Department of Computer Engineering

Project 2

Lecture 3, 4, 5

Course Principles of Database Design

Dr. Shahriari

Spring 2023

مقدمه

در این پروژه قصد داریم یک سامانه بانکداری ساده را به شکل event-driven پیاده‌سازی کنیم. در روش event-driven، به جای اینکه نتایج یک عمل را ثبت کنیم، مجموعه اعمال را ثبت می‌کنیم و در نهایت در زمان نیاز، اعمال را شبیه‌سازی کرده و وضعیت کنونی را به دست می‌آوریم که آن را به نام «snapshot» سامانه ذخیره می‌کنند. از بزرگترین مزایای این روش، قابلیت اصلاح خطاها و نگهداری تاریخ و شفافیت آن است. به طور مثال اگر درصد بهره یک حساب به اشتباه ثبت شده باشد، بدون داشتن event ها نمی‌توانیم عقب برگردیم و مقدار درست آن را جایگزین کنیم و ماوالتفاوت بگیریم. در واقعیت هم، علت تاخیر در تراکنش‌های پایا و ساتنا، و خشک شدن تراکنش‌ها در برخی ساعت‌های نیمه شب همین روش هست.

مفاهیم اولیه

برای شروع پروژه نیاز هست یک سیستم مدیریت دیتابیس روی کامپیوتر خود نصب کنید که می‌توانید از mysql، mariadb یا postgresql استفاده کنید. پیشنهاد ما، استفاده از postgresql هست که سرعت بسیار بالایی دارد و قابلیت‌های زیادی به sql ساده (مانند جستجوی متن) به آن اضافه شده. نصب و راه‌اندازی پایگاه داده به صورت داکر کانتینر نمره امتیازی دارد.

جداول سامانه

در سامانه بانکداری برای هر مشتری یک حساب کاربری تعریف می‌شود. هر حساب کاربری مشخصات ذیل را در خود ذخیره می‌کند. توجه کنید که شناسه کاربر باید یکتا باشد و توسط سامانه مشخص شود. توجه کنید که برای پیاده‌سازی چنین عملیاتی می‌بایست Triggers تعریف کنید. به عنوان مثال یک Trigger تعریف کنید که به ثبت اطلاعات در پایگاه‌داده حساس باشد و با کمک نام و نام خانوادگی یک شناسه کاربری یکتا ایجاد کند. شناسه حساب یک رشته ۱۶ کاراکتری از اعداد می‌باشد. رمز عبور نیز باید باید به صورت هش (رمزنگاری شده) ذخیره شود. در ادامه به پیاده‌سازی بیشتر می‌پردازیم.

account:

- username: <unique username>
- accountNumber: <16 digit ID>
- password: <hashed password>
- first_name: <first name>
- last_name: <last name>
- national_id: <10 digit national ID>
- date_of_birth: <date of birth in yyyy/mm/dd format>
- type: <client or employee>
- interest rate: <interest rate of account. Equals to ۰ if type is employee>

login_log:

در این جدول تاریخچه ای از افرادی که وارد سامانه شدند نگه می‌داریم. بعدا در عملیات‌ها، آخرین فردی که لاگین کرده را به عنوان کاربر فعال در نظر می‌گیریم.

- username: <foreign key>
- login_time: <timestamp>

transactions:

جدول event های ماســـت که در واقع چهار نوع دارد: واریز (deposit)، برداشـــت (withdraw)، انتقال وجه (transfer) و واریز بهره (interest).

- type: <deposit, withdraw, transfer, interest>
- transaction_time: <timestamp>
- from: <source AccountNumber foreign key>
- to: <destination AccountNumber foreign key>
- amount: <amount of transaction>

در حالت‌های واریز و واریز بهره، فیلد from برابر با NULL می‌شود. در حالت برداشت از حساب فیلد To برابر با NULL می‌شود.

latest_balances:

این جدول، نقش آخرین snapshot موجود در سیستم را دارد.

- accountNumber: <AccountNumber foreign key>
- amount: <account balance at the time of snapshot>

snapshot_log:

در این جدول زمان ساخته شدن اسنپ‌شات‌های سیستم را ثبت می‌کنیم.

- snapshot_id: <auto increment ID>
- snapshot_timestamp: <timestamp>

عملیات‌ها

هر کدام از این عملیات‌ها باید به شکل یک procedure تعریف شوند. برای برگرداندن پیام مناسب می‌توانید از یک SELECT با یک متن ثابت استفاده کنید.

