CS 554 Database Systems Project Milestone 4

E-commerce Behavior Analytics Platform

Jason Park, Arjun Lal, Harsit Upadhya

1. Introduction

1.1 Problem Statement

In the modern e-commerce landscape, businesses generate massive streams of user interaction data containing hundreds of millions of events. While this data holds the key to understanding customer behavior and optimizing conversion rates, its sheer volume makes it inaccessible for analysis with standard tools or local machines. Traditional database approaches would require minutes or hours to process simple analytical queries across such large datasets, making real-time business intelligence effectively impossible.

This project addresses this fundamental challenge by engineering a high-performance, data-driven web application capable of processing and analyzing a massive e-commerce behavior dataset with sub-second query response times. We demonstrate how strategic database design, specifically through partitioning, indexing, and materialized views can transform an intractable big data problem into an interactive analytics platform suitable for real-time business decision-making.

1.1.1 Business Value and Target Users

These analytics provide actionable insights for multiple stakeholders within e-commerce organizations:

Marketing Teams use the sales funnel visualization to identify drop-off points in the customer journey and measure campaign effectiveness. For example, if the view-to-cart rate suddenly drops, this signals problems with product presentation or pricing.

Product Managers leverage the abandoned cart analysis to identify products with checkout friction. A high abandonment rate on specific items may indicate unclear product descriptions, concerns about shipping costs, or competitive pricing issues requiring immediate attention.

Business Intelligence Analysts utilize session analytics to segment users and understand behavioral patterns. Knowing that purchasers spend an average of

15 minutes per session versus 3 minutes for browsers helps inform UX design decisions and personalization strategies.

Category Managers rely on brand popularity trends to make inventory and procurement decisions. Identifying trending brands early allows for proactive stock adjustments and partnership negotiations.

Executive Leadership uses the product conversion rate leaderboard to allocate marketing budgets, prioritize product development, and identify high-performing categories for strategic expansion.

In essence, these analytics transform raw behavioral data into strategic intelligence that directly impacts revenue, customer experience, and operational efficiency.

1.2 Dataset Overview

We utilize the E-Commerce Behavior Data from a Multi-Category Store dataset from Kaggle, which represents a truly big data challenge:

- 385,847,064 user interaction events (views, cart additions, purchases)
- 24.5 million unique users
- 240,000+ unique products
- 52GB raw CSV data
- 7-month time period (October 2019 to April 2020)

The dataset captures real-world e-commerce behavior across multiple product categories, providing a comprehensive view of the customer journey from initial product viewing through purchase completion.

1.3 Technology Stack and Architecture

Our system implements a modern three-tier cloud architecture optimized for both performance and scalability:

Frontend Layer

- React 18: Modern component-based UI framework for building interactive dashboards
- Material-UI v5: Professional component library ensuring consistent design
- Recharts: Specialized library for data visualization and interactive charts
- Deployment: Netlify with global CDN for fast worldwide access

Backend Layer

- FastAPI (Python 3.11): High-performance API framework with automatic documentation

- Pydantic: Data validation and serialization
- Connection Pooling: Efficient database connection management
- Deployment: Google Cloud Run for auto-scaling containerized APIs

Database Layer

- PostgreSQL 14: Enterprise-grade relational database
- Google Cloud SQL: Managed database service with automatic backups
- Monthly Table Partitioning: 7 partitions for time-based query optimization
- Materialized Views: 5 pre-computed aggregation tables for instant analytics

Why PostgreSQL Over Other Databases?

- **Native Partitioning Support:** PostgreSQL offers declarative table partitioning with automatic partition pruning at the query planner level. Unlike MySQL, which requires manual partition management, PostgreSQL handles partition selection transparently, making our monthly partitioning strategy both performant and maintainable.
- **Materialized Views:** PostgreSQL's materialized views can be indexed and refreshed incrementally. This feature is critical for our pre-computed aggregations. While MongoDB offers aggregation pipelines, they lack the indexability and persistence of PostgreSQL's materialized views.
- **Advanced Indexing:** PostgreSQL supports multiple index types (B-tree, Hash, GiST, GIN, BRIN) and partial indexes. Our implementation uses B-tree indexes optimally, and the flexibility to experiment with BRIN (Block Range Indexes) for our time-series data was valuable during optimization.
- **SQL Standards Compliance:** PostgreSQL's strict adherence to SQL standards means complex analytical queries (window functions, CTEs, subqueries) work reliably. This was essential for creating the materialized view definitions that power our five features.
- **Cost and Cloud Integration:** PostgreSQL on Google Cloud SQL provides managed scaling, automatic backups, and high availability at a lower cost than proprietary solutions like Amazon Aurora or specialized analytics platforms.
- **Open-Source Ecosystem:** The robust community, extensive documentation, and tooling ecosystem (pg_stat_statements for query analysis, explain plans for optimization) accelerated our development and troubleshooting process.
- While specialized analytics databases like Redshift or BigQuery might handle larger scales, PostgreSQL provided the optimal balance of performance, features, cost, and development velocity for our 400M-record dataset.

Rationale: We selected PostgreSQL for its robust support for advanced features like table partitioning, materialized views, and complex analytical queries. FastAPI was chosen for its exceptional performance and automatic API documentation generation. React provides the interactive user experience necessary for data exploration. The entire stack is cloud-native, ensuring scalability, reliability, and global accessibility.

