06c 19.137692307697704047 3.03393948789043						
00 13						
68887 d3e812af9c3f6a6e856f363c5d1424f946e09aca6a678a495db9106130f82046 24450e3369facd5bfc6198a83dc0a9478df8127c11292dse8405f43a34						
00 ;						

شکل ۳: بخشی از خروجی کوئری ۳

در خروجی با usage (function_id, day) آرایه unnest برای هر daily_invocations می‌شود و مجموع همه دقيقه‌ها به عنوان daily_total به دست می‌آید. سپس در stats روى روزهای موجود، AVG و شمار روزها محاسبه می‌شود؛ شرط $\text{HAVING COUNT}(> 1)$ تضمین می‌کند حداقل دو روز داده برای محاسبه CV داریم.

در خروجی با JOIN function_mapping و apps.functions تغییرات daily_total محاسبه coefficient_of_variation (stddev_invocations / avg_invocations) و نزولی مرتب می‌گردد تا توابع با نوسان استفاده بیشتر در صدر بیایند.

۴. بررسی رابطه بین مدت زمان اجرای توابع با تعداد فراخوانی‌ها (برای کنترل تأثیر حجم فراخوانی).

```

1 WITH invocation_totals AS (
2     SELECT
3         function_id,
4         day,
5         CAST(SUM(unnested_value) AS BIGINT) AS
6             daily_invocations
7     FROM (
8         SELECT
9             function_id,
10            day,
11            unnest(usage) AS unnested_value
12        FROM fact_invocations_minutely_sparse
13    ) t

```

```

13     GROUP BY function_id, day
14 )
15 SELECT
16     fdd.function_id,
17     fm.hash_function,
18     f.hash_app,
19     fdd.day,
20     fdd.avg_ms,
21     fdd.count as duration_samples,
22     COALESCE(it.daily_invocations, 0) as invocations,
23     CORR(fdd.avg_ms, it.daily_invocations) OVER (PARTITION BY
24         fdd.function_id) as correlation
25 FROM fact_function_duration_daily fdd
26 LEFT JOIN invocation_totals it
27     ON it.function_id = fdd.function_id
28     AND it.day = fdd.day
29 JOIN functions f ON f.function_id = fdd.function_id
30 JOIN function_mapping fm ON fm.function_id = fdd.function_id
31 WHERE fdd.count > 0
32 ORDER BY fdd.function_id, fdd.day;
33

```

function_id	day	avg_ms	duration_samples	hash_function	invocations	correlation	hash_app
11 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	375 34 0 0 0 0 0 0					
11 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	196 27 0 0 0 0 0 0					
11 2 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	196 27 0 0 0 0 0 0					
11 2 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	165 131 0 0 0 0 0 0					
11 3 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	165 131 0 0 0 0 0 0					
11 3 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	165 131 0 0 0 0 0 0					
11 4 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	91 157 0 0 0 0 0 0					
11 4 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	91 157 0 0 0 0 0 0					
11 5 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	171 21 0 0 0 0 0 0					
11 5 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	171 21 0 0 0 0 0 0					
11 6 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	264 36 0 0 0 0 0 0					
11 8 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	264 36 0 0 0 0 0 0					
11 9 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	96 73 0 0 0 0 0 0					
11 9 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	96 73 0 0 0 0 0 0					
11 10 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	37 238 0 0 0 0 0 0					
11 10 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	1 1 00004840e45e3cdffdef18a6013bb1bb1aafe785bd19acf1258540e20288c45 0cd73d488dade56428fe8fc064414646d4bac2638234aa235ac8543c67c71	37 238 0 0 0 0 0 0					

شکل ۴: بخشی از خروجی کوئری ۴ ام

در (function_id, day) مجموع فراخوانی روزانه برای هر invocation_totals با

محاسبه و بهصورت daily_invocations unnest(usage) بر می گردد.
 سپس داده های مدت زمان روزانه از fact_function_duration_daily با همین مجموعه LEFT JOIN می شود تا حتی روزهای بدون شمارش فراخوانی هم بمانند؛ COALESCE صفر می گذارد و شرط $fdd.count > 0$ فقط روزهای دارای نمونه مدت زمان را نگه می دارد.
 در نهایت CORR(fdd.avg_ms, it.daily_invocations) OVER (PARTITION BY fdd.function_id) همبستگی پرسون بین میانگین مدت زمان و حجم فراخوانی را در سطح هرتابع محاسبه می کند تا اثر بار را ارزیابی کنیم؛ خروجی به ازای هر روز/تابع همراه با این مقدار گزارش می شود.

۵. تجمعی مجموع فراخوانی ها، میانگین مدت اجرا و میانگین حافظه تخصیص داده شده بر اساس مالک (Owner) برای مشخص کردن پرمصرف ترین مالکان.

```

1 WITH usage_agg AS (
2     SELECT
3         f.function_id,
4         a.hash_app,
5         o.hash_owner,
6         SUM(COALESCE(unnest_val, 0)) AS total_usage
7     FROM owners o
8     JOIN apps a ON a.hash_owner = o.hash_owner
9     JOIN functions f ON f.hash_app = a.hash_app
10    JOIN fact_invocations_minutely_sparse fis ON fis.
11        function_id = f.function_id
12        CROSS JOIN LATERAL unnest(fis.usage) AS u(unnest_val)
13        GROUP BY f.function_id, a.hash_app, o.hash_owner
14    ),
15    joined AS (
16        SELECT
17            ua.function_id,
18            ua.hash_app,
19            ua.hash_owner,
20            ua.total_usage,
21            fd.avg_ms,
22            fam.avg_mb
23        FROM usage_agg ua
24        LEFT JOIN fact_function_duration_daily fd ON fd.
25            function_id = ua.function_id
26        LEFT JOIN fact_app_memory_daily fam ON fam.hash_app = ua.
27            hash_app
28    )

```

```
 26 SELECT
 27     j.hash_owner,
 28     SUM(j.total_usage)      AS sum_total_usage,
 29     SUM(j.avg_mb)          AS sum_avg_mb,
 30     SUM(j.avg_ms)          AS sum_avg_ms
 31 FROM joined j
 32 GROUP BY j.hash_owner
 33 ORDER BY sum_total_usage DESC;
```

شکل ۵: بخشی از خروجی کوئری ۵ ام

در زنجیره usage_agg می‌شود و با unnest(usage) کل فراخوانی هر تابع اپ/مالک به صورت total_usage جمع می‌شود.

سپس در joined تجمعی با میانگین مدت اجرا از fact_function_duration_daily و میانگین حافظه از fact_app_memory_daily بهتر ترتیب بر اساس function_id و hash_app می شود تا مقادیر در صورت نبود نیز صفر خالی نشوند.

در نهایت بر مبنای hash_owner گروه بندی شده و $\text{SUM}(\text{avg_mb})$, $\text{SUM}(\text{total_usage})$, $\text{SUM}(\text{avg_ms})$ و $\text{SUM}(\text{sum_total_usage})$ محاسبه و به ترتیب نزولی مرتب می شود تا پر مصرف ترین مالکان مشخص شوند.

