\*\*Backend Schema Document: Google Ads KPI MCP Server Prototype (Final Enhanced)\*\*  
  
This document provides a comprehensive outline of the backend schema for the Google Ads KPI MCP server prototype. It details the data storage, API integration, and handling mechanisms, explicitly differentiating between the initial prototype implementation and potential future enhancements aimed at supporting data visualization