Stripe Data Architecture

Stripe is a leading global financial technology company, founded in 2010, that powers online payment processing for millions of businesses across over 120 countries. With billions of transactions processed annually and clients ranging from startups to Fortune 500 companies, Stripe operates at massive scale and complexity. As its operations have grown, Stripe's data architecture has become a strategic priority, requiring the integration in a single system of a large variety of data. This proposal outlines a comprehensive data infrastructure designed to ensure performance, consistency, and compliance while enabling advanced use cases such as fraud detection, customer insights, and predictive analytics.

Architecture Overview

The data integration architecture is presented in figure 1. It follows a hybrid model combining real-time streaming and batch processing, supporting low-latency data sync for operational use cases (e.g., fraud detection) and high-throughput batch processing for analytics.

- Data streams originating from Stripe API (e.g. bank transaction information, telemetry) are pipelined to the relevant database systems using kafka streams.
- A reference database holds slowly changing reference data such as country and currency codes, merchant information of currency exchange rates. Any change in this data is reflected to the systems that depend on it through change data capture (CDC).
- Data is archived periodically in a data lake, with batch processing managed by Apache Airflow.
- The loading of less time-sensitive data such as audit logs or customer feedback is handled by Airflow batches.

We present in table 1 a list of possible providers for the various systems of our architecture.

Table 1: Proposed providers for the various systems of the architecture.

System	Provider
OLTP/OLAP	Snowflake, Redshift
NoSQL	MongoDB, DynamoDB
Data Lake	Amazon S3, Azure Data Lake

Reference Database

Slowly changing or static reference data (e.g. country reference, merchant information, currency change rates) is stored in a reference database that serves as a single source of truth. The OLTP, OLAP and NoSQL systems incorporate a copy of the relevant reference tables for faster access. Central updates are propagated through change data capture (CDC). The data is subject to the security and compliance policy described below (e.g. encryption of merchant information). This approach has the advantage of a finer-grained monitoring and control of data access and modification. We present in table 2 a data dictionary for some tables in the reference database.

Online Transaction Processing (OLTP) Data Model

Our proposed OLTP database architecture is presented in figure 2. The core of the database is a registry of all financial transactions occurring within Stripe scope. Tables containing information about merchants and customers are cached from the reference database. Fraud indicators are stored in a dedicated