۶. شناسایی توابعی که در یک روز خاص تعداد فراخوانی‌شان بیش از ۲ برابر میانگین روزهای دیگر بوده (دای، تشخیص، ناهمجای).

```

1 WITH daily_invocations AS (
2     SELECT
3         function_id,
4         day,
5         CAST(SUM(unnested_value) AS BIGINT) as daily_total
6     FROM (
7         SELECT
8             function_id,
9             day,
10            unnest(usage) as unnested_value
11        FROM fact_invocations_minutely_sparse
12    ) t
13    GROUP BY function_id, day
14),
15 avg_invocations AS (
16     SELECT
17         function_id,
18         AVG(daily_total) as avg_daily,
19         MAX(daily_total) as max_daily
20     FROM daily_invocations
21     GROUP BY function_id
22)
23
24     SELECT
25         di.function_id,
26         fm.hash_function,
27         f.hash_app,
28         a.hash_owner,
29         di.day,
30         di.daily_total as invocations_on_day,
31         ai.avg_daily as average_daily_invocations,
32         di.daily_total / NULLIF(ai.avg_daily, 0) as ratio
33     FROM daily_invocations di
34     JOIN avg_invocations ai ON ai.function_id = di.function_id
35     JOIN functions f ON f.function_id = di.function_id
36     JOIN apps a ON a.hash_app = f.hash_app
37     JOIN function_mapping fm ON fm.function_id = di.function_id
38     WHERE di.daily_total > 2 * ai.avg_daily
39     ORDER BY ratio DESC;

```

function_id	hash_function	hash_app			
		hash_owner	day	invocations_on_day	average_daily_invocations
51585	95e7add9bb4c8cb169f27584fb62cc3807c79267fe63fcf254d75d774f946bf8ff9	f322aaad1fc9c723d63f6c4c7d262839cf047988e87aebf9c3db002b24d4ac			
96	2934889ab55bfeffcd2c12fc54f2be4ff18281b33cc0e92f9f1a1c8c579e01c9	14		567	43.21428571285713
34	51585	9e7add9bb4c8cb169f27584fb62cc3807c79267fe63fcf254d75d774f946bf8ff9	f322aaad1fc9c723d63f6c4d7428389cf847988e87abf9c3db002b24d4ac		
96	2934889ab55bfeffcd2c12fc54f2be4ff18281b33cc0e92f9f1a1c8c579e01c9	14		567	43.21428571285713
34	7175	15ecceeaee3aa2b95ba25d58b0ch0265720d593758a85f07d16a8e8750	3b12b1859600226bf1c1dd3ed8e8ff63e29dca11/b2d30n032352931735236		
44	7180441a53951a192d7fe4627fc0be4b738b2d42407565fa0761d901	12		1863	51.748215738285713
85	7175	15ecceeaee3aa2b95ba25d58b0ch0265720d593758a85f07d16a8e8750	3b12b1859600226bf1c1dd3ed8e8ff63e29dca11/b2d30n032352931735236		
44	7180441a53951a192d7fe4627fc0be4b738b2d42407565fa0761d901	12		1863	51.748215738285713
85	53427	a46d72742f2323c8c5c753e6ac19881dcf31c23a532499b5f0d65862a277f	bf415a17fe7b1a8f7986e6ec4d3dc5b0ff8e69c34e5bfb06e659f0ac8c7		
44	f9fd3e1317836c0f00aa3894038ef92bc7271c6823a323553a70aa7f	3		727	66.7857142857142857
44	53427	a46d72742f2323c8c5c753e6ac19881dcf31c23a532499b5f0d65862a277f	bf415a17fe7b1a8f7986e6ec4d3dc5b0ff8e69c34e5bfb06e659f0ac8c7		
44	f9fd3e1317836c0f00aa3894038ef92bc7271c6823a323553a70aa7f	3		727	66.7857142857142857
44	1201	03ab418d71c1104d868b92114bbcfa85f3d404a05bd33a4e47e7f8229	793b566c445d35997d39a0e80bf487f773c65058dfbcbbda3e8a3896		
29	3fe0b28c8e4678058332d836ed58b167e6139856cf086088ed5e0c	12		6.307692369623977	11.09759750975097
61	1201	03ab418d71c1104d868b92114bbcfa85f3d404a05bd33a4e47e7f8229	793b566c445d35997d39a0e80bf487f773c65058dfbcbbda3e8a3896		
29	3fe0b28c8e4678058332d836ed58b167e6139856cf086088ed5e0c	12		6.307692369623977	11.09759750975097
61	26720	521b9503ab2b6ba353eb7e2f139967a44647d4f0a64-96b52b681c25	24aa6cb4be5cd0a6c73fb3f180b62790-876165519ef9d28b2717fa202a		
c5	4192458e308a14f7fbbedd3c54557dab4fa0b33d269cb565d6bd5bbf0e6	4		579	52.250000000000000
c5	26720	521b9503ab2b6ba353eb7e2f139967a44647d4f0a64-96b52b681c25	24aa6cb4be5cd0a6c73fb3f180b62790-876165519ef9d28b2717fa202a		
c5	4192458e308a14f7fbbedd3c54557dab4fa0b33d269cb565d6bd5bbf0e6	4		579	52.250000000000000
> 3	48176	940281b074f971c0fddc63a6469fa3a908c984206f2bf41ceef8a023dd831c	3b12b1859600226bf1c1dd3ed8e8ff43e29dca11/b2d30n032352931735236		

شکل ۶: بخشی از خروجی کوئری ۶ ام

در daily_invocations آرایه usage با unnest (function_id, day) برای هر daily_invocation در می‌شود و مجموع دقیقه‌ها به عنوان daily_total محاسبه می‌گردد.
در avg_invocations میانگین روزانه هرتابع (avg_daily) و همچنین بیشینه آن (max_daily) استخراج می‌شود.

سپس با الحاق به جداول مرجع، نسبت daily_total / NULLIF(avg_daily,0) به عنوان ratio محاسبه و فقط روزهایی نگه داشته می‌شوند که daily_total > 2 * avg_daily؛ خروجی بر اساس ratio نزولی مرتب می‌شود تا ناهنجاری‌ها در صدر بیایند.

