گزارش بخش نهایی

دوره کارآموزی کرونایی

احمد زعفرانی

شرکت پیشتاز رایانش

مقدمه

این پروژه از چندین فایل تشکیل شده است : دو فایل به زبان C، یک فایل بش‌اسکریپت و یک فایل پایتون. برای اجرای پروژه، ابتدا مطابق دستورالعمل فایل ReadMe، فایل ... را با یک ویرایشگر متنی بازکرده و خط زیر را در آن پیدا کنید:

local all all peer

سپس کلمه «Peer» را پاک کنید و کلمه «Trust» را به جای آن بنویسید و تغییرات را در فایل ذخیره کنید. همچنین لازم است تا پکیج‌هایی را دانلود و نصب کنید تا برنامه به درستی کار کند (پیشفرض توسعه دهندگان این است که برنامه هایی مانند کامپایلر gcc، پایتون3 و پایگاه‌داده PostgreSQL روی سیستم به درستی نصب شده‌اند). حال کافی است پروژه را از ریپازیتوری گیت Clone کنید، سپس با باز کردن یک ترمینال در مسیری که پروژه را در آن دانلود کرده اید، فایل بش‌اسکریپت را در اجرا کنید. همچنین برای اجرای فایل پایتون، می‌توانید یک ترمینال جدید باز کنید و برنامه پایتون را در آن اجرا کنید.

با دنبال کردن مسیر فوق، می‌توان داده های حاصل از فروش را تجمیع کرد و مورد تحلیل قرار داد. در ادامه به شرح بیشتر نحوه عملکرد هر کدام از ماژول ها و توضیح تحلیل های چاپ شده توسط برنامه پایتون می‌پردازیم.

فایل بش‌اسکریپت

این فایل وظیفه آماده سازی پیشنیاز ها برای اجرای صحیح برنامه C را دارد. این دستورات به شرح زیر می‌باشند:

1. پاک کردن پوشه /tmp/final_project و محتوای آن از روی سیستم (تذکر: این کار برای اطمینان از اینکه داده‌های نامربوط در پایگاه داده وارد نمی شوند انجام می شود. اگر مایل به اجرای چنین دستوری نیستید، این خط از فایل بش‌اسکریپت را کامنت کنید: *rm -rf /tmp/final_project*)
2. ایجاد دوباره این پوشه
3. کامپایل کردن و اجرای فایل pre_aggregation.c
4. کامپایل کردن فایل reader.c
5. اجرای حلقه زیر تا بینهایت:

- فرستادن درخواست به آدرس سرور (تذکر: برای این کار از دستور wget در لینوکس استفاده می کنیم. در صورتی که wget روی سیستم شما نصب نشده است، با استفاده از این دستور آن را نصب کنید: sudo apt install wget)
- Extract کردن فایل دانلود شده از سرور و انتقال آنها به پوشه /tmp/final_project
- اجرای فایل reader.c
- ایجاد وقفه به مدت 60 ثانیه

فایل های C

pre_aggregation.c

این فایل یک وظیفه ساده دارد که به دلیل ماهیت متفاوت آن نسبت به فایل reader.c و اینکه لزومی به اجرای چندین باره آن وجود ندارد، در ماژول جدایی طراحی شده است. وظیفه این فایل اتصال به سرور postgres و ایجاد پایگاه داده fpdb و جدول fp_stores_data می‌باشد.

دقت کنید که برای این ماژول، اولا باید کتابخانه مربوط به postgres در زبان C را که پیش از دانلود پروژه نصب کردیم، در ابتدای برنامه include کنیم (*#include <libpq-fe.h>*)؛ ثانیا آدرس آن را در هنگام کامپایل به gcc معرفی کنیم (*gcc -o pre_aggregation.o pre_aggregation.c -I /usr/include/postgresql -lpq*). این موارد برای ماژول بعدی نیز باید رعایت شوند.