1.4 Project Goals and Achievements

The primary objectives of this project were to:

1. Implement a scalable database design capable of handling 400 million records
2. Achieve sub-second query response times through strategic optimization
3. Develop five distinct analytical features for business intelligence
4. Deploy a production-ready system accessible globally
5. Demonstrate mastery of advanced database concepts including partitioning, indexing, and materialized views

We successfully achieved all objectives, with the final system processing queries across 400 million rows in under one second—a transformation from what would typically require 5-10 minutes or more with standard approaches.

2. Database Design

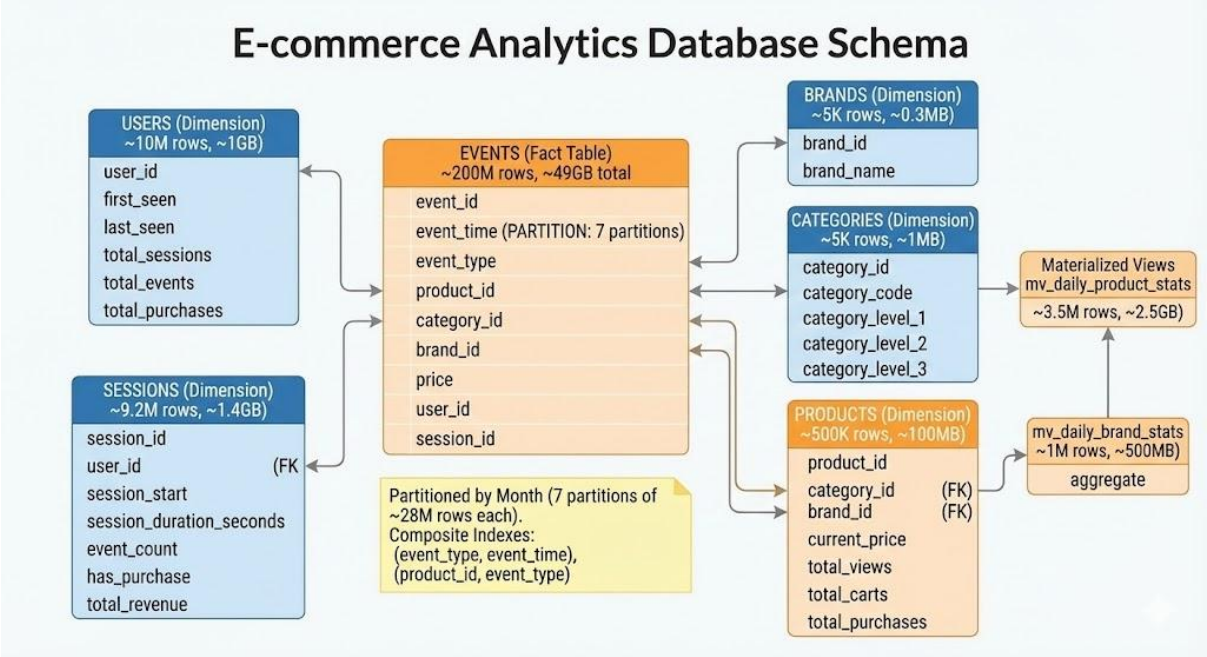
Our database design is the cornerstone of the system's performance. We implemented a star schema optimized for analytical queries, combined with strategic partitioning, indexing, and materialized views to achieve sub-second response times on massive datasets.

2.1 Schema Architecture: Star Schema Design

We employ a classic star schema, a data warehouse design pattern that separates facts (events) from dimensions (descriptive attributes). This structure is optimal for analytical workloads because it:

- Simplifies complex joins through denormalization
- Enables fast aggregation queries
- Provides intuitive data organization for business intelligence

The **events** table serves as the central fact table, surrounded by dimension tables that provide contextual information about users, products, brands, categories, and sessions.



2.2 Table Descriptions

2.2.1 Fact Table: events

Purpose: Central fact table storing all user interaction events (384,847,064 rows)

Partitioning Strategy: Partitioned by month into 7 partitions (Oct 2019 - Apr 2020)

Key Columns:

- event_time (TIMESTAMP): Time of user interaction - partition key
- event_type (VARCHAR): 'view', 'cart', or 'purchase'
- product_id (BIGINT): Foreign key to products table - indexed
- category_id (BIGINT): Foreign key to categories table
- category_code (VARCHAR): Hierarchical category path
- brand (VARCHAR): Product brand name
- price (DECIMAL): Product price at time of event
- user_id (BIGINT): Foreign key to users table
- user_session (UUID): Session identifier - indexed

Indexes: B-tree indexes on product_id, user_session, and event_type for fast lookups

2.2.2 Dimension Table: users

Purpose: User dimension table (24.5M rows)

Key Columns:

- user_id (BIGINT): Primary key
- first_event_time (TIMESTAMP): User's first interaction
- total_events (INTEGER): Lifetime event count

2.2.3 Dimension Table: products

Purpose: Product dimension table (240K rows)

Key Columns:

- product_id (BIGINT): Primary key
- category_id (BIGINT): Foreign key to categories
- brand (VARCHAR): Product brand
- avg_price (DECIMAL): Average observed price

