به نام خدا

غزاله کریمی ۴۰۲۱۰۶۳۷۲
۴۰۳۱۰۰۴۰۵ مهشید بارگاهی

---

# ۱ بارگذاری داده‌ها (ETL) و پر کردن جداول پایگاه‌داده

در این بخش از پروژه، داده‌ها از جداولstaging (مانند bdbkala_full، products_properties و branch_product_suppliers) به جداول اصلی پایگاه‌داده منتقل شدند. هدف، تبدیل دیتاست خام به ساختار نرمال‌شده و سازگار با قیود (Constraints) و قواعد تجاری بود. همچنین برای جلوگیری از توقف فرآیند در صورت وجود داده‌های نامعتبر، مکانیزم ثبت خطا در سطح ETL پیاده‌سازی شد.

## ۱.۱ زیرساخت ثبت خطا (ETL Error Logging)

ابتدا جدول etl_error_log برای ذخیره خطاهای رخ‌داده در فرآیند بارگذاری ساخته شد. این جدول شامل اطلاعاتی مانند نام جدول مقصد، نوع عملیات، نام قید (در صورت وجود)، SQLSTATE، پیام خطا و داده‌ی همان رکورد به صورت JSONB است. سپس تابع log_etl_error() ایجاد شد تا در تمام مراحل بارگذاری، هر خطا به صورت یکپارچه ثبت شود و عملیات ادامه پیدا کند.

## ۲.۱ پر کردن جداول پایه با run_staging_load()

یک روال (Procedure) جهت پر کردن جداول پایه پیاده‌سازی شد تا وابستگی‌های کلیدی و ارجاعی (FK) رعایت شوند. ترتیب اجرای این مرحله به صورت زیر بود:

- **Category**: ابتدا دسته‌های ریشه از products_properties وارد شدند. سپس زیردسته‌ها به صورت self-referencing با فیلد parent_category_id اضافه شدند.
- **Branch_Manager**: مدیران شعب از branch_product_suppliers استخراج و در جدول branch_manager ذخیره شدند.
- **Address**: سه نوع آدرس وارد شد:
  ۱. آدرس‌های ارسال سفارش از bdbkala_full (همراه با city/region/zip)
  ۲. آدرس شعب از branch_product_suppliers (با مقادیر NULL برای شهر/منطقه/کدپستی)
  ۳. آدرس تامین‌کنندگان از branch_product_suppliers (با مقادیر NULL برای شهر/منطقه/کدپستی)
- **Branch**: شعب با اتصال به address و branch_manager ایجاد شدند تا FKها برقرار باشند.
- **Supplier**: تامین‌کنندگان وارد شدند و در صورت وجود آدرس، address_id نیز به رکورد متصل شد.

در تمامی مراحل فوق، در صورت رخ‌داد هرگونه خطا یا نقض constraint، اطلاعات رکورد و خطا در etl_error_log ثبت شد.

### ۳.۱ پر کردن جدول Customer با اعتبارسنجی

برای مشتری‌ها، علاوه بر درج داده‌ها، عملیات اعتبارسنجی انجام شد:

- تابع normalize_phone_us_e164 برای نرمال‌سازی شماره تلفن پیاده‌سازی شد.
- رکوردهای نامعتبر (ایمیل نامعتبر، تلفن نامعتبر، نام خالی) در جدول customer_reject_log ذخیره شدند.
- تنها رکوردهای معتبر وارد جدول customer شدند.
- فیلدهای customer_type، customer_status و vat_exemption_percent بر اساس customer_segment و income تولید شدند.

### ۴.۱ پر کردن جدول Product

در این مرحله، داده‌های محصولات از products_properties وارد جدول product شدند:

- specifications از فیلد attributes به JSONB تبدیل شد.
- category_id محصول از طریق category و زیردسته‌ی متناظر تعیین شد.
- vat_exemption_percent به صورت deterministic (بر اساس hashtext(product_name)) تولید شد تا برای هر محصول ثابت باشد.