درباره نحوه عملکرد این ماژول هم تنها ذکر این نکته ضروری می‌باشد: ساز و کار مورد استفاده برای تولید پایگاه داده جدید به این صورت است که ابتدا در این ماژول تلاش می کنیم از همان ابتدای برقراری ارتباط با سرور، به fpdb متصل شویم؛ اما در صورتی که اتصال با پیغام خطا، مبنی بر عدم وجود چنین پایگاه داده ای به اتمام برسد، اتصال جدیدی را به پایگاه داده پیشفرض (با همان نام postgres) برقرار می کنیم و fpdb را می سازیم (در غیر اینصورت خطای غیرمنتظره ای رخ داده است؛ بنابراین برنامه با چاپ کردن پیغام خطا به پایان می رسد). چون در PostgreSQL برخلاف MySQLserver، اتصال به سرور مستقل از اتصال به پایگاه داده نمی‌باشد، مجبوریم این اتصال را نیز قطع کنیم و برای بار سوم به سرور متصل شویم، اما این بار مطمئن هستیم که درخواست اتصال به fpdb با موفقیت صورت می‌پذیرد. در نهایت دستور ایجاد جدول fp_stores_data را صادر می‌کنیم.

reader.c

این ماژول ابتدا ارتباطی با پایگاه داده fpdb برقرار می‌کند، سپس تمام فایل‌های متنی درون دایرکتوری /tmp/final_project را می‌خواند و هر خط این فایل‌ها را با فرمت معتبر داده‌ها تطبیق می‌دهد. در نهایت اگر این خط از فایل معتبر بود، آن را به یک دستور INSERT درون جدول fp_stores_data تبدیل می‌کند و برای سرور postgres ارسال می‌کند. پس از اجرای این پروسه برای هر کدام از فایل‌های متنی این پوشه، آن‌ها را از این مسیر پاک می‌کند و به عنوان آخرین مرحله، دو Query جهت ساختن جداول fp_store_aggregation, fp_city_aggregation برای پایگاه داده ارسال می‌کند.

برای بررسی بیشتر این ماژول، روند بالا را دوباره مرور می کنیم:

از *#include dirent.h* و تابع *opendir* برای بازکردن پوشه موردنظر استفاده می‌کنیم؛ سپس به سرور postgres متصل می‌شویم. حال شروع به خواندن تمام فایل‌های این دایرکتوری می کنیم. نکته‌ای که باید به آن توجه کنیم، این است که تابع *opendir* تمام فایل‌ها و حتی پوشه‌های آدرس ورودی‌اش را لیست می‌کند (مشابه دستور ls -a)؛ بنابراین باید مراقب باشیم تا تنها فایل‌های متنی را بازکنیم. برای این کار چک می‌کنیم که آیا تابع *fgets* قادر به خواندن خطی از این فایل هست یا خیر. درصورتی که این تابع بتواند خطی بخواند، لاگ مربوط به بازشدن فایل چاپ می‌شود.

اکنون کافی‌است تا با پردازش خطوط فایل‌ها، آنها را به کوئری‌های معتبر برای پایگاه‌داده تبدیل نماییم. برای این کار، ابتدا دو کاراکتر انتهایی این خطوط خوانده‌شده توسط *fgets* را حذف می‌کنیم (این کاراکترها به ترتیب '\r' ,'\n' هستند)؛ سپس رشته حاصله را بر اساس کاراکتر "," split می‌کنیم. توجه کنید که برخلاف زبان پایتون، در سی برای split کردن یک رشته باید تابع *strtok* را در یک حلقه صدا بزنیم تا تمام زیر رشته‌ها را بر اساس کاراکتر موردنظر پیدا کند.