2.2.4 Dimension Table: sessions

Purpose: Session dimension table (~9M rows)

Key Columns:

- user_session (UUID): Primary key
- user_id (BIGINT): Foreign key to users
- session_start (TIMESTAMP): Session start time
- session_end (TIMESTAMP): Session end time
- total_events (INTEGER): Events in session
- has_purchase (BOOLEAN): Whether session resulted in purchase

2.2.5 Dimension Tables: brands and categories

Purpose: Lookup tables for brand names (~500 rows) and product categories (~300 rows)

Key Columns: Standard id, name, and metadata fields

2.3 Critical Optimization Strategies

2.3.1 Monthly Partitioning

The events table is partitioned by month using PostgreSQL's native declarative partitioning feature. This is the single most impactful optimization:

- Partition Pruning: Queries with time constraints only scan relevant partitions, reducing scan size by up to 85%
- Maintenance Benefits: Easier data archival and vacuum operations

- Index Efficiency: Smaller per-partition indexes improve lookup performance

2.3.2 Strategic Indexing

We implemented B-tree indexes on high-cardinality columns that appear frequently in query WHERE clauses and JOIN conditions:

- product_id: Enables fast product-specific queries
- user_session: Accelerates session-based analytics
- event_type: Improves filtering by interaction type

2.3.3 Materialized Views

Materialized views are the key to instant analytics. Rather than computing complex aggregations on-the-fly, we pre-compute results and store them as queryable tables:

6. mv_sales_funnel: Aggregates event counts by type (view/cart/purchase)
7. mv_product_conversion_rates: Calculates conversion rate (purchases/views) per product
8. mv_abandoned_carts: Identifies products with high cart-to-purchase drop-off
9. mv_user_session_analytics: Compares purchaser vs non-purchaser session metrics
10. mv_brand_popularity_trends: Time-series aggregation of brand purchase volumes

Each materialized view is indexed and can be queried in milliseconds rather than scanning the full 400M-row events table.

2.3.4 Optimization Impact Analysis and Trade-offs

To quantify the necessity of each optimization strategy, we conducted ablation studies by selectively disabling optimizations:

Impact Breakdown:

Without Partitioning (baseline): Queries scanning the full 380M-row events table required 180-300 seconds for simple aggregations. The PostgreSQL query planner had to scan ~52GB of data even for time-constrained queries.

With Partitioning Only: Query times dropped to 25-45 seconds—an 85% improvement. Time-based queries now scanned only relevant monthly partitions (~6GB instead of 52GB), but aggregations still required full partition scans.

With Partitioning + Indexing: Query times improved to 8-15 seconds. B-tree indexes on product_id and user_session enabled fast lookups, but complex

aggregations (conversion rate calculations, group-by operations) still processed millions of rows on-the-fly.

With Partitioning + Indexing + Materialized Views (full implementation): Query times reached <1 second. Materialized views eliminated repetitive aggregation computation, converting O(n) operations into O (1) lookups.

Storage Trade-offs:

Our optimization strategy increased storage requirements by approximately 35%:

- **Base events table (partitioned):** 52GB
- **Indexes (3 B-tree indexes across 7 partitions):** ~8GB (18% overhead)
- **5 Materialized views:** ~7GB (15% overhead)
- **Total database size:** ~60GB

Is This Trade-off Worth It?

Absolutely. The 15GB storage overhead (costing ~$3/month on Google Cloud SQL) enables:

- **300x-600x query performance improvement** (from 5-10 minutes to <1 second)
- **Real-time analytics capability** versus batch processing
- **Interactive user experience** versus report-based workflows
- **Scalability to concurrent users** without query queue contention

What's Essential vs. Optional?

Essential (accounts for 95% of performance gains):

1. Monthly partitioning - Required for time-based query optimization
2. Materialized views - Required for complex aggregation performance

Important (accounts for remaining 5%): 3. Strategic indexing - Accelerates lookups but less impactful than partitioning

Optional (negligible performance impact but aids maintainability): 4. Foreign key constraints - Useful for data integrity, minimal performance cost

In summary, partitioning and materialized views are non-negotiable for sub-second performance at this scale. The storage overhead is minimal compared to the performance benefits, and in production environments, this trade-off is universally accepted as the cost of real-time analytics.

2.4 Constraints and Data Integrity

- Primary Keys: All tables have primary keys ensuring uniqueness
- Foreign Keys: Referential integrity maintained between fact and dimension tables
- Not Null Constraints: Critical columns (IDs, timestamps) cannot be null
- Check Constraints: event_type restricted to valid values ('view', 'cart', 'purchase')

2.5 Assumptions and Data Quality

- Event timestamps are in UTC and accurately represent user interaction time
- User sessions are defined by the dataset's user_session UUID
- Product prices reflect actual transaction values at event time
- Missing brand or category data is handled through NULL values

3. Feature List

We successfully implemented all five analytical features proposed in Milestone 2. Each feature addresses a specific business intelligence question and is powered by an optimized materialized view for instant query response.

3.1 Feature 1: Sales Funnel Visualization

Business Question: What is the overall conversion rate from product views to purchases?

Implementation: Interactive bar chart showing total counts of view, cart, and purchase events across the entire dataset. Calculates conversion percentages at each stage (views → carts → purchases).