ثبت‌نام (register):

این عملیات اطلاعات موجود در account را می‌گیرد و فرد را در سامانه ثبت نام می‌کند. دقت کنید که رمزی که این فرآیند می‌گیرد به شکل غیر رمزنگاری شده هست و خود عملیات آن را رمزنگاری می‌کند. همچنین افراد زیر ۱۳ سال حق ثبت حساب در بانک را ندارند. هنگام ثبت نام فرد در حساب خود صفر تومان دارد که باید در latest_balances اضافه شود.

ورود به سامانه (login):

این عملیات نام کاربری و رمز عبور غیر هش شده را ورودی می‌گیرد و در صورت صحت، این ورود را به login_log اضافه می‌کند.

واریز وجه (deposit):

این عملیات یک مقدار ورودی میگیرد و آن را به عنوان یک event برای فردی که آخرین لاگین را انجام داده ثبت می‌کند. دقت کنید که مقدار موجودی درون latest_balances را تغییری نمی‌دهیم.

برداشت وجه (withdraw):

مشابه با واریز، یک مقدار ورودی می‌گیرد و آن را به صورت event ثبت می‌کنیم.

انتقال وجه (transfer):

یک مقدار و یک شناسه حساب دریافت می‌کند و در صورت وجود داشتن حساب مقصد، یک event ثبت می‌کند.

پرداخت بهره (interest_payment):

این عملیات نیز ورودیی ندارد.

برای هر حساب بر اساس درصد بهره ثبت شده موقع ثبت نام، یک event ثبت می‌شود.

به روز رسانی موجودی‌ها (update_balances):

این عملیات ورودیی ندارد. بر اساس event های ثبت شده از زمان ساخت آخرین snapshot، جدول latest_balances را به روز رسانی می‌کنیم. بعد از انجام به روز رسانی‌ها، یک سطر جدید به جدول snapshot_log می‌سازیم و در نهایت یک جدول جدید به نام snapshot_id می‌سازیم که مقادیر آن، کپی مقادیر latest_balances است و مقدار id همان آیدی ثبت شده در snapshot_log است.

دقت کنید در زمان به روز رسانی موجودی‌ها بر اساس eventها، ممکن است برخی تراکنش‌ها حساب فرد را منفی کنند که در این صورت این تراکنش اعمال نمی‌شود. این عملیات فقط در حالتی مجاز هست که فرد لاگین شده، از نوع کارمند باشد. در یک سیستم واقعی همچین عملیاتی به صورت زمان دار ساعتی یا دقیقه‌ای اجرا می‌شود اما درون این پروژه فراخوانی دستی کافی‌ست.

گرفتن موجودی (check_balance):

مقدار موجودی فرد لاگین شده در latest_balances به او نشان داده می‌شود. در صورتی که کاربر می‌خواهد موجودی دقیق و اخیر خود را ببیند باید ابتدا یک کارمند بانک update_balances را برای او فراخوانی کند.

رابط کاربری

در بخش آخر، یک برنامه با زبان دلخواه بنویسید که نقش رابط کاربری را دارد. فرد ابتدا باید لاگین یا ثبت‌نام کند، و سپس به سایر قابلیت‌های سامانه دسترسی پیدا می‌کند. رابط کاربری که برای پروژه اجباری هست، به شکل CLI یا همان اجرا در کنسول هست که کاربر با وارد کردن شماره عملیات، آن را صدا می‌زند. در این بخش حق استفاده از ORMها را ندارید و تنها کار آن، اتصال به پایگاه داده، ورودی گرفتن از کاربر و فراخوانی procedure هاست. پیاده‌سازی رابط گرافیکی نمره امتیازی دارد.

توضیحات بیشتر

توضیح ۱: هش کردن به چه دردی میخورد؟

توابع hash، توابع یک طرفه‌ای و (در حالت کلی) یک به یکی هستند که مقدار ورودی را به هم می‌ریزند. به همین علت برای نگهداری از رمزها، از این توابع استفاده می‌کنیم تا در صورت هک شدن یا مشاهده جدول توسط کارمند، باز هم نفوذ به حساب کاربران کار ساده‌ای نباشد. برای توضیح بیشتر راجع به ایمن‌سازی سامانه‌ها می‌توانید این ویدیو را ببینید (در هش کردن در پروژه نیازی به نگه‌داری الگوریتم نیست). پیاده سازی salt نمره امتیازی دارد.