### ۵.۱ پر کردن جدول Branch_Product

این جدول رابطه‌ی بین شعب و محصولات را نگهداری می‌کند:

- ارتباط محصول و شعبه از branch_product_suppliers استخراج شد.
- sale_price با استفاده از میانگین unit_price در bdbkala_full محاسبه شد و در صورت نبود، از فرمول min_supply_price * 1.30 استفاده شد.
- discount از میانگین تخفیف‌ها استخراج شد (در غیر این صورت ۰).

### ۶.۱ پر کردن جدول Supply

پس از آماده شدن Supplier و Branch_Product:

- برای هر (supplier, branch_product) رکورد supply ایجاد شد.
- supply_time از lead_time_days و cost_price از supply_price برداشت شد.

### ۷.۱ پر کردن جدول Wallet

موجودی کیف پول از جدول wallet_balances به جدول wallet منتقل شد. برای جلوگیری از درج تکراری، از ON CONFLICT DO UPDATE استفاده شد تا در صورت وجود رکورد قبلی، مقدار balance به‌روزرسانی گردد.

### ۸.۱ پر کردن جدول Ordere با داده‌های تاریخی

به دلیل وجود تریگرهایی که درج داده‌ی تاریخی را محدود می‌کردند (مانند الزام order_date = CURRENT_DATE)، تریگرهای مربوطه موقتاً غیرفعال شدند. سپس:

- customer_id با اولویت email > phone > name تعیین شد.
- تاریخ order_date با پشتیبانی از چند قالب (مانند MM.DD.YYYY و YYYY-MM-DD) تبدیل شد.
- مقادیر enum مانند priority/status/payment_method به مقادیر مجاز نگاشت شدند.

در نهایت تریگرها مجدداً فعال شدند.

### ۹.۱ پر کردن جدول Order_Item با داده‌های تاریخی

برای order_item نیز به دلیل تریگرهای قیمت‌گذاری، عملیات درج تاریخی با غیرفعال‌سازی موقت تریگرهای BEFORE INSERT انجام شد. سپس:

- مقدار quantity قبل از تبدیل نوع (cast) اعتبارسنجی شد تا مواردی مثل 1- یا 1.0- باعث توقف فرآیند نشوند.
- اتصال به branch_product_id با استفاده از staging branch_product_suppliers انجام شد تا انتخاب شعبه دقیق باشد.
- خطاهای هر رکورد در etl_error_log ثبت شدند و فرآیند ادامه یافت.

### ۱۰.۱ پر کردن جدول Feedback

بازخوردها با اتصال reviews به اطلاعات محصول و سپس branch_product تولید شدند. قیود مربوط به امتیاز (۱ تا ۵) و طول کامنت بررسی شد و در صورت نقض، رکورد در etl_error_log ذخیره شد.

### ۱۱.۱ پر کردن جدول Shipment

برای shipment، تاریخ‌ها دارای قالب‌های مختلف بودند؛ بنابراین تاریخ‌ها با regex تشخیص و با to_date تبدیل شدند. همچنین فیلد packaging تحلیل شد تا نوع/سایز/جنس بسته‌بندی استخراج گردد. هرگونه نقض قید (مانند قوانین حمل‌ونقل و بسته‌بندی) در etl_error_log ثبت شد.

### ۱۲.۱ جمع‌بندی

در این بخش، بارگذاری داده‌ها از staging به جداول اصلی با رعایت ترتیب وابستگی‌ها انجام شد. داده‌ها پاکسازی و استاندارد شدند و برای داده‌های تاریخی، تریگرهای محدودکننده به صورت موقت غیرفعال شدند. همچنین با ثبت خطاها در etl_error_log، فرآیند ETL بدون توقف اجرا شد و رکوردهای مشکل‌دار برای بررسی و اصلاح بعدی نگهداری شدند.