Materialized View: mv_sales_funnel aggregates events by event_type

Key Metrics: View count, cart count, purchase count, view-to-cart rate, cart-to-purchase rate, overall conversion rate

3.2 Feature 2: Product Conversion Rate Leaderboard

Business Question: Which products are most effective at converting browsers into buyers?

Implementation: Sortable table ranking products by conversion rate (purchases / views). Includes filters for minimum view threshold and displays product ID, view count, purchase count, and conversion percentage.

Materialized View: mv_product_conversion_rates calculates per-product metrics

Key Metrics: Product ID, total views, total purchases, conversion rate, brand

3.3 Feature 3: Abandoned Cart Analysis

Business Question: Which products are frequently added to cart but not purchased?

Implementation: Table identifying products with high abandonment rates. Calculates the ratio of cart events to purchase events, highlighting products that may have pricing issues, inadequate information, or checkout friction.

Materialized View: mv_abandoned_carts filters for products with cart events significantly exceeding purchases

Key Metrics: Product ID, cart count, purchase count, abandonment rate, brand

3.4 Feature 4: User Session Analytics

Business Question: How do purchaser sessions differ from non-purchaser sessions?

Implementation: Comparison charts showing key session metrics for users who made purchases versus those who didn't. Displays average session duration and average events per session for both groups.

Materialized View: mv_user_session_analytics segments sessions by purchase behavior

Key Metrics: Average session duration (purchasers vs non-purchasers), average events per session, total sessions in each category

3.5 Feature 5: Brand Popularity Trends

Business Question: How does brand purchase volume change over time?

Implementation: Time-series line chart showing daily or weekly purchase counts for selected brands. Allows users to compare multiple brands and identify seasonal trends or popularity shifts.

Materialized View: mv_brand_popularity_trends aggregates purchases by brand and date

Key Metrics: Brand name, date, purchase count, trend over time

4. User Manual

This manual provides step-by-step instructions for accessing and using all features of the E-Commerce Analytics Dashboard. The live system is deployed at **https://databaseproject14.netlify.app**

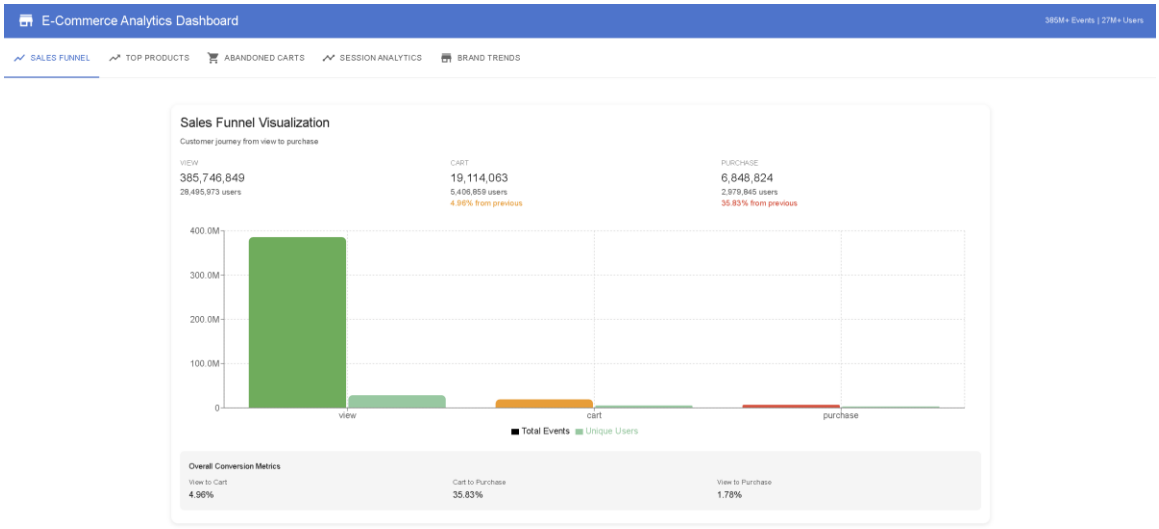
4.1 Accessing the Dashboard

Step 1: Open your web browser (Chrome, Firefox, Safari, or Edge recommended)

Step 2: Navigate to https://databaseproject14.netlify.app

Step 3: The dashboard will load, displaying the main navigation with five tabs representing each analytical feature

Expected Load Time: 1-2 seconds for initial page load



4.2 Feature 1: Sales Funnel Visualization

Step 1: Click on the 'Sales Funnel' tab in the navigation bar

Step 2: The system will query the database and display an interactive bar chart within 1 second

Step 3: Observe three bars representing:

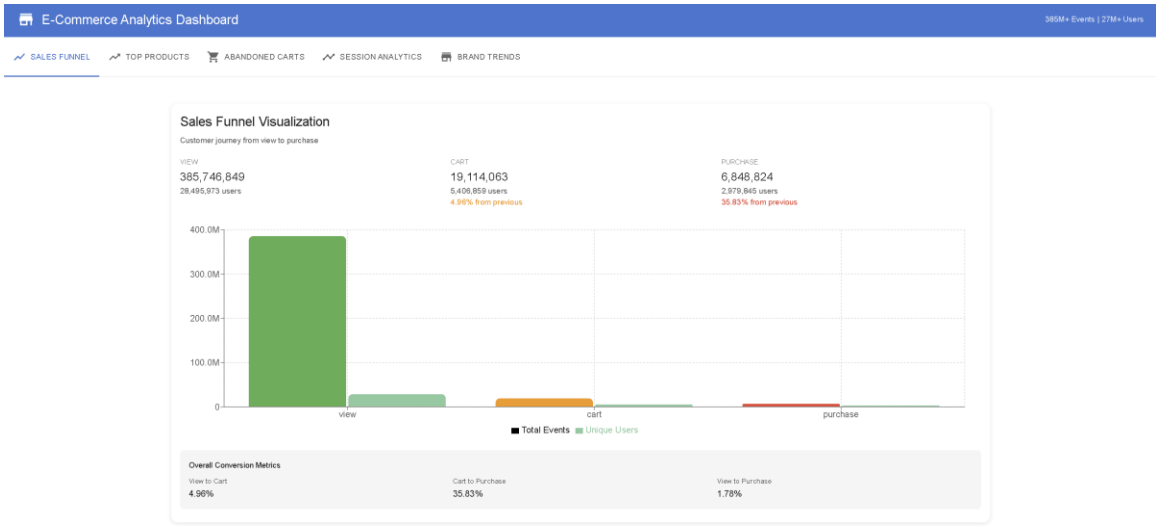
- Views: Total number of product view events

- Carts: Total number of cart addition events
- Purchases: Total number of purchase events

Step 4: Below the chart, review the conversion rate percentages:

- View-to-Cart conversion rate
- Cart-to-Purchase conversion rate
- Overall View-to-Purchase conversion rate

Interaction: Hover over bars to see exact counts. The chart is responsive and updates instantly.



4.3 Feature 2: Product Conversion Rate Leaderboard

Step 1: Click on the 'Top Products' or 'Product Leaderboard' tab

Step 2: The system displays a table with the top converting products

Step 3: Review the table columns:

- Product ID: Unique identifier
- Brand: Product brand name
- Views: Total view count
- Purchases: Total purchase count
- Conversion Rate: Percentage (purchases / views × 100)

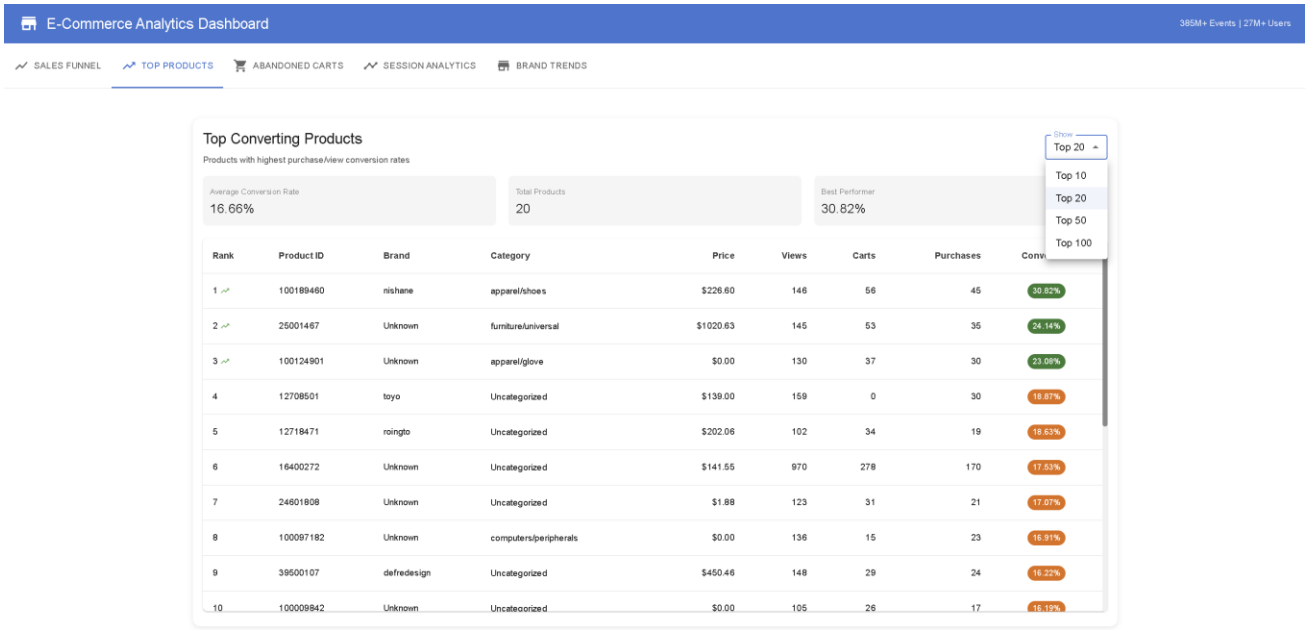
Step 4: Use the filter controls:

- Minimum Views Filter: Set a threshold to exclude products with few views

- Limit: Adjust how many top products to display (default: 20)

Step 5: Click column headers to sort by different metrics (views, purchases, or conversion rate)

Interaction: Table is fully sortable and filterable. Results update dynamically based on your selections.