توضیح ۲: چرا جدول LATEST_BALANCES یک جدول جداست؟ نمیشد موجودی کاربر را در ACCOUNTS نگه داریم؟

در سیستم‌های event driven، معمولا یک جدول برای eventها داریم، و یک جدول جداگانه برای خروجی این event ها در نظر می‌گیریم. برای بررسی علت این طراحی، باید قدم قدم منطق event driven و بهینه سازی آن را طی کنیم.

۱. می‌توان در زمان انجام هر تراکنش، کل event ها را محاسبه کرد تا ببینیم کاربر موجودی کافی دارد یا نه، که این کار بسیار کند هست.
۲. می‌توان در کنار نگهداری از eventها، به صورت دوره‌ای تمام eventها را از اول محاسبه کنیم و مقادیر حساب‌ها را به عنوان snapshot (تصویر) نگه داریم. اما این کار بعد از چند سال بانکداری و بالا رفتن تعداد eventها از لحاظ زمانی معقول نیست.
۳. می‌توانیم از snapshot قبلی استفاده کنیم و صرفا تراکنش‌های مابین snapshot قبلی و زمان کنونی را استفاده کنیم تا snapshot جدید بسازیم. اما در صورت وجود خطای محاسبه یا انسانی (به طور مثال در محاسبات مالیات یا بهره)، باید snapshot را حذف کرده و تمامی eventها را از اول حساب کنیم.
۴. می‌توانیم علاوه بر روش قبلی، هر زمان که یک snapshot جدید می‌سازیم، مقادیر آن را در یک جدول جداگانه همراه با زمان ثبت snapshot نگهداری کنیم. حالا اگر یک خطا رخ داد، تمام snapshotهایی که از زمان رخداد خطا تا الان وجود دارند را مردود (invalidate) می‌کنیم و از snapshotای که قبل از رخداد خطا داریم (که مطمئنیم صحیح هست) کمک می‌گیریم تا مقادیر درست را حساب کنیم. مشکل اساسی این راه حل فضایی هست که نگهداری تمامی snapshotها می‌گیرد.

مطالعه بیشتر

۱. یکی دیگر از مهم‌ترین کاربردهای سیستم‌های event driven، استفاده آن‌ها در پایگاه داده‌های توزیع‌شده روی چند سیستم است. در این سیستم‌ها ممکن است در هر لحظه مقادیر نشان داده شده توسط سیستم درست نباشد، اما با گذر زمان این تضمین می‌شود که مقادیر نمایش داده شده به مقادیر واقعی تبدیل می‌شوند. یکی از مثال‌های آن، شمارنده لایک در یوتیوب یا توییتر هست که در سرورهای متفاوتی این عملیات‌ها را ذخیره می‌کنند تا سرعت سایت بالا برود. در مثال ما، اگر هر دقیقه یک بار update_balances را صدا می‌زدیم در واقع این تضمین را برقرار می‌کردیم. در مدل‌های پیچیده‌تر با اثبات ریاضی نشان می‌دهیم همگرایی به مقادیر واقعی وجود دارد که خود باعث تضمین صحت می‌سازد.

به این روش، eventual consistency می‌گویند.

۲. یکی از الگوهای بسیار رایج در طراحی سیستم‌ها جداسازی مسئولیت کوئری و دستور، یا Command Query Responsibility Segregation هست که در آن عملیات‌های تغییر دهنده‌ی اطلاعات، تا جای ممکن از عملیات‌های خواننده اطلاعات در پایگاه داده جدا سازی می‌شوند. یکی از علت‌های این کار، این است که بهینه‌سازی لازم برای خواندن سریع، با بهینه‌سازی لازم برای نوشتن سریع، از لحاظ نرم‌افزار مدیریت پایگاه داده با هم بسیار متفاوت اند. همچنین در سیستم‌های توزیع‌شده، برای تسریع پاسخ به کوئری‌های کاربر (که در اکثر مواقع بیشتر از عملیات‌های تغییر دهنده داده اند)، نیاز داریم یک تجمیع از داده‌های توزیع‌شده داشته باشیم که سیستم از روی آن، مشابه با cache داده‌ها را بخواند. در پروژه ما، این نقش را همان جدول latest_balances داشت. برای مطالعه بیشتر به این لینک مراجعه کنید.

به نکات زیر توجه کنید.

- این پروژه باید به صورت انفرادی انجام شده و کد های مشابه چک خواهند شد.
- این پروژه تحویل آنلاین داشته و کارکرد برنامه شما به صورت عملی تست خواهد شد.
- پروژه را در قالب یک فایل زیپ به نام db-proj۱-studentnumber.zip بارگزاری کنید.
- در صورت داشتن هرگونه ابهام و سوال از تدریسیاران درس کمک بگیرید.