طبق فرمت بیان‌شده در صورت سوال، اولین زیر رشته بدست آمده، مربوط به زمان است. این زیررشته باید 10 کاراکتری باشد. همچنین زیر رشته های دوم و سوم باید تنها شامل حروف فارسی و بقیه زیررشته ها تنها باید شامل ارقام باشند؛ برای چک‌کردن این موضوع، از regex[1] استفاده می‌کنیم. ابتدا باید کتابخانه مربوط به رجکس، یعنی *regex.h* را، به کامپایلر معرفی کنیم و مشابه زبان‌های دیگر، ابتدا یک رشته رجکس بنویسیم، تابعی را برای کامپایل‌کردن این رشته توسط موتور جست‌وجوی رجکس صدا بزنیم (در اینجا *regcomp*) و در صورتی که سینتکس رجکسی که نوشتیم معتبر بود، متن، رشته یا هر چیز دیگری را که قرار است الگوی رجکس در آن شناسایی شود را مشخص‌کنیم (در اینجا *regexec*). اما نکته بسیار مضحک درباره رجکسی که بصورت پیشفرض در زبان سی نوشته شده است، این است که سینتکس آن[2] با سینتکس تقریبا تمام زبان‌های برنامه‌نویسی دیگر متفاوت است! درهر صورت، بعد از چک‌کردن این موارد، و چک کردن اینکه تعداد زیر رشته‌هایی که با ویرگول از یکدیگر جداشده‌اند دقیقا 8 مورد باشد، زیر رشته‌ها را در قالب یک کوئری برای ارسال به پایگاه‌داده می‌فرستیم؛ اما در صورتی که یکی از شرایط فوق برقرار نباشد، لاگ مربوط به معیوب بودن آن خط از فایل را چاپ می‌کنیم، و سراغ خواندن خط بعدی فایل باز شده می‌رویم.

در نهایت پس از خواندن تمام خطوط یک فایل و ذخیره اطلاعات آن در fp_stores_data، لاگ مربوط به اتمام خواندن موفقیت‌آمیز این فایل چاپ می‌شود و با کمک تابع *system* (اجرای دستورات ترمینال در زبان سی)، آن را از پوشه /tmp/final_project پاک می‌کنیم و لاگ مربوط به پاک‌شدن آن چاپ خواهدشد.

بعد از اجرای عملیات فوق برای تمام فایل‌های این دایرکتوری، چهار کوئری برای postgres می فرستیم:

DROP TABLE IF EXISTS fp_city_aggregation

CREATE TABLE fp_city_aggregation …

و دو کوئری مشابه همین برای ایجاد جدول fp_store_aggregation . در اینجا ذکر دو نکته ضروری است:

1. مجبوریم این دو جدول را در هر بار (هر یک دقیقه) اجرای *reader.c* «بازسازی» کنیم؛ یعنی آن‌ها را پاک کرده، دوباره بسازیم. این مورد ناشی از آن است که با هر بار اضافه شدن داده‌های جدید به داده‌های قبلی fp_stores_data، محاسبات از روی این داده‌ها نیز باید دوباره انجام شود.

[1] Regular Expression

[2] این سینتکس که POSIX نام دارد، همان سینتکسی است که در سیستم عامل‌های nix* از آن استفاده شده‌است. مثلا سینکس آن، همان سینتکس رجکس در دستوری مانند *grep* در ترمینال لینوکس است؛ اما همانطور که اشاره‌شد، این سینتکس با سینتکس رجکس در زبان هایی مانند javascript, python3 متفاوت می‌باشد. البته شباهت‌های بسیار زیادی هم بین این دو سینتکس وجود دارد اما مشکلاتی مانند برخی تفاوت‌ها، نبود منابع آموزشی سریع برای آن و البته نبود مکانی برای تمرین کردن این سینتکس (مثلا سایت regex101 از آن پشتیبانی نمی‌کند ☹) باعث شده تا یادگیری و استفاده از آن، انرژی زیادی از توسعه‌دهندگان بگیرد.

2. با توجه به قبلیت‌های مهم PostgreSQL ، تمام محاسبات برای ایجاد این جداول را در قالب یک کوئری (کمی طولانی ☺) به پایگاه داده محول کردیم و از قابلیت ذخیره سازی نتیجه کوئری در قالب یک جدول جدید (*CREATE TABLE … AS*)، استفاده کردیم. برای مشاهده نام کامل ستون‌های این جداول و نحوه استخراج آن‌ها از fp_stores_data، به تابع ***aggregation*** (خط 124 کد) مراجعه فرمایید.

لازم به ذکر است که با ایجاد موفقیت‌آمیز هر جدول، لاگ مربوط به این اتفاق نیز چاپ خواهد شد.

