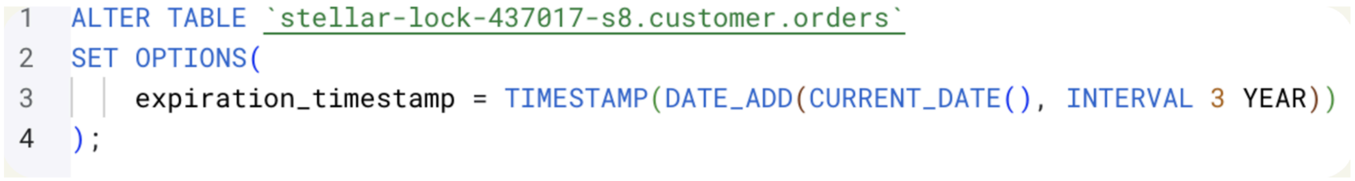
Data Governance Strategies

Data Retention Policies: Effective data retention policies were established to manage the lifecycle of stored data and ensure compliance with regulatory requirements. The steps taken include:

- Automated Data Lifecycle Management:
  - Policies were configured to automatically delete records that are outdated or no longer relevant. This minimizes storage costs and optimizes database performance
  - For example, the SQL command shown applies an expiration timestamp to a table, specifying that records will be deleted three years after the current date
- Regulatory Compliance:
  - Compliance with data privacy laws like GDPR was prioritized, ensuring data is removed after its intended use or within a predefined timeframe
  - This aligns with legal mandates requiring organizations to handle personal data responsibly, minimizing risks of non-compliance penalties

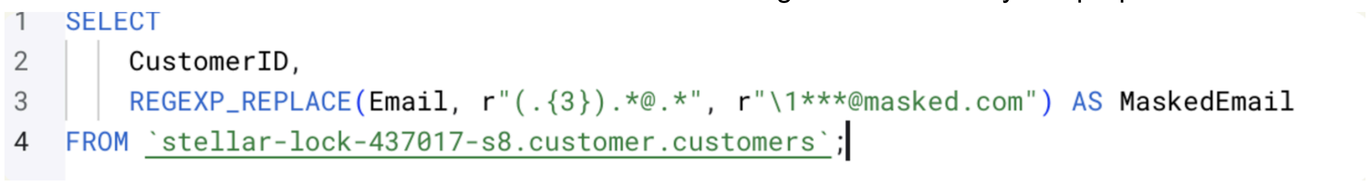


```sql
ALTER TABLE `stellar-lock-437017-s8.customer.orders`
SET OPTIONS(
    expiration_timestamp = TIMESTAMP(DATE_ADD(CURRENT_DATE(), INTERVAL 3 YEAR))
);
```

Figure1: Example code to implement Data Retention Policies

Masking and Tokenization: To enhance data security, sensitive information such as Personally Identifiable Information (PII) was protected through masking and tokenization techniques. The following measures were implemented

- Masking Sensitive Data:
  - Data masking was applied to partially obfuscate sensitive information while retaining data integrity for analytical purposes
  - For example, email addresses were masked using a REGEXP_REPLACE function to anonymize identifiable portions. The resulting format, e.g., ***@masked.com, ensures privacy without compromising usability in non-identifying analyses
  - The SQL snippet illustrates a practical example of masking:
    - The REGEXP_REPLACE function locates and replaces parts of email strings while preserving the domain for partial visibility
    - This ensures data obfuscation while maintaining context for analytical purposes



```sql
SELECT
    CustomerID,
    REGEXP_REPLACE(Email, r"(.{3}).*@.*", r"\1***@masked.com") AS MaskedEmail
FROM `stellar-lock-437017-s8.customer.customers`;
```

Figure2: Example code to implement masking and tokenization