## ۲ تولید داده‌های مصنوعی (Synthetic Data Generation)

در این بخش، به منظور تست سناریوهای واقعی‌تر سیستم (وفاداری مشتریان، خرید با BNPL، تراکنش‌های کیف‌پول و فرآیند مرجوعی)، مجموعه‌ای از داده‌های مصنوعی روی دیتابیس تولید شد. هدف این بود که بدون تغییر ساختار داده‌ها یا قوانین تجاری، رکوردهای جدیدی ایجاد شوند که برای تحلیل، گزارش‌گیری و ارزیابی عملکرد تریگرها و قیود مناسب باشند.

## ۱.۲ تعیین محدوده داده‌های جدید

ابتدا بیشترین order_id موجود در جدول ordere استخراج شد تا محدوده سفارش‌های جدید مشخص شود:

- SELECT MAX(order_id) FROM ordere;
- خروجی: 25260

در نتیجه، تمامی سفارش‌های تولیدشده در این مرحله دارای order_id بزرگ‌تر از مقدار فوق هستند و در زمان خروجی گرفتن CSV نیز از همین مرز برای فیلتر استفاده شد.

## ۲.۲ محاسبه وفاداری (Loyalty) مشتریان

برای محاسبه امتیاز وفاداری، خریدهای سه ماه اخیر مشتریان در نظر گرفته شد:

- مبلغ خرید سه ماه اخیر هر مشتری: SUM(quantity * final_price_at_order_time)
- تبدیل به امتیاز وفاداری: FLOOR(total/100)

سپس مشتریانی که points_3m > 0 داشتند به عنوان مشتریان واجد شرایط Eligible) (Customers انتخاب شدند.

## ۳.۲ افزایش امتیاز وفاداری با سفارش‌های تصادفی

برای افزایش مصنوعی امتیاز وفاداری برخی مشتریان، ابتدا یک جدول موقت target_boost ساخته شد که شامل ۴۰۰ مشتری تصادفی بود. این مشتری‌ها طوری انتخاب شدند که سابقه BNPL نداشته باشند تا سناریوی BNPL بعدی با کمترین تداخل انجام شود.

### ۱.۳.۲ ایجاد سفارش‌های جدید در ۴۵ روز اخیر

برای هر مشتری در target_boost، سه سفارش جدید با ویژگی‌های زیر ایجاد شد:

- status = 'Received'
- priority = 'High'
- payment_method = 'In-App Wallet'
- order_date به صورت تصادفی در بازه ۴۵ روز اخیر

از آنجا که تریگر trg_enforce_order_date درج سفارش با تاریخ غیر از امروز را محدود می‌کرد، این تریگر به صورت موقت غیرفعال و پس از درج سفارش‌ها مجدداً فعال شد.

### ۲.۳.۲ ایجاد آیتم‌های سفارش Order) (Items

پس از ایجاد سفارش‌ها، برای هر سفارش تعدادی آیتم سفارش ساخته شد:

- انتخاب محصول از میان ۲۰ محصول گران‌تر (برای افزایش مبلغ خرید و امتیاز وفاداری)
- تعداد هر آیتم به صورت تصادفی در بازه ۳ تا ۵
- return_status = NULL

## ۴.۲ ایجاد سفارش‌های BNPL برای مشتریان واجد شرایط

پس از افزایش فعالیت مشتریان، برای مشتریانی که امتیاز وفاداری آنها مثبت بود (points_3m > 0) سفارش‌هایی با روش پرداخت BNPL ایجاد شد:

- payment_method = 'BNPL'
- order_date = CURRENT_DATE
- status = 'Received'
- priority = 'High'

### ۱.۴.۲ ایجاد آیتم سفارش برای سفارش‌های BNPL

برای هر سفارش BNPL، یک آیتم سفارش ایجاد شد:

- quantity = 1
- انتخاب محصول ارزان‌تر (برای کاهش ریسک بدهی BNPL)

## ۵.۲ ایجاد Plan BNPL برای سفارش‌های BNPL

برای تمامی سفارش‌هایی که روش پرداخت آنها BNPL و تاریخ آنها امروز بود، یک رکورد در جدول bnpl_plan ایجاد شد و وضعیت آن Active تنظیم گردید.