فایل پایتون

این فایل هر 30 دقیقه یک بار اجرا می‌شود و وظیفه آن تحلیل داده‌های جمع‌آوری شده توسط برنامه سی می‌باشد. اینکه اجرای برنامه در هر 30 دقیقه یکبار رخ بدهد، توسط یک حلقه بینهایت هندل می‌شود که در این حلقه پس از چاپ شدن تحلیل‌ها، توسط تابع ***sleep*** ماژول ***time*** یک وقفه 30 دقیقه‌ای ایجاد می‌شود. برای ارتباط با PostgreSQL، از ماژول *psycopg2* استفاده کردیم که می‌بایست بوسیله *pip3* دانلود و نصب شود. شیوه ارتباط این ماژول با پایگاه داده و توابعی که برای این کار وجود دارند، مشابه توابع موجود در کتابخانه مربوط به این پایگاه‌داده در زبان سی است؛ به جز آنکه علاوه بر اتصال به پایگاه داده، باید یک مکان‌نما[3] نیز به پایگاه داده تعریف کنیم. برای اجرای برنامه پایتون، کافی‌است تا یک ترمینال جدید باز کنیم و آن را در این ترمینال اجرا کنیم.

شرح تحلیل‌های صورت گرفته

همانطور که در قسمت اصلی کد نیز مشخص است، 6 تابع در هر بار اجرای حلقه اصلی برنامه صدا زده می‌شوند. هر کدام از این توابع وظیفه انجام یک تحلیل و چاپ نتایج آن را دارند. در ادامه به شرح هر کدام از این تحلیل‌ها و فرمت خروجی آن‌ها می‌پردازیم:

1. بیشترین محصولی که در هر استان مورد استقبال مشتریان قرار گرفته‌است، کدام است؟ این محصول به چه میزان (قیمت و تعداد) به فروش رسیده است؟

برای پاسخ به این سوال، محصولی که بیشترین فروش (بر اساس تعداد) را در هر استان داشته، در خروجی چاپ می کنیم. بعد از پیدا کردن محصولی با این مشخصه در هر استان، نام استان، آی‌دی محصول، کل واحدهای به فروش رفته و میزان سرمایه حاصل از فروش را به ازای هر استان چاپ می کنیم. برای استخراج این داده‌ها از fp_stores_data استفاده می‌کنیم. این تحلیل برای هدفمندسازی سامانه توزیع و فروش در استان‌ها مفید است.

2. بهترین و بدترین فروشگاه‌های ما در سراسر کشور، از لحاظ فروش، کدام فروشگاه‌ها بودند؟

منظور از بهترین فروشگاه، فروشگاهی است که بیشترین درآمد را بدست آورده است؛ فارغ از اینکه در ابتدا چقدر جنس در انبار داشته و...، همچنین بدترین فروشگاه نیز شعبه‌ای است که کمترین فروش را داشته‌است. آی‌دی 10

[3] cursor

فروشگاهی که بیشترین درآمد و آی‌دی 10 فروشگاهی که کمترین درآمد را داشته‌اند، در خروجی به همراه میزان درآمدشان چاپ می‌شود. برای تولید این اعداد از جدول fp_store_aggregation استفاده می‌کنیم. این تحلیل برای رتبه بندی و اولویت بندی میان شعب مختلف موثر است؛ مثلا با شناسایی شعب با بیشترین درآمد، می توانیم آن‌ها را به صورت نمادین تشویق کنیم (مثلا تسهیلاتی به ایشان اعطا کنیم و این موضوع را به مسئولان دیگر شعب نیز اطلاع دهیم) تا انگیزه‌ای برای رقابت میان مدیران شعب دیگر ایجاد شود و میزان فروش شرکت افزایش یابد.

3. ساعت شلوغی شعب ما در هر شهر، چه زمانی است؟

در این تحلیل سعی می‌کنیم متوجه شویم مجموع تمام اجناس فروخته شده در هر شهر، در چه زمانی از روز بیشتر بوده‌است؛ و خروجی (نام شهر، ساعت شلوغی و کل تعداد اجناس به فروش رفته) را به تفکیک هر شهر چاپ کنیم. در این تحلیل، از جدول fp_city_aggregation استفاده می‌کنیم. با استفاده از این تحلیل، می‌توان الگوی رفتاری مردم شهرها را بررسی کرد و ساعت کار شعب آن شهر را بر اساس آن تنظیم کرد؛ مثلا تایم نهار و نماز نباید با ساعت شلوغی تداخل کند، یا اگر می‌خواهیم از تبلیغات شفاهی استفاده کنیم، بهتر است در ساعت شلوغی فروشگاه باشد.