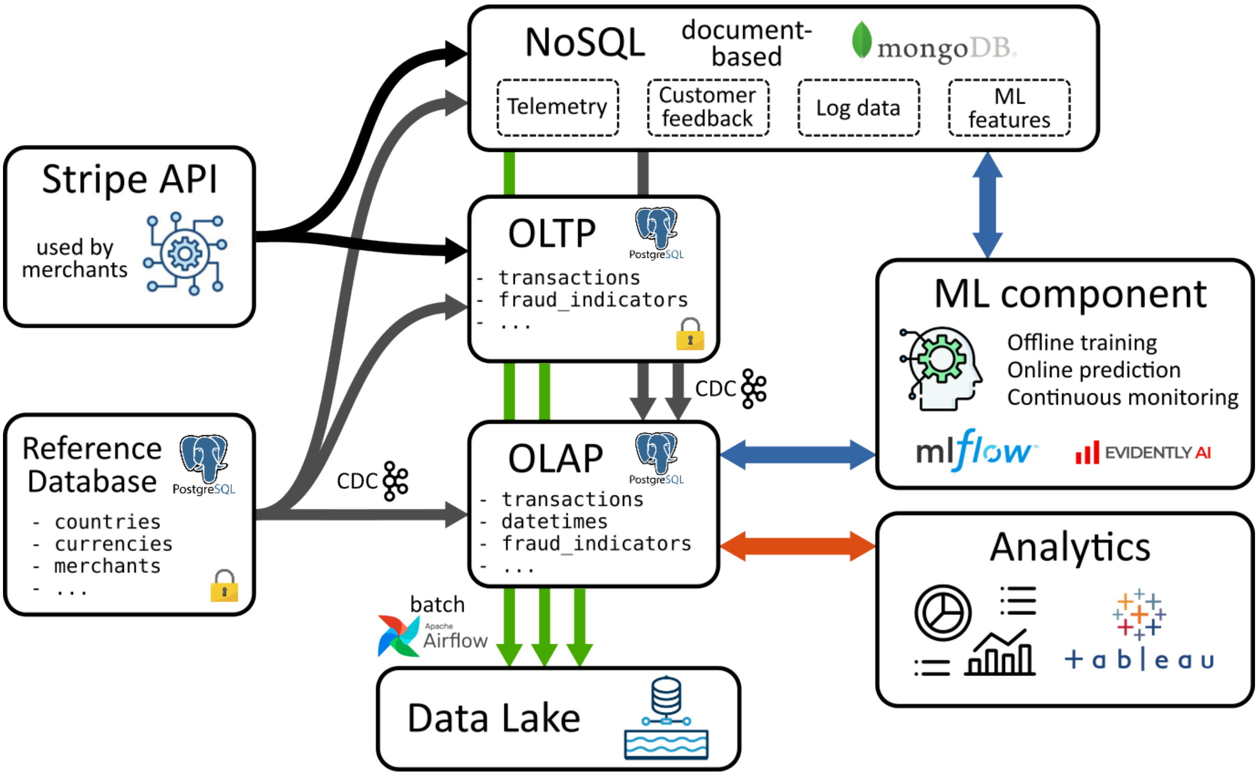

Figure 1: Overview of the proposed data architecture. The description is given in the text. The lock symbols in the OLTP and Reference databases indicate that the security of these systems is critical.

table in a one-to-one correspondence with the main transactions table. The rationale behind this choice is that fraud indicators originate from a different pipeline (e.g. reports after bank investigation).

Online Analytical Processing (OLAP) Data Model

Pour l'analyse et la préparation de features pour la détection de fraude, on utilisera une base

NoSQL Data Model

L'entreprise doit aussi gérer des données semi-structurées telles que les fichiers de log, les reçus de transactions ou encore la télémétrie effectuée sur la plateforme de paiement. Les contraintes pour chaque catégorie de données sont diverses et on adoptera une solution basée sur une document-based NoSQL database.

Stripe must handle semi-structured and unstructured data not well-suited to relational models:

- **Telemetry**. Click stream events, user interactions, session data from the payment API.
- **Customer feedback**. Surveys responses, reviews, support transcripts.
- **Log data**. Access and error logs, API usage.
- **Machine learning features**. Generated or derived features used in model inference.

We recommend the use of a document-based NoSQL database, such as MongoDB, due to its flexibility in handling nested and heterogeneous data structures.

///The model will follow a domain-oriented, loosely coupled design with a focus on performance and future extensibility.

- Les informations de session ayant principalement des contraintes de disponibilité associée à des requêtes simples, on s'orientera vers une database de type key-value.