۷. پیدا کردن اپلیکیشن ها با حافظه زیاد اما استفاده کم (با توجه به تعداد فراخوانی و مدت زمان اجرا) که می توان آن ها را کوچک سازی کرد.

```
1 CREATE OR REPLACE FUNCTION array_sum(arr INTEGER[])
2 RETURNS BIGINT AS $$$
3 DECLARE
4     total BIGINT := 0;
5     elem INTEGER;
6 BEGIN
7     IF arr IS NULL THEN
8         RETURN 0;
9     END IF;
10
11    FOREACH elem IN ARRAY arr
12    LOOP
13        total := total + elem;
```

```

14     END LOOP;
15
16     RETURN total;
17 END;
18 $$ LANGUAGE plpgsql IMMUTABLE STRICT;
19
20 CREATE OR REPLACE FUNCTION has_consecutive_zeros(arr SMALLINT
21     [], consecutive_count INT)
22 RETURNS BOOLEAN AS $$
23 DECLARE
24     current_streak INT := 0;
25     i INT;
26 BEGIN
27     FOR i IN 1..array_length(arr, 1) LOOP
28         IF arr[i] = 0 THEN
29             current_streak := current_streak + 1;
30             IF current_streak >= consecutive_count THEN
31                 RETURN TRUE;
32             END IF;
33             ELSE
34                 current_streak := 0;
35             END IF;
36         END LOOP;
37         RETURN FALSE;
38     END;
39 $$ LANGUAGE plpgsql IMMUTABLE;
40
41 --+
42 CREATE INDEX IF NOT EXISTS idx_invocations_function_day ON
43     fact_invocations_minutely_sparse(function_id, day);
44 CREATE INDEX IF NOT EXISTS idx_duration_function_day ON
45     fact_function_duration_daily(function_id, day);
46 CREATE INDEX IF NOT EXISTS idx_memory_app_day ON
47     fact_app_memory_daily(hash_app, day);
48 CREATE INDEX IF NOT EXISTS idx_functions_app ON functions(
49     hash_app);
50
51
52 SELECT
53     f.function_id,

```

```

49     f.hash_app,
50     f.trigger,
51     inv.day,
52     array_sum(inv.usage) as total_daily_invocations
53 FROM functions f
54 JOIN fact_invocations_minutely_sparse inv ON f.function_id =
55     inv.function_id
56 WHERE has_consecutive_zeros(inv.usage, 720)
57 ORDER BY inv.day, f.function_id;
58 WITH invocation_stats AS (
59     SELECT
60         function_id,
61         SUM(array_sum(usage)) AS total_invocations
62     FROM fact_invocations_minutely_sparse
63     GROUP BY function_id
64 ),
65 duration_stats AS (
66     SELECT
67         function_id,
68         CASE
69             WHEN SUM(count) > 0 THEN SUM(avg_ms * count) / SUM
70             (count)
71             ELSE 0
72         END AS weighted_avg_duration,
73         SUM(count) AS total_executions
74     FROM fact_function_duration_daily
75     GROUP BY function_id
76 ),
77 memory_stats AS (
78     SELECT
79         f.function_id,
80         CASE
81             WHEN SUM(m.sample_count) > 0 THEN SUM(m.avg_mb * m
82             .sample_count) / SUM(m.sample_count)
83             ELSE 0
84         END AS weighted_avg_memory,
85         SUM(m.sample_count) AS total_samples
86     FROM functions f
87     JOIN fact_app_memory_daily m ON f.hash_app = m.hash_app
88     GROUP BY f.function_id

```

```

86 ),
87 criticality_calculation AS (
88     SELECT
89         f.function_id,
90         f.hash_app,
91         f.trigger,
92         COALESCE(i.total_invocations, 0) AS total_invocations,
93         COALESCE(d.weighted_avg_duration, 0) AS
94             avg_duration_ms,
95             COALESCE(d.total_executions, 0) AS total_executions,
96             COALESCE(m.weighted_avg_memory, 0) AS avg_memory_mb,
97             COALESCE(m.total_samples, 0) AS total_memory_samples,
98
99             (COALESCE(i.total_invocations, 0) * 4.0 +
100             COALESCE(d.weighted_avg_duration, 0) * 0003.0 +
101             COALESCE(m.weighted_avg_memory, 0) * (003.0 AS
102                 criticality_score
103             FROM functions f
104             LEFT JOIN invocation_stats i ON f.function_id = i.
105                 function_id
106             LEFT JOIN duration_stats d ON f.function_id = d.
107                 function_id
108             LEFT JOIN memory_stats m ON f.function_id = m.function_id
109         )
110     SELECT
111         function_id,
112         hash_app,
113         trigger,
114         total_invocations,
115         avg_duration_ms,
116         total_executions,
117         avg_memory_mb,
118         total_memory_samples,
119         criticality_score,
120
121         CASE
122             WHEN criticality_score > 1000 THEN ' '
123             WHEN criticality_score > 100 THEN ' '
124             WHEN criticality_score > 10 THEN ' '
125             ELSE ' '

```

```

122     END AS criticality_level
123 FROM criticality_calculation
124 ORDER BY criticality_score DESC;
125
126 WITH daily_invocations AS (
127     SELECT
128         function_id,
129         day,
130         array_sum(usage) AS daily_count
131     FROM fact_invocations_minutely_sparse
132 ),
133 function_avg_invocations AS (
134     SELECT
135         function_id,
136         AVG(daily_count) AS avg_daily_count,
137         STDDEV(daily_count) AS std_daily_count
138     FROM daily_invocations
139     GROUP BY function_id
140 ),
141 anomaly_detection AS (
142     SELECT
143         d.function_id,
144         d.day,
145         d.daily_count,
146         f.avg_daily_count,
147         f.std_daily_count,
148
149         CASE
150             WHEN d.daily_count > 2 * f.avg_daily_count THEN
151                 TRUE
152                 ELSE FALSE
153             END AS is_anomaly,
154
155             (d.daily_count - f.avg_daily_count) / NULLIF(f.
156             std_daily_count, 0) AS z_score
157             FROM daily_invocations d
158             JOIN function_avg_invocations f ON d.function_id = f.
159             function_id
160             WHERE f.avg_daily_count > 0
161 )

```

```

159 | SELECT
160 |     a.function_id,
161 |     f.hash_app,
162 |     f.trigger,
163 |     a.day,
164 |     a.daily_count,
165 |     ROUND(a.avg_daily_count::numeric, 2) AS avg_daily_count,
166 |     ROUND(a.std_daily_count::numeric, 2) AS std_daily_count,
167 |     ROUND(a.z_score::numeric, 2) AS z_score,
168 |     a.is_anomaly,
169 |
170 |     CASE
171 |         WHEN a.z_score > 3 THEN ' '
172 |         WHEN a.z_score > 2 THEN ' '
173 |         ELSE ' '
174 |     END AS anomaly_severity
175 | FROM anomaly_detection a
176 | JOIN functions f ON a.function_id = f.function_id
177 | WHERE a.is_anomaly = TRUE
178 | ORDER BY a.z_score DESC, a.day DESC;
179 |