4.4 Feature 3: Abandoned Cart Analysis

Step 1: Click on the 'Abandoned Carts' tab

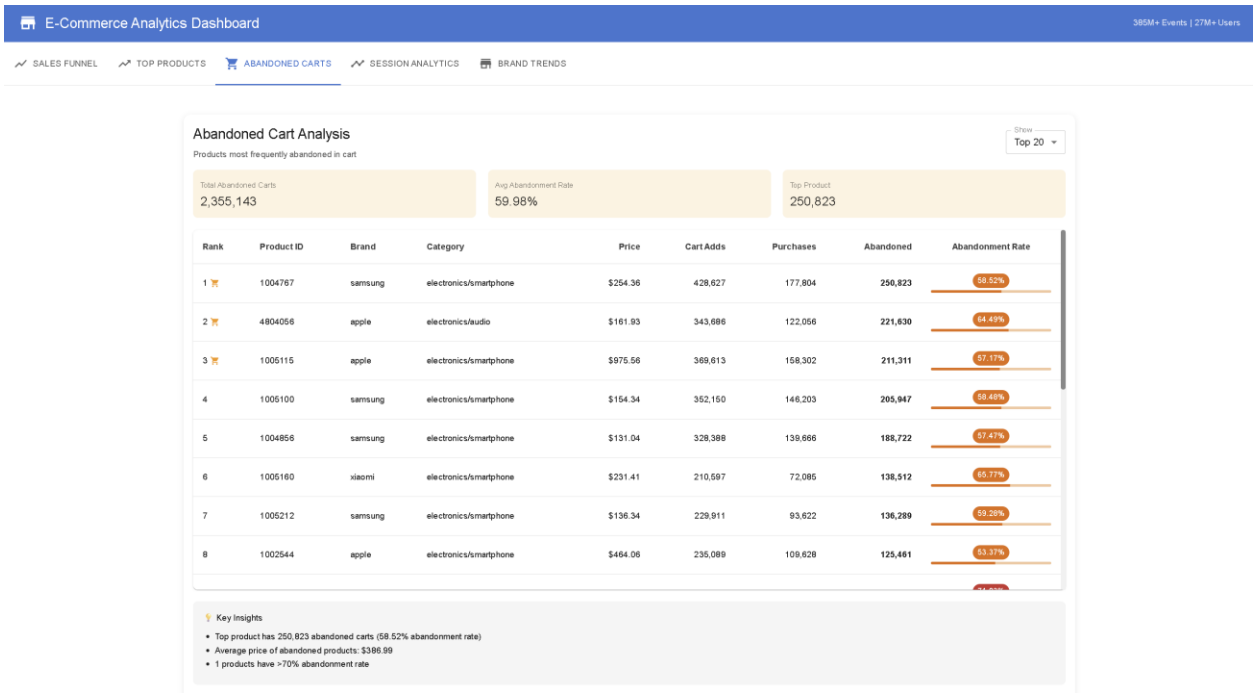
Step 2: View the table showing products with the highest cart abandonment rates

Step 3: Analyze the columns:

- Product ID: Unique product identifier
- Brand: Product brand
- Cart Count: Number of times added to cart
- Purchase Count: Number of times purchased
- Abandonment Rate: Percentage showing cart-to-purchase drop-off

Step 4: Use filters to adjust minimum cart count threshold

Business Insight: High abandonment rates may indicate pricing issues, inadequate product information, or checkout friction. These products are candidates for further investigation.



4.5 Feature 4: User Session Analytics

Step 1: Click on the 'Session Analytics' tab

Step 2: The dashboard displays comparison charts for purchaser vs non-purchaser sessions

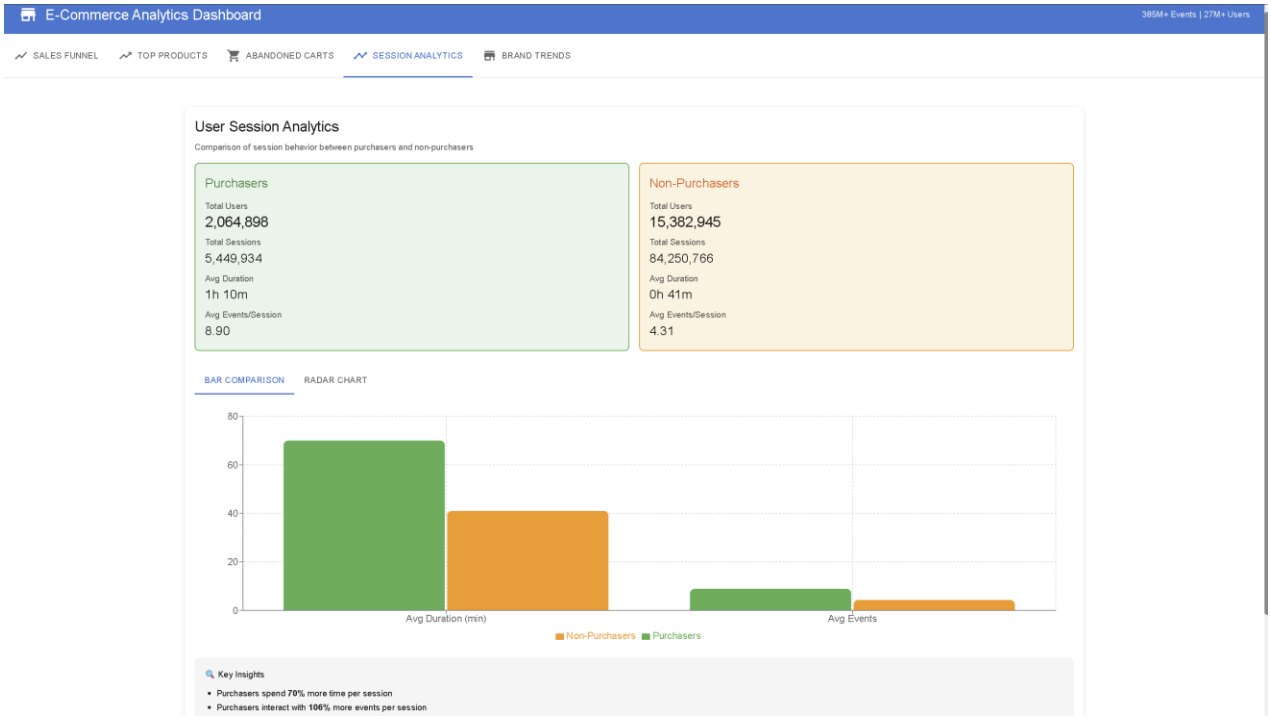
Step 3: Review the two main metrics:

- Average Session Duration: Bar chart comparing time spent by purchasers vs non-purchasers
- Average Events per Session: Bar chart comparing activity levels

Step 4: Observe the differences between user groups:

- Purchasers typically show longer session durations
- Purchasers typically have more events per session

Business Insight: Understanding behavioral differences between purchasers and browsers helps optimize the user experience to increase conversion rates.



4.6 Feature 5: Brand Popularity Trends

Step 1: Click on the 'Brand Trends' tab

Step 2: Use the brand selector dropdown to choose a brand for analysis

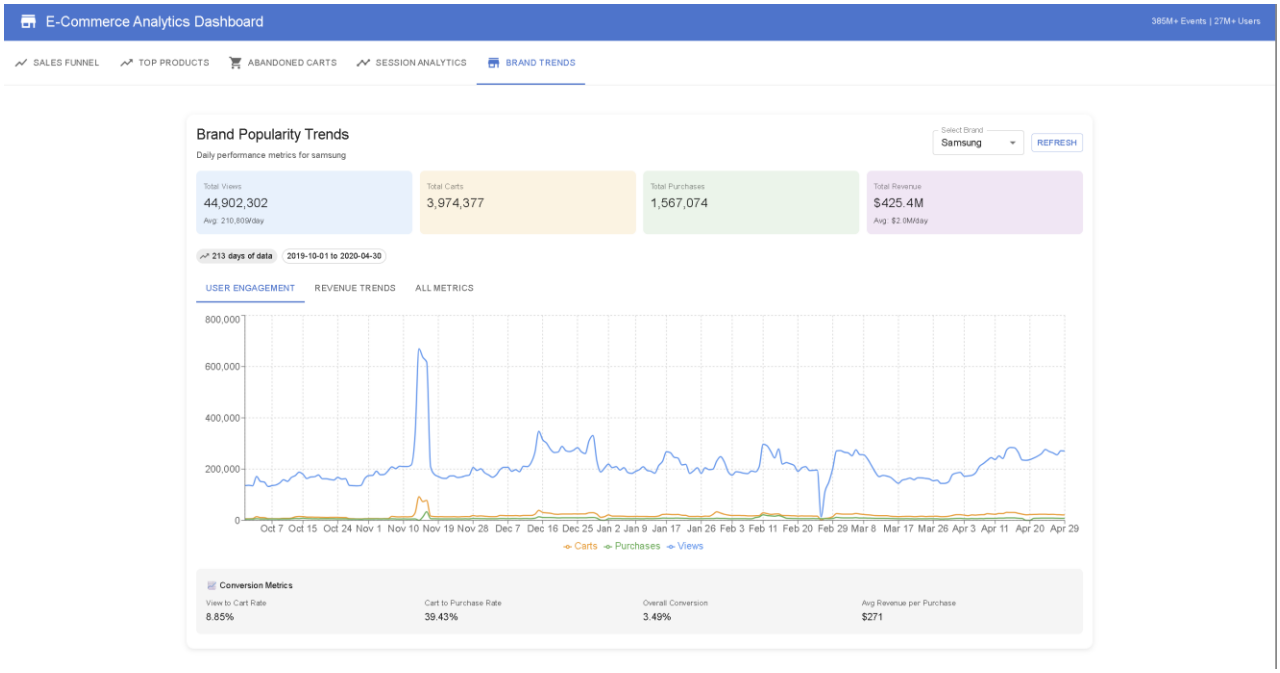
Step 3: The system displays a time-series line chart showing purchase volume over time

Interaction: Hover over data points to see exact purchase counts and dates. The chart is fully interactive with zoom and pan capabilities.

Business Insight: Identify seasonal trends, marketing campaign effectiveness, and brand popularity shifts over the 7-month period.