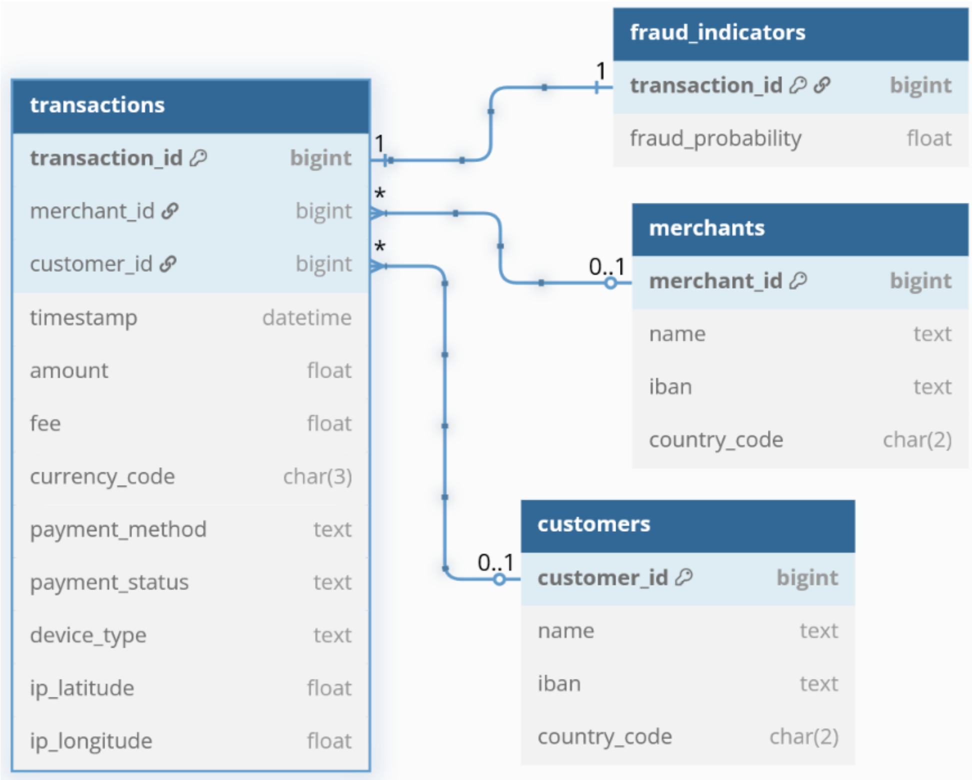

Figure 2: Proposed OLTP database structure.

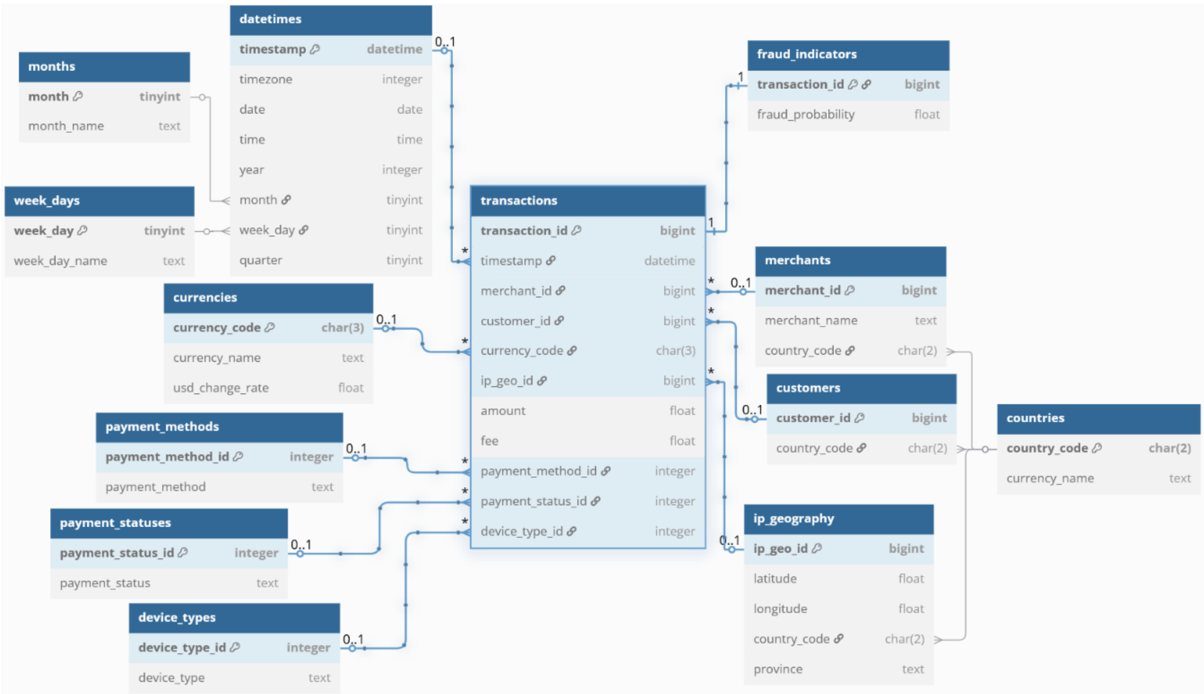

Figure 3: Proposed OLAP database structure.

Table 2: Data dictionary for the `transactions` OLTP schema.

Field Name	Type	Description	Example
		`countries` table	
`country_code`	`char(2)`	ISO 3166-1 2-letter	`'GB'`
`country_name`	`text`	Country name	`'United Kingdom'`
		`currencies` table	
`currency_code`	`char(3)`	Currency code (ISO 4217)	`'EUR'`
`currency_name`	`text`	Currency name	`'Euro'`
`usd_change_rate`	`char(2)`	Currency to USD change rate	`1.08`
		`merchants` table	
`merchant_id`	`bigint`	Unique merchant id	`12345`
`name`	`text`	Merchant name	`'Amazon UK'`
`iban`	`text`	Merchant IBAN	`'GB82WEST12345678765432'`
`country_code`	`char(2)`	Merchant registration country	`'GB'`
		`customers` table	
`customer_id`	`bigint`	Unique customer id	`234567`
`name`	`text`	Customer name	`'John Doe'`
`iban`	`text`	Customer IBAN	`'GB82WEST12345678765432'`
`country_code`	`char(2)`	Customer country code	`'GB'`