```

function_id	hash_app	total_executions	avg_memory_mb	total_memory_samples	criticality_score	trigger	total_invocations	avg_duration_ms	t
42388 94409f2485ebd997a61cbd06906595e4f3ef1846ed7406f9e3fa03cf4d5060a unknown 1119457 1649.3357856332075									
1684044453 266_511481316411 80958184 447784.0943351796 0, حملر 1119457 1649.3357856332075									
1684044453 94409f2485ebd997a61cbd06906595e4f3ef1846ed7406f9e3fa03cf4d5060a queue 1119457 1649.3357856332075									
1684044453 266_511481316411 80958184 447784.0943351796 0, حملر 1024730 285.4951695417021									
1187345938 73abb940a12d26e917f5e4cc093a6c95d11a 80958184 447784.0943351796 0, حملر 1024730 285.4951695417021									
1187345938 207.21346770230812 94517652 409892.77288954 0, حملر 1024730 285.4951695417021									
1187345938 207.21346770230812 94517652 409892.77288954 0, حملر 569413 184.18129254586526									
72869 4b7f5252a5f03341fd1268d543ad0119d2e89505a022f2851ba5d0c8a5e6b queue 0, حملر 569413 184.18129254586526									
980312059 160_59977451415718 4612183 227765.7370269001 0, حملر 569413 184.18129254586526									
980312059 160_59977451415718 4612183 227765.7370269001 event 345091 45.55331740257108									
82335 de811e1eb2a67b05b135+c8c4931cfcd4249b6b8612e950d886e9964ace795d event 211.880675488235 1384242 138037.0493880217 0, حملر 345091 45.55331740257108									
188148364 211.880675488235 1384242 138037.0493880217 0, حملر 345091 45.55331740257108									
82335 de811e1eb2a67b05b135+c8c4931cfcd4249b6b8612e950d886e9964ace795d unknown 1384242 138037.0493880217 0, حملر 345091 45.55331740257108									
188148364 211.880675488235 1384242 138037.0493880217 0, حملر 345091 45.55331740257108									
82199 f8bb64943c057c2b18f3b741b5b36a70e4bb8bb86e9c9475af160088111d78f unknown 14463298 80116.8461528276 0, حملر 222791 278.04206769835673									
82199 f8bb64943c057c2b18f3b741b5b36a70e4bb8bb86e9c9475af160088111d78f queue 14463298 80116.8461528276 0, حملر 222791 278.04206769835673									
120_01422080911283 120_01422080911283 14463298 80116.8461528276 0, حملر 222791 278.04206769835673									
82199 120_01422080911283 14463298 80116.8461528276 event 222791 278.04206769835673									
81808 228ef3c9f06cde19b4382e39c9339c9b62b521ae71d70ed02d75df205294a3 event 336710715 181.6915021027387 12283988 86829.7501059942 0, حملر 217073 17.071626300933133									
81808 228ef3c9f06cde19b4382e39c9339c9b62b521ae71d70ed02d75df205294a3 unknown 336710715 181.6915021027387 12283988 86829.7501059942 0, حملر 217073 17.071626300933133									
2053 228ef3c9f06cde19b4382e39c9339c9b62b521ae71d70ed02d75df205294a3 event 307326655 181.6915021027387 12283988 97950.1461880615 0, حملر 199374 3.711850568249604									
2053 228ef3c9f06cde19b4382e39c9339c9b62b521ae71d70ed02d75df205294a3 unknown 307326655 181.6915021027387 12283988 97950.1461880615 0, حملر 199374 3.711850568249604									
80854 407da163d137262786553d062d3486faf4a3faecf51c38bb68c3c3139f1b unknown 80854 189.970052172593 1720058 65143.044130365575 0, حملر 162856 247.40069685054368									
200053227 189.970052172593 1720058 65143.044130365575 queue 200053227 189.970052172593 1720058 65143.044130365575 0, حملر 162856 247.40069685054368									
1459 73fc37dfe254f2264ea/afce89c2891ffdbbe4c35522814e9a895c3e08b01a3 unknown 1459 73fc37dfe254f2264ea/afce89c2891ffdbbe4c35522814e9a895c3e08b01a3 unknown 155538 2.212928259415681									

شکل ۷: بخشی از خروجی کوئری ۷ ام

تابع های کمکی و ایندکس ها: ابتدا دوتابع کمکی در PL/pgSQL تعریف می شود:

که مجموع عناصر آرایه را به صورت BIGINT (برای آرایه usage)، و (arr SMALLINT[], consecutive_count INT) که وجود has_consecutive_zeros(arr) دنباله صفرهای پیوسته با طول دلخواه (مثلاً ۷۲۰ دقیقه) را تشخیص می‌دهد. سپس ایندکس‌های پوششی روی جداول فکت برای بهبود WHERE/JOIN‌ها ساخته می‌شوند.

یافتن روزهایی با ۱۲ ساعت بیکاری پیوسته: کوئری اول با JOIN بین functions و has_consecutive_zeros(inv.usage، fact_invocations_minutely_sparse، ۷۲۰، روزاتابع‌هایی را برای گرداند که حداقل ۱۲ ساعت متولی هیچ فراخوانی نداشته‌اند؛ همچنین با array_sum(inv.usage) مجموع فراخوانی همان روز (total_daily_invocations) محاسبه می‌شود.

امتیازدهی بحرانی بودن (Criticality): در بخش invocation_stats مجموع کل فراخوانی هر تابع از روی همه روزها بدست می‌آید. در duration_stats میانگین وزنی مدت اجرا (weighted_avg_duration) با وزن count و همچنین مجموع نمونه‌ها (total_executions) حساب می‌شود. در memory_stats نیز میانگین وزنی حافظه‌ای پ متناظر هر تابع (weighted_avg_memory) با وزن sample_count محسوبه می‌گردد. این آمارها در criticality_calculation بهم JOIN شده و یک criticality_score خطا ساخته می‌شود: ترکیبی از total_invocations (ضریب ۴/۰)، میانگین مدت (ضریب ۳/۰۰۰) و میانگین حافظه (ضریب ۳/۰۰۰). در خروجی، علاوه بر امتیاز، سطح کفی آن (بالا/بالا/متوسط/پایین خیلی) نیز بر اساس آستانه‌ها گزارش می‌شود.

تشخیص ناهنجاری حجمی روزانه: در daily_invocations تعداد روزانه فراخوانی برای هر function_avg_invocations (array_sum(usage) با) بدست می‌آید. سپس در anomaly_detection میانگین و انحراف معیار روزانه هر تابع محاسبه می‌شود. روزهایی را که (daily_count > 2 * avg_daily_count / std - می‌سازد. خروجی فقط موارد ناهنجار را به همراه شدت (شدید/متوسط/خفیف) برای گرداند و بر اساس Z_score مرتب می‌کند.

کاربرد عملی برای «حافظه زیاد اما استفاده کم»: با تلاقی نتایج سه بخش می‌توانید اپ/تابعی را بیابید که (۱) دوره‌های طولانی بیکارند، (۲) امتیاز بحرانی بودن شان پایین است (بهویژه مؤلفه memory)، و (۳) جهش‌های ناهنجار ندارند؛ این‌ها نامزدهای مناسب برای کوچکسازی منابع‌اند.