## ۶.۲ شبیه‌سازی بازپرداخت اقساطی (Repayment) از طریق کیف‌پول

برای طبیعی‌تر شدن داده‌های BNPL، فرآیند بازپرداخت اقساط به صورت زیر شبیه‌سازی شد:

۱. محاسبه مبلغ کل بدهی هر BNPL از روی order_item

۲. تعیین تصادفی تعداد اقساط پرداخت‌شده (paid_installments بین ۰ تا ۳)

۳. تولید اقساط با generate_series(1,3) تا سقف تعداد پرداخت‌شده

۴. ایجاد wallet_transaction از نوع Payment برای هر قسط

۵. ایجاد رکوردهای repayment با اتصال به wallet_id و transaction_sequence_number (طبق constrain-های موجود)

این روند باعث شد داده‌های پرداخت BNPL هم با قیود بانک اطلاعاتی سازگار باشد و هم رفتار واقعی‌تری را شبیه‌سازی کند.

## ۷.۲ ایجاد تراکنش‌های پرداخت برای سفارش‌های Wallet

برای سفارش‌هایی که روش پرداخت آنها In-App Wallet بود، تراکنش پرداخت ایجاد شد:

- مبلغ تراکنش برابر مجموع مبلغ آیتم‌های سفارش
- زمان تراکنش نزدیک به تاریخ سفارش (۱ یا ۲ روز قبل به صورت تصادفی)

### ۸.۲ ایجاد تراکنش برداشت (Withdrawal)

برای شبیه‌سازی برداشت از کیف پول:

- کیف‌پول‌هایی با balance $> 50$ انتخاب شدند.
- حدود ۳۰٪ از آنها به صورت تصادفی انتخاب شدند.
- برای آنها تراکنش Withdrawal با مبلغی بین ۱۰٪ تا ۴۰٪ موجودی ایجاد شد.

### ۹.۲ ایجاد تراکنش واریز (Deposit) برای سازگاری با موجودی نهایی

از آنجا که wallet.balance موجودی نهایی را نگه می‌دارد، لازم بود ledger تراکنش‌ها با آن سازگار شود. بنابراین:

- مجموع پرداخت‌ها و برداشت‌ها به عنوان total_out محاسبه شد.
- اگر balance $>$ total_out بود، یک تراکنش Deposit به اندازه (balance - total_out) ایجاد شد.

### ۱۰.۲ تولید درخواست‌های مرجوعی (Return Request)

برای شبیه‌سازی فرآیند مرجوعی:

- ۵۰۰ آیتم سفارش تصادفی انتخاب شد که سفارش آنها حتماً Received باشد.
- دلیل مرجوعی به صورت تصادفی از Damaged / Wrong Item / Late Delivery
- نتیجه بررسی به صورت تصادفی از Return Pending Review / Return Approved / Return Rejected
- request_date چند روز بعد از order_date
- decision_date با احتمال ۷۰٪ مقداردهی شد و در غیر این صورت NULL باقی ماند.

### ۱۱.۲ خروجی CSV از داده‌های تولیدشده

در پایان، داده‌های تولیدشده برای ارائه و تحلیل به صورت CSV خروجی گرفته شدند. خروجی‌گیری با ابزار گرافیکی انجام شد و فیلترهایی مانند order_id $> 25260$ برای انتخاب فقط رکوردهای جدید در نظر گرفته شد.

### ۱۲.۲ جمع‌بندی

در مجموع، این بخش با تولید داده‌های مصنوعی کنترل‌شده، سناریوهای کلیدی سیستم را پوشش داد: افزایش وفاداری مشتریان، ایجاد سفارش‌های BNPL و بازپرداخت اقساط، ساخت تراکنش‌های کیف‌پول و همچنین شبیه‌سازی فرآیند مرجوعی. تمام داده‌ها به گونه‌ای تولید شدند که با قیود و روابط پایگاه‌داده سازگار بوده و برای تحلیل و تست قابل استفاده باشند.