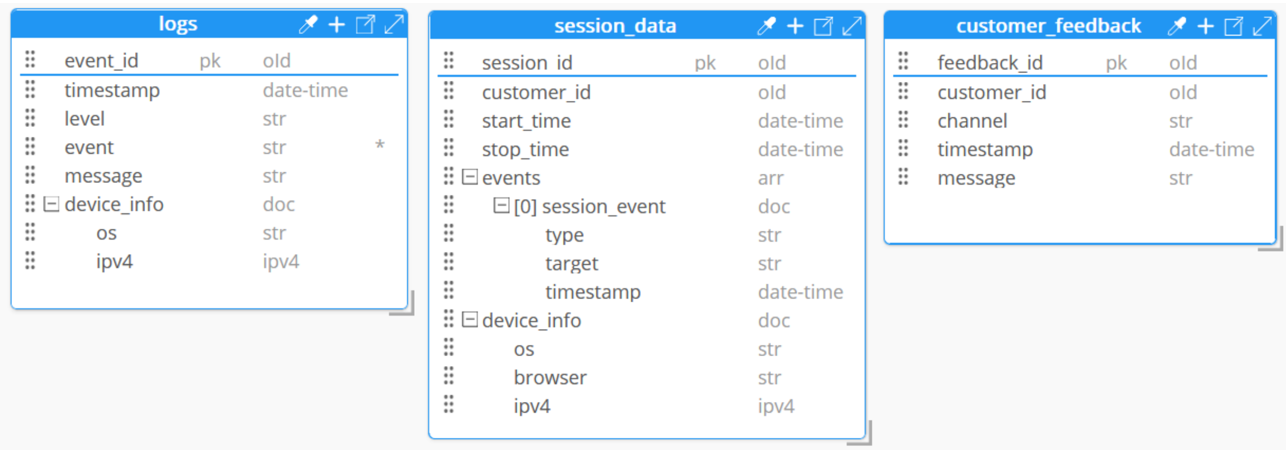

Figure 4: Illustration of the NoSQL database structure. The schema is made using Hackolade .

- Les fichiers de log sont structurés comme une succession d'évènements. Ceux-ci doivent être stockés de façon à pouvoir alimenter en temps réel des algorithmes de détection d'anomalies, afin d'assurer une réponse rapide en cas d'incident. Il est aussi nécessaire qu'ils soient lisibles par un humain pour une analyse approfondie. On s'orientera donc naturellement vers une document-based NoSQL database pour ce cas d'usage.
- Les fichiers non sensibles peuvent être stockés dans un espace approprié (Amazon S3) et indexés par une document-based database qui contiendra aussi les métadonnées.
- La télémétrie dans une column database
- Les features pour le machine learning peuvent être stockés dans une graph database.

Security and Compliance

Stripe handles highly sensitive user data, including banking and payment information. Securing this data is critical for two main reasons: first, to comply with international and regional regulations such as

Table 3: Data dictionary for the transactions OLTP schema.

Field Name	Type	Description	Example
		transactions table	
transaction_id	bigint	Unique transaction id	123456789
merchant_id	bigint	Merchant id	12345
customer_id	bigint	Customer id	234567
timestamp	datetime	UTC transaction timestamp	2023-11-18T17:43:02.4
amount	float	Transaction amount	43.15
fee	float	Transaction fee	0.53
currency_code	char(3)	Currency code (ISO 4217)	'GBP'
payment_method	text	Payment method	'credit_card'
payment_status	text	Transaction status	'sucess'
device_type	text	Device used for payment	'mobile'
ip_latitude	float	IP-based geolocation latitude	49.6833300
ip_longitude	float	IP-based geolocation longitude	10.5333300
		fraud_indicators table	
transaction_id	bigint	Transaction id	123456789
is_fraud	boolean	Is the transaction a fraud ?	false

GDPR and PCI-DSS; and second, to preserve customer trust. A data breach could result in reputational damage and potential revenue loss.