۸. ساخت یک معیار اهمیت برای هر تابع بر اساس مجموع فراخوانی، مدت اجرا و مصرف حافظه برای تعیین توابع حیاتی تر.

```

1 WITH invocation_stats AS (
2     SELECT
3         function_id,
4         SUM(inv_count) AS total_invocations
5     FROM (
6         SELECT
7             function_id,
```

```

8     day,
9     SUM(unnested_usage) AS inv_count
10    FROM fact_invocations_minutely_sparse,
11    LATERAL unnest(usage) AS unnested_usage
12    GROUP BY function_id, day
13  ) daily_invocations
14  GROUP BY function_id
15),
16 duration_stats AS (
17  SELECT
18    function_id,
19    CASE
20      WHEN SUM(count) > 0 THEN SUM(avg_ms * count) / SUM
21      (count)
22      ELSE 0
23    END AS weighted_avg_duration,
24    SUM(count) AS total_executions
25  FROM fact_function_duration_daily
26  GROUP BY function_id
27),
28 memory_stats AS (
29  SELECT
30    f.function_id,
31    CASE
32      WHEN SUM(m.sample_count) > 0 THEN SUM(m.avg_mb * m
33      .sample_count) / SUM(m.sample_count)
34      ELSE 0
35    END AS weighted_avg_memory
36  FROM functions f
37  JOIN fact_app_memory_daily m ON f.hash_app = m.hash_app
38  GROUP BY f.function_id
39)
40SELECT
41  f.function_id,
42  f.hash_app,
43  COALESCE(i.total_invocations, 0) AS total_invocations,
44  COALESCE(d.weighted_avg_duration, 0) AS avg_duration_ms,
45  COALESCE(d.total_executions, 0) AS total_executions,
46  COALESCE(m.weighted_avg_memory, 0) AS avg_memory_mb,
47  --

```

```

46     )COALESCE(i.total_invocations, 0) * 4.0 +
47     COALESCE(d.weighted_avg_duration, 0) * 0003.0 + --
48
49     COALESCE(m.weighted_avg_memory, 0) * (003.0 AS
50     criticality_score --+
51   FROM functions f
52   LEFT JOIN invocation_stats i ON f.function_id = i.function_id
53   LEFT JOIN duration_stats d ON f.function_id = d.function_id
54   LEFT JOIN memory_stats m ON f.function_id = m.function_id
55   ORDER BY criticality_score DESC;
56

```

function_id	avg_memory_mb	criticality_score	hash_app	total_invocations	avg_duration_ms	total_executions
42388 266.511481316111 21346770230812 42388 94409f2485ebd997a61cb0d6906595e4f3ef1846ed7406f9e3fa03cf4d5060a 1119457 1649.3357856332075 1684044453						
42388 266.511481316111 21346770230812 42388 94409f2485ebd997a61cb0d6906595e4f3ef1846ed7406f9e3fa03cf4d5060a 1119457 1649.3357856332075 1684044453						
207.21346770230812 207.21346770230812 207.21346770230812 207.21346770230812 427032 734bb9ab90a41d2e6917f75e6c093a6c95d114970e624b9975b98bad86c12f1ab 1024730 285.4951695417021 1187345930						
207.21346770230812 207.21346770230812 207.21346770230812 207.21346770230812 27032 734bb9ab90a41d2e6917f75e6c093a6c95d114970e624b9975b98bad86c12f1ab 1024730 285.4951695417021 1187345930						
160..59077445145718 160..59077445145718 160..59077445145718 160..59077445145718 72869 4b75f2532af503341fd1268d543ad0119dd2e8950a022f2851b4a5d0c8a5e0a 569413 184.18192254586526 900312659						
160..59077445145718 160..59077445145718 160..59077445145718 160..59077445145718 72869 4b75f2532af503341fd1268d543ad0119dd2e8950a022f2851b4a5d0c8a5e0a 569413 184.18192254586526 900312659						
82335 82335 82335 82335 82199 f8b0b649d3cf3bf32d2b1f3b741b5b36a7e48bbdb8a0c9475af16d0081011d78f 345091 45.55331749257198 188148364						
82199 82199 82199 82199 82199 f8b0b649d3cf3bf32d2b1f3b741b5b36a7e48bbdb8a0c9475af16d0081011d78f 345091 45.55331749257198 188148364						
120..91422089911283 120..91422089911283 120..91422089911283 120..91422089911283 88108 223013c9f06cd19ba384339c9339c9b62b521ae71d70e0d2775df20529a43 217073 17.071626300933133 336710715						
181..6915021027387 181..6915021027387 181..6915021027387 181..6915021027387 88054 86829..750195994 199374 3..711850568249604 307326655						
181..6915021027387 181..6915021027387 181..6915021027387 181..6915021027387 88054 86829..750195994 199374 3..711850568249604 307326655						
189..9708521726593 189..9708521726593 189..9708521726593 189..9708521726593 88054 65143..04413836557 162856 247..40069685054368 200053227						
189..9708521726593 189..9708521726593 189..9708521726593 189..9708521726593 88054 65143..04413836557 162856 247..40069685054368 200053227						
1459 1459 1459 1459 1459 7fc1c7df2e54f2204e8781ce09c2891ffddbebecc552281499853e0ff80801a3 155538 2..212928259415681 229001299						

شکل ۸: بخشی از خروجی کوئری ۸ ام

ابتدا در `invocation_stats` مجموع فراخوانی هر تابع (`total_invocations`) از تجمعی روزانه (`unnest(usage)`) محاسبه می شود؛ در `duration_stats` میانگین وزنی مدت اجرا (`weighted_avg_duration`) با وزن `count` و همچنین `total_executions` به دست می آید؛ و در `memory_stats` میانگین وزنی حافظه اپ متناظر هر تابع (`weighted_avg_memory`) بر پایه `sample_count` محاسبه می گردد.

سپس این آمارها با LEFT JOIN functions می شوند و یک امتیاز اهمیت خطی ساخته می شود: ترکیب `criticality_score` (`صریب ۰/۴`، `total_invocations` (`صریب ۰/۰۰۳`)، `weighted_avg_duration` (`صریب ۰/۰۰۳`) و `weighted_avg_memory` (`صریب ۰/۰۰۳`).

خروجی به صورت نزولی بر اساس `criticality_score` مرتب می شود تا توابع حیاتی تر در صدر قرار گیرند؛ ضرایب قابل تنظیم اند تا با مقیاس داده های شما و اولویت های کسب و کار تان هم تراز شوند.

۹. شناسایی توابع با نوسان زیاد در مدت زمان اجرا (اختلاف زیاد بین صدک ۷۵ و صدک ۹۵) که ممکن است ریسک نقض SLA داشته باشد.

```
1 SELECT
2   fdd.function_id,
3   fm.hash_function,
4   f.hash_app,
5   a.hash_owner,
6   AVG(fdd.p75) as avg_p75,
7   AVG(fdd.p99) as avg_p99,
8   AVG(fdd.p99 - fdd.p75) as avg_difference,
9   MAX(fdd.p99 - fdd.p75) as max_difference,
10  -- Volatility score
11  CASE
12    WHEN AVG(fdd.p75) > 0
13    THEN AVG(fdd.p99 - fdd.p75) / AVG(fdd.p75)
14    ELSE 0
15  END as volatility_ratio
16 FROM fact_function_duration_daily fdd
17 JOIN functions f ON f.function_id = fdd.function_id
18 JOIN apps a ON a.hash_app = f.hash_app
19 JOIN function_mapping fm ON fm.function_id = fdd.function_id
20 WHERE fdd.p75 IS NOT NULL AND fdd.p99 IS NOT NULL
21 GROUP BY fdd.function_id, fm.hash_function, f.hash_app, a.
22   hash_owner
23 HAVING AVG(fdd.p99 - fdd.p75) > 100 -- At least 100ms
24   difference
25 ORDER BY volatility_ratio DESC;
```