4. 10 محصول گران‌تر در چه شهرهایی بیشتر فروختند؟ 10 محصول ارزان‌تر چطور؟

ابتدا 10 محصول گرانتر و 10 محصول ارزانتر را شناسایی می‌کنیم (ممکن است یک محصول با قیمت‌های مختلفی به فروش رسیده باشد، به همین دلیل میانگین قیمت هر محصول را در تمام ادوار ثبت اطلاعات فروش در جدول fp_stores_data ، به عنوان قیمت آن محصول درنظر می‌گیریم)؛ سپس مشخص می‌کنیم این محصول در چه شهری بیشترین/کمترین میزان فروش را داشته‌است. بنابراین؛ خروجی به صورت 20 خط از اطلاعات با فرمت مقابل است: میانگین قیمت محصول، نام شهر و مجموع تعداد محصول به فروش رفته در کلیه زمان‌ها

ار این تحلیل می‌توان به قدرت خرید و الگوی مصرف مردم شهرهای مختلف پی برد و از آن برای هدفمند سازی ارسال محصولات استفاده کرد؛ برای مثال فرض کنید ما چاره‌ای جز عرضه محصول گران‌قیمتی مانند «یخچال» نداشته‌باشیم (مثلا قرارداد خرید تعداد زیادی از این محصول را با کارخانه سازنده منعقد کرده‌ایم؛ و فسخ قرارداد هزینه بیشتری نسبت به حمل و نقل این محصول به شهرهای دورتر دارد. پس چاره‌ای جز فروش این محصول نداریم)؛ پس نیاز داریم بدانیم در اولین عرضه این محصول که احتمالا به تعداد یکسان اما کم، به تمامی شعب ارسال شده است، استقبال خریداران هر شهر چگونه بوده است. سپس با توجه به این داده‌ها، ارسال این محصول را به شهرهایی که مردم آن‌ها اقبال بیشتری به این محصول داشته‌اند، افزایش می‌دهیم (توجه کنید لزوم وجود چنین تحلیلی نسبت به تحلیل شماره 1، غیر از اینکه این تحلیل در مقیاس شهر انجام می‌شود و نسبت به مقیاس استان دقیق‌تر است، این است که معمولا اجناس گران‌تر سود بیشتری به همراه دارند مخصوصا اگر بصورت عمده از کارخانه خریداری شوند، بنابراین چنین تحلیلی برای مدیران تحت تاثیر نظام‌های سرمایه‌داری لیبرال! سودمند واقع می‌شود).

5. چه محصولی قابلیت نقدشوندگی اش بیشتر/کمتر است؟

این معیار شاید مهمترین معیار برای ارزیابی یک محصول، از دید فروشندگان است. برای پیاده سازی این معیار، محصولات را بر اساس «نسبت تعداد محصول فروخته‌شده (has_sold) به کل محصولات درون انبار (quantity + has_sold)» مرتب می‌کنیم و 10 محصول برتر و 10 محصول پایین‌تر را معرفی می‌کنیم.

6. با گذر زمان، میزان فروش محصولات چه تغییری می‌کند؟

برخی محصولات در زمان‌های مختلفی به فروش رسیده‌اند و گزارش فروش آن‌ها در fp_stores_data ثبت شده‌است. برای 10 محصولی که تعداد عرضه بیشتری داشته‌اند (در دفعات بیشتری به فروش رسیده‌اند)، تعداد محصول به فروش‌رسیده برحسب زمان نشان‌داده می‌شود. در واقع خروجی این تحلیل 10 نمودار 2 بعدی (میزان فروش – زمان) برای 10 محصول متفاوت است. بدیهی است که منظور از تعداد محصول فروخته‌شده در یک زمان، حاصل تجمیع داده‌های فروش یک محصول بر حسب زمان در تمام فروشگاه‌ها است.

این تحلیل هم به تنهایی می‌تواند در تصمیم‌گیری برای ادامه خرید و عرضه محصول توسط مدیر فروشگاه‌های زنجیره‌ای اثرگذار باشد، هم برای مشاهده و ارزیابی نتایج اقداماتی که در اثر تحلیل‌های قبلی داشته‌ایم مفید واقع می‌شود.

با تشکر از توجه شما

پایان