To minimize the attack surface, sensitive data is isolated within the OLTP system, where it is encrypted both at rest and in transit. Along with the reference database, this system serves as the primary and most secure repository for confidential information.

Some of this data must be made available for analytical purposes, such as detecting financial crimes (e.g., fraud or money laundering). To mitigate risks, the OLAP system only stores anonymized or tokenized derivatives of sensitive fields, such as bank location or institution name, with no direct identifiers. These fields are not encrypted to preserve query performance, but access is strictly controlled through role-based permissions and secure data transfer protocols.

The NoSQL system does not store any sensitive information. Instead, it is used for storing and indexing semi-structured and unstructured data, such as logs or behavioral metadata, none of which poses regulatory risks if properly managed.

Files are handled according to their security classification. Highly sensitive files, like bank receipts, are stored in a dedicated, access-restricted data lake and are indexed only through the OLTP system (not represented in the main architecture diagram). All other files are similarly stored in a general-purpose data lake and indexed via the NoSQL system.

Encrypted backups are created and refreshed at regular intervals to ensure disaster recovery capabilities in the event of a major incident.

Finally, a centralized logging system tracks all server connections, data accesses, and queries. Anomaly detection mechanisms can be integrated to identify and alert on suspicious activities in real time.

Machine Learning Integration

Machine learning (ML) plays a critical role in Stripe's data ecosystem, enabling real-time fraud detection, customer segmentation, and predictive analytics. These use cases require a tightly integrated ML pipeline that supports both batch and real-time processing, and can handle structured as well as unstructured data. The pipeline fulfills three primary functions:

- **Feature extraction and model training**. Features are extracted from OLAP and NoSQL systems, with orchestration handled by tools such as Apache Airflow. Models are trained in cloud-based environments (e.g., AWS EC2) and managed using a model registry (e.g., MLflow) to ensure traceability and version control.

Table 4: Data dictionary for the main tables in `transactions` OLAP schema.

Field Name	Type	Description	Example
		`transactions` table	
transaction_id	bigint	Unique transaction id	123456789
merchant_id	bigint	Merchant id	12345
customer_id	bigint	Customer id	234567
timestamp	datetime	UTC transaction timestamp	2023-11-18T17:43:02.4
amount	float	Transaction amount	43.15
fee	float	Transaction fee	0.53
currency_code	char(3)	Currency code (ISO 4217)	'GBP'
payment_method_id	integer	Payment method id	1
payment_status_id	integer	Payment status id	2
device_type_id	integer	Device id	3
ip_geo_id	bigint	IP geolocation id	123456
		`datetimes` table	
timestamp	datetime	UTC transaction timestamp	2023-11-18T17:43:02.4
timezone	integer	Timezone offset in minutes	-120 for UTC-02:00
date	date	Transaction date	2023-11-18
time	time	UTC transaction time	17:43:02.4
year	mediumint	Transaction year	2023
month	tinyint	Transaction month	11
week_day	tinyint	Transaction week day	6 (saturday)
quarter	tinyint	Transaction quarter	4
		`fraud_indicators` table	
transaction_id	bigint	Transaction id	123456789
is_fraud	boolean	Is the transaction a fraud ?	false
fraud_score	float	Predicted fraud probability	0.12
		`ip_geography` table	
ip_geo_id	bigint	IP geolocation id	123456
latitude	float	IP-based geolocation lat.	49.6833300
longitude	float	IP-based geolocation long.	10.5333300
country_code	char(2)	ISO 3166-1 alpha-2	'DE'
province	text	Province name	'Darmstadt'

- **Real time inference**. Models are exposed via APIs and queried asynchronously, typically by the OLTP system, to support real-time decision-making (e.g., fraud scoring). Inference responses can trigger automated actions such as alerts or transaction blocks. Models are deployed within a high-availability infrastructure (e.g., Kubernetes) to ensure scalability and low-latency responses under high transaction loads.

- **Batch predictions**. Models are also executed on a scheduled basis to generate insights for use cases like churn prediction or personalization. The resulting predictions are written back to OLAP or NoSQL systems, depending on their usage, and are consumed by recommendation engines or business intelligence tools.

Monitoring tools are deployed to track data drift, feature distribution changes, and prediction accuracy over time. Alerts are triggered when drift exceeds defined thresholds. Retraining can be initiated either on a fixed schedule or manually by an operator based on monitoring insights.