function_id	hash_function	hash_owner	avg_p75	avg_p99	hash_app	avg_difference	max_diff
erence	volatility_ratio						
e6	526b1cccd3a552b9e40612c15f231f500e133362d4e9446ea36442fe9b8e76ff	faae977e2ed60b3e1f283a4bbd68ed5004854fa44e50c7e67123c39d2a					
66322	7e6594584d03424815f3025ca4d00de9761977152e6293534a6e308		1	57457.71428571428	57456.71428571428		
42491	824c216d1a5718c6762c737c8ec00670411da1811589ca2321acf9e77333	649494964d0386b6cd8a5279232c9592e5cfaaa0b23e53479a4f57473ca70					
bd	778ad8880d2c9lac9850e38947922670e83fcaf95816789cd1c6664ab		1.0	1.0285714285714286	1.028571428572	1.0285714285714286	
50464	9053040c7ecfa331483281f2e26a0447bcdfcf9749cb214d5a2e6da70bde67	ba5ed392e0ed65b6f4fb8cd4494fa88fe24433398f044475e2448b99b					
54	cbe0d8e84e75f5d8f5e7f52cb0c8661726e736646e5294ed96e8ff2f23b13c	0.07142857142857142857	0.07142857142857142857	0.07142857142857142857	0.07142857142857142857	0.07142857142857142857	
129							
34334	692a379fc2f6a1f8377d68849e02fcf78724a1178f547bf3faadd50eb0ddf139	9543fdabb5b5ebc4975a7e13aaaf77e1dfb1a215395958bd0d454cb59c2437					
0b	1c8df546eebb1910d6897afec2fcf40d88919122c7a4d7693d953c642cb8059eee	8.857142857142858	77823.21428571429	77811.35714285714	1050.142857142857	1050.142857142857	
92080	8785.491935483871						
40468	766c61295c9e64015f0d1f3d38283c918f8fa5f7d76746460ba389afc557871	8d4eb51c26b4fc1b0fa76cd17238d8fe1e39423b43cda1e2cc1c9e1f1f0b583					
if	fbec07b7a4e4b4e0d31e2a45ff5bd5cc6d04af99bc7169caf097af874a84	94927.14285714286	94914.4285714286	94914.4285714286	94914.4285714286	94914.4285714286	
158265	7456.714285714286						
17640	35d41c864201f5330ce03639a160f857c13dc8157338f5e2a2f7517cbef8860	90273308319e53bcf6f6be96d366dd21cf0093b8febaaacbc76b12bc68e2					
e9	b9b4f7eb937b13bac0421a1eccdc8880eee88aaf05bc26741dec0b0d1a3b	0.162857142857142857	0.162857142857142857	0.162857142857142857	0.162857142857142857	0.162857142857142857	
1169	6900						
50516	987779140ffff39a64fde165a24d02f2895a6607411611af19874afc068d3	4854551b254831f6e727fb7297f7fc489539991f62ad77e3c4e3aff770cf					
db	33f3e93a4f5232c2679e21af2b382b0a01699ba1bc210386801932b68da257b	0.142857142857142857	0.142857142857142857	0.142857142857142857	0.142857142857142857	0.142857142857142857	
1188	967						
71103	a6a6339877a39447aa480d7d2eh139232981c2e09hebe7e188f63577c5acfc6a	910a1b65ec432ed535h3n8a84da590d0767349c16a92067718f999b					
5e	e1e695a82badf45bed51138a99861d4b951923822b8e4461345fb236922b	0.162857142857142857	0.162857142857142857	0.162857142857142857	0.162857142857142857	0.162857142857142857	
993	6661						
us	80223	f6b5182a32c23a1e8a881c9cfaa0542476ae5f1d71f653bb89708	31d6f7bc0d74930a1f1ab0a0dfbc6e4131d03861296f1604f60aa51cbd55				
40	13ac6b9a0c9b0bf547cb5a9eae6b79215793dc5a942bdac36e1chfac0f4612c	4612.5	4612.5	4612.5	4612.5	4612.5	
7008	5454.6						
43898	86ce0e5a6c379c30968b388d075acc1c5c03f08d9f826288e0e6f6f6fcf5	8703290d6289ef9299bb461290d44a6c6d6f508677fa019392b995dab01f					
33b86268cf88d4e755d1f0e82dd1d4b4e5120297f2e473ed994abb3fb4be	23.57142857142857	124628.42857142857	124604.42857142857	124604.42857142857	124604.42857142857	124604.42857142857	
112456	5286.666666666666						
9573	1d3405e1d9a74db0ff9924b481e0c15a1b3fc703a99751b881f0564	fa0e6b6bf5f46bd7e1b1bc38522fc9c0975271437feade10dbfc054a1c24					
:	:						

شکل ۹: بخشی از خروجی کوئری ۹ ام

در آرایه usage برای هر daily_invocations (function_id, day) به ردیفها باز می‌شود و مجموع دقیقه‌ها به عنوان daily_total محاسبه می‌گردد. میانگین روزانه هر تابع (avg_daily) و همچنین بیشینه آن (max_daily) در استخراج می‌شود.

سپس با الحق به جداول مرچ، نسبت daily_total / NULLIF(avg_daily,0) به عنوان ratio محاسبه و فقط روزهایی نگه داشته می‌شوند که daily_total > 2 * avg_daily. خروجی بر اساس ratio نزولی مرتب می‌شود تا ناهنجاری‌ها در صدر بیانند.

۱۰. یک اپلیکیشن خاص را انتخاب کنید و بررسی کنید میانگین حافظه استفاده شده آن در طول ۱۲ روز چگونه تغییر کرده است.

```

1 WITH memory_trend AS (
2   SELECT
3     hash_app,
4     day,
5     avg_mb,
6     sample_count,
7     LAG(avg_mb, 1) OVER (PARTITION BY hash_app ORDER BY
8       day) as prev_day_mb,
9     AVG(avg_mb) OVER (PARTITION BY hash_app ORDER BY day
10    ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) as
11    moving_avg_3day
12    FROM fact_app_memory_daily
13    WHERE hash_app = 'YOUR_HASH_APP' -- Replace with actual
14    hash

```

```

11      AND day <= 12
12  )
13  SELECT
14      hash_app,
15      day,
16      avg_mb,
17      sample_count,
18      prev_day_mb,
19      avg_mb - prev_day_mb as daily_change_mb,
20      CASE
21          WHEN prev_day_mb > 0
22              THEN ((avg_mb - prev_day_mb) / prev_day_mb) * 100
23          ELSE 0
24      END as daily_change_percent,
25      moving_avg_3day,
26      MIN(avg_mb) OVER () as min_mb_period,
27      MAX(avg_mb) OVER () as max_mb_period,
28      AVG(avg_mb) OVER () as avg_mb_period
29 FROM memory_trend
30 ORDER BY day;
31