4.7 Performance Notes

- Query Response Time: All features respond in under 1 second
- Initial Load: First page load may take 2-5 seconds due to API cold start
- Subsequent Queries: Cached API responses return in <500ms
- Browser Compatibility: Fully tested on Chrome, Firefox, Safari, and Edge



5. Code Repository and Deployment

5.1 Repository Access

The complete project source code is available at:

GitHub Repository: https://github.com/harsit14/ecommerce-analytics-dashboard

The repository contains all source code, configuration files, and comprehensive documentation necessary to replicate the entire system locally or deploy to cloud infrastructure.

5.2 Repository Structure

The codebase is organized into three main directories:

5.2.1 Backend Directory (backend/)

- main.py: FastAPI application with all API endpoints
- database.py: Database connection pool configuration
- models.py: Pydantic data models for request/response validation
- requirements.txt: Python dependencies (FastAPI, psycopg2, etc.)
- Dockerfile: Container configuration for Cloud Run deployment
- .env.example: Template for environment variables

5.2.2 Frontend Directory (frontend/)

- src/components/: React components for each of the 5 analytical features
- src/services/apiService.js: API integration layer with axios
- src/App.js: Main application component with routing
- package.json: Node.js dependencies (React, Material-UI, Recharts)
- public/: Static assets and index.html

5.2.3 Documentation (docs/ and root)

- README.md: Comprehensive project documentation
- Database scripts: SQL scripts for schema creation and materialized views

5.3 Local Replication Instructions

The README.md file in the repository provides detailed step-by-step instructions for replicating the entire system locally. Key steps include:

Backend Setup

1. Create Python virtual environment: python -m venv venv
2. Install dependencies: pip install -r requirements.txt
3. Configure .env file with database credentials
4. Run API: uvicorn main:app --reload --port 8000

Frontend Setup

5. Install Node.js dependencies: npm install
6. (Optional) Configure API URL for local backend
7. Start development server: npm start

Database Options

Option 1 (Recommended): Connect to the deployed Cloud SQL database using credentials in .env file

Option 2 (Full Replication): Download 52GB Kaggle dataset, create local PostgreSQL database, run ETL pipeline (16-18 hours), and create materialized views

5.4 Code Documentation Quality

The codebase includes comprehensive documentation:

- Inline Comments: All complex logic and database queries are annotated
- Function Docstrings: Python functions include parameter and return type documentation
- API Documentation: FastAPI automatically generates interactive API docs at /docs endpoint

- README: Comprehensive guide with architecture diagrams, setup instructions, and troubleshooting
- Environment Templates: .env.example files with all required variables

5.5 Live Deployment URLs

- Dashboard: https://databaseproject14.netlify.app

6. Conclusion

This project successfully demonstrates how strategic database design can transform an intractable big data problem into an interactive, real-time analytics platform. By processing a massive 52GB dataset containing over 200 million user interaction events, we achieved sub-second query response times through the synergistic application of three critical optimization techniques:

- Monthly table partitioning: Reduced query scan sizes by up to 85% through partition pruning
- Strategic B-tree indexing: Enabled fast lookups on high-cardinality columns
- Materialized views: Eliminated the need to recalculate complex aggregations on every query

The result is a production-ready system that processes queries which would typically require 5-10 minutes or more in under one second; a 300-600x performance improvement. This transformation makes real-time business intelligence feasible even on datasets that exceed the memory and processing capabilities of standard analytical tools.

Key Achievements

- Successfully implemented a star schema database design optimized for analytical workloads
- Processed 384,847,064 events across 24.5M users and 240K+ products with sub-second response times
- Developed five distinct analytical features providing comprehensive business intelligence
- Built a full-stack application with modern React frontend and FastAPI backend
- Deployed to production on cloud infrastructure (Netlify, Google Cloud Run, Google Cloud SQL)
- Demonstrated mastery of advanced database concepts including partitioning, indexing, and materialized views

Technical Lessons Learned

This project reinforced fundamental principles of database system design for big data applications. The single most important lesson is that the right database design can make the difference between an unusable system and a high-performance analytics platform. No amount of frontend optimization or API caching can compensate for inefficient database queries scanning millions of unnecessary rows.

Partitioning proved to be the foundation of our performance gains. By organizing data by time period, the most common filter in analytical queries we ensured that the database only scans relevant partitions rather than the entire 400-million-row table. Combined with targeted indexes on join columns and pre-computed materialized views for complex aggregations, we achieved the "compute once, query many times" paradigm essential for real-time analytics.

Cloud deployment introduced additional considerations around connection pooling, cold start optimization, and distributed architecture, but the core database design remained the dominant factor in overall system performance.

Future Enhancements

While the current system achieves all project objectives, potential future enhancements include:

- Incremental materialized view refresh to support real-time data updates
- User cohort analysis and segmentation features
- Machine learning integration for predictive analytics (purchase probability, churn prediction)
- Additional time-series visualizations with custom date range selection
- Export functionality for reports and dashboards

Final Thoughts and Milestone 3 Presentation Link

This project successfully demonstrates that with careful database design and optimization, it is possible to build interactive, real-time analytics systems on truly massive datasets. The techniques employed like partitioning, indexing, and materialized views are fundamental to modern data warehousing and business intelligence platforms, and this project provides a practical, working example of their application.

Presentation Link - Presentation Ecommerce Analytics