```

	hash_app	day	avg_mb	sample_count	prev_day_mb	daily_change_mb	daily_change_percent	moving_avg_3day	min_mb_period	max_mb_period	avg_mb_period
1	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	1	113	25154							
2		113	113	118	115.0833333333333333						
3	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	2	113	20373	113	0					
4		113	113	118	115.0833333333333333						
5	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	3	114	19593	113	1	0.8849557522				
6		113	114	118	115.0833333333333333						
7	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	4	116	22488	114	2	1.7543859649				
8		113	116	118	115.0833333333333333						
9	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	5	116	21599	116	0					
10		113	116	118	115.0833333333333333						
11	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	6	115	8151	116	-1	-0.8620689655				
12		113	115	118	115.0833333333333333						
13	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	7	117	7103	115	2	1.7391304347				
14		113	117	118	115.0833333333333333						
15	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	8	113	23820	117	-4	-3.418803418				
16		115	113	118	115.0833333333333333						
17	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	9	114	19686	113	1	0.8849557522				
18		113	114	118	115.0833333333333333						
19	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	10	117	16729	114	3	2.631578947				
20		113	117	118	115.0833333333333333						
21	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	11	115	20233	117	-2	-1.7894017894				
22		113	115	118	115.0833333333333333						
23	68692c2002ff07ce4d63ee9c9a9e7d5a6d32639aa59b1ab999694c5872f5f5186a	12	118	18875	115	3	2.60869695652				
24		113	118	118	115.0833333333333333						
25	(12 rows)										
26	[END]										

شکل ۱۰: بخشی از خروجی کوئری ۱۰ ام

در دریافت برای یک hash_app خاص (جایگزین YOUR_HASH_APP) و بازه

روزه ۱۲ با LAG مقدار روز قبل (day <= 12) و با پنجره (prev_day_mb) میانگین متوجه ROWS BETWEEN 2 PRECEDING AND CURRENT ROW میانگین محاسبه می‌شود.

در خروجی، تغییر روزبه روز مطلق و درصدی ($(avg_mb - prev_day_mb) / prev_day_mb * 100$) بهمراه حداقل/حداکثر اینگین کلی همین دوره AVG/MAX/MIN روى کل نتیجه گزارش می‌گردد و بر اساس day مرتب می‌شود. نکته: اگر فیلد day تاریخ کامل است، شرط بازه ۱۲ روزه را متناسب با قالب داده خود تنظیم کنید.

۸ استریم و دریافت اطلاعات برخط

در این بخش یک سرویس سبک برای دریافت آنلاین داده‌ها پیاده‌سازی شده تا ورودی‌ها (JSON یا فایل‌های CSV) بلافاصله با سیاست نرم‌السازی پروژه ذخیره شوند. سرویس، مقدار day را از درخواست می‌گیرد، سلسه‌مراتب Owner → App → Function را تضمین می‌کند، hash_function را به function_id نگاشت raw_invocations، fact_function_duration_daily، fact_app_memory_daily درج می‌کند. کل فرآیند در تراکنش انجام می‌شود و در صورت خطا بازگشت داده (rollback) دارد.

- برقراری اتصال به پایگاهداده و بازگشت یک کانکشن آماده تراکنش: **get_connection()**
- **hash_function, insert_owner_app_function(cur, hash_owner, hash_app, trigger=None)**: تضمین وجود owner و app (با درج بدون تداخل)، نگاشت hash_function به fact_app_memory_daily (تفصیل در صورت عدم وجود)، و درج رکورد function_id
- خواندن فایل بارگذاری شده و تبدیل به فهرست رکوردها (دیکشنری سطری): **parse_csv(file)**
- یکنواختساری ورودی JSON به قالب «فهرست رکوردها» حتی اگر یک شیء تکی ارسال شده باشد: **parse_json(data)**
- شمارش دقیقه‌های ۱۴۴۰ را به ردیف‌های raw_invocations در minute, day, (function_id, count) تبدیل و درج می‌کند: **:insert_invocations(data, day, cur)**
- درج آمار روزانه مدت‌زمان اجرا (میانگین، کمینه/بیشینه، day, count) در fact_function_duration_daily صدکها: **:insert_durations(data, day, cur)**
- درج شاخص‌های حافظه برنامه (میانگین و صدکها app) در fact_app_memory_daily در sample_count با تکیه بر سطح بهمراه **:insert_memory(data, day, cur)**

• نقطه ورود مشترک برای هر سه نوع داده؛ تعیین نوع ورودی **process_upload(data_type)** .
فراخوانی تابع درج مناسب، و مدیریت CSV) یا، JSON commit/rollback.

• **upload_durations_endpoint() / upload_invocations_endpoint()** / .
مسیرهای HTTP که درخواست بارگذاری را دریافت و به **upload_memory_endpoint()** هدایت می‌کنند. **process_upload**

۹ بهینه‌سازی نحوه کار با داده و طراحی پایگاهداده

این بخش شرح می‌دهد که چگونه طرح داده (schema)، مسیر ورود داده و ایندکس‌گذاری را به‌گونه‌ای طراحی کردۀ‌ایم که حجم ذخیره‌سازی کاهش یابد، زمان پاسخ‌دهی کوئری‌ها بهبود پیدا کند و هزینه محاسباتی (CPU/IO) برای تحلیل‌های روزانه و دقیقه‌ای مدیریت شود. علاوه بر آن، تنظیمات سطح سرور برای استفاده بهینه از حافظه و موازی‌سازی اعمال شده است. در انتهای بخش، مطابقت کار انجام‌شده با نیازمندی‌های «بهینه‌سازی نحوه کار با داده» جمع‌بندی می‌شود.

۱.۹ نمای کلی طرح داده

• جداول هویت و نگاشت: **functions** (با کلید خارجی به مالک)، **apps** (با کلید خارجی به مالک)، **owners** (تابع‌ها با **hash_function** (نگاشت **function_id**, **trigger**) و **function_mapping** (**function_id** به کلید مرکب **function_id** با سکانس خودکار).

• جداول فکت روزانه/دقیقه‌ای:

fact_invocations_minutely_sparse(**function_id**, day, usage[1440]) ◻
شمار فراخوان‌ها برای هر دقیقه روز به صورت آرایه ۱۴۴۰ تایی. این چیدمان، نسبت به ذخیره ۱۴۴۰ ردیف برای هر روز/تابع، فضای ذخیره‌سازی و سربار ایندکس را بهشت کاهش می‌دهد و بازیابی‌های روزانه را سریع می‌کند.

fact_function_duration_daily ◻
خلاصه آماری مدت زمان اجرا (میانگین/کمینه/بیشینه/صدک‌های ۰، ۱، ۲۵، ۵۰، ۷۵، ۹۹ به همراه شمارش).

fact_app_memory_daily ◻
آمار مصرف حافظه اپلیکیشن (میانگین و صدک‌های ۱ تا ۱۰۰).

raw_invocations(**function_id**, day, minute, count)
لاغ خام برای ورود سریع: به عنوان بافر ورودی رکورد-محور.

۲.۹ مسیر ورود داده و بهروزرسانی درجا

برای رسیدن به نوشتمن سریع و کم‌هزینه در جدول آرایه‌ای، یک مسیر دو مرحله‌ای طراحی شده است:

۱. درج رکورد خام در raw_invocations (از طریق اسکریپت writer یا API).
۲. یک تریگر AFTER INSERT با فراخوانی تابع increment_minute_fast مقدار دقیقه مربوط را مستقیماً در آرایه usage به روزرسانی می‌کند. اگر رکورد روز/تابع موجود نباشد، درجا ساخته شده و مقدار دقیقه افزایش می‌یابد. این الگو، هزینه upsert دقیقه‌ای را کم می‌کند و مسابقه همزمانی را به حداقل می‌رساند.

نمونه منطق به روزرسانی:

اگر سطر آن روز/تابع وجود دارد، همان عنصر آرایه افزایش یابد؛ و گرنه سطر ایجاد و سپس همان عنصر به روزرسانی شود.

```

1 UPDATE fact_invocations_minutely_sparse
2 SET usage[p_minute] = LEAST(usage[p_minute] + p_count, 2147483647)
3 WHERE function_id = p_function_id AND day = p_day;
4
5
6 IF NOT FOUND THEN
7   INSERT INTO fact_invocations_minutely_sparse(function_id, day,
8     usage)
9   VALUES (p_function_id, p_day, array_fill(0::integer, ARRAY[1440])
10    );
11  UPDATE fact_invocations_minutely_sparse
12  SET usage[p_minute] = LEAST(usage[p_minute] + p_count,
13    2147483647)
14  WHERE function_id = p_function_id AND day = p_day;
15 END IF;

```

۳.۹ ایندکس‌گذاری و منطق انتخاب ستون‌ها

با توجه به الگوهای پر تکرار در JOIN/WHERE، ایندکس‌ها روی ستون‌هایی قرار گرفته‌اند که بیشترین کاهنده دامنه جستجو هستند:

- پرس‌وجوهای زمانی و تابعی:

CREATE INDEX idx_fim_function_day ON fact_invocations_minutely_sparse(function_id, \sqcup
day);

CREATE INDEX idx_fim_day ON fact_invocations_minutely_sparse(day); \sqcup

این دو ایندکس، بازیابی سطرهای «یک تابع در یک روز» یا «همه توابع در یک روز» را بسیار سریع می‌کنند و قبل از UNNEST دامنه را کوچک می‌سازند.

• نگاشت و ابعاد:

CREATE INDEX idx_function_mapping_hash ON function_mapping(hash_function); ۰
برای .function_id به hash_function سریع resolve

CREATE INDEX idx_apps_owner ON apps(hash_owner); ۰
برای تجمعی ها و فیلترها بر مبنای مالک.

• ورود رکورد خام:

CREATE INDEX idx_raw_invocations_function_day_minute ON raw_invocations(function_id, day, minute);
به سرعت رکوردهای مرتبط trigger را پیدا کند (و در سناریوهای گزارش گیری از لاغ خام نیز کارا باشد).

۴.۹ تنظیمات کارایی سطح سرور

برای کاهش I/O موقت و بهبود کارایی کوئری های تحلیلی:

• فضای موقت روی RAM: ایجاد TABLESPACE در /mnt/ramdisk و تنظیم temp_tablespaces برای عملیات موقتی (سُورت/هش) تا از دیسک سمت I/O دور شویم.

• حافظه و موازی سازی: maintenance_work_mem=4GB ,work_mem=128MB ,shared_buffers=16GB ,max_parallel_workers=8 ,max_parallel_workers_per_gather=4 ،

• اعمال آنلاین تنظیمات: SELECT pg_reload_conf(); برای بارگذاری سریع پارامترها بدون توقف سرویس.

۵.۹ دلایل طراحی و اثرات مورد انتظار

ذخیره سازی آرایه ای (Sparse Array). به جای ۱۴۴۰ ردیف در هر روز/تابع، یک سطر با آرایه ۱۴۴۰ تایی نگه می داریم. این کار:

۱. اندازه جدول و ایندکس را کاهش می دهد،

۲. اسکن های روزانه را سریع می کند (فقط یک سطر به ازای هر تابع/روز)،

۳. و با تریگر به روزرسانی درجا، هزینه نوشتن دقیقه ای را ثابت و کوچک نگه می دارد.

ایندکس های هدفمند. اکثر کوئری ها با قیود ... WHERE day=... و/یا WHERE function_id=... آغاز می شوند، بنابراین ایندکس های (day) و (function_id, day) بیشترین کاهش دامنه جستجو را ایجاد می کنند و قبل از UNNEST، هزینه پردازش آرایه را کم می کنند. ایندکس های hash_function و hash_owner نیز مسیرهای GROUP BY/JOIN را روان کرده و dimension را شتاب می دهند.

تنظیمات سرور. work_mem و shared_buffers می‌دهد؛ temp_tablespaces روی RAM فشار I/O را می‌کاهد؛ موازی‌سازی برای کوئری‌های سنگین تحلیلی (به‌ویژه UNNEST+تجمعی) سودمند است.

۶.۹ مطابقت با نیازمندی‌های «بھینه‌سازی نحوه کار با داده»

- بھینه‌سازی نحوه ذخیره‌سازی: انجام شد. تغییر مدل به آرایه دقیقه‌ای، فشرده‌سازی ساختاری و کاهش چشمگیر کار دیناتی سطروی.
- ایندکس‌گذاری و سنجش اثر: انجام شد. ایندکس‌های کلیدی ایجاد شده‌اند و برای سنجش قبل/بعد، اجرای اسکریپت‌های پایه (01_schema، 02_load_data) و سپس 03_optimized پیشنهاد و استفاده شده است.
- بھینه‌سازی در مسیر نوشتن: انجام شد. تریگر و تابع به روزرسانی درجا، upsert دقیقه‌ای را به عملیات ثابت (O(1)) تبدیل می‌کند.
- بھینه‌سازی سطح سرور: انجام شد. تنظیمات حافظه/موازی‌سازی و temp_tablespaces روی RAM اعمال شده است.
- قابلیت سناریوهای همزمانی: زیرساخت داده و ایندکس‌ها آماده اجرای سناریوهای چندکلابینتی است؛ آزمون‌های بار و Connection Pool نیز در لایه اپلیکیشن قابل افزودن هستند.

۷.۹ خلاصه ادعاهای بھینه‌سازی

۱. کاهش حجم و هزینه I/O: با مدل آرایه‌ای و کاهش تعداد ردیف‌ها و ایندکس‌ها.
۲. بهبود زمان پاسخ: با ایندکس‌های (function_id, day) و (day) برای محدودسازی سریع دامنه UNNEST قبل از.
۳. نوشتن کار: تریگر AFTER INSERT و increment_minute_fast بدون اسکن یا قفل گذاری گسترده.
۴. کارایی تحلیلی پایدار: تنظیم shared_buffers/work_mem و موازی‌سازی، هزینه کوئری‌های پنجره‌ای و صدکی را پایین می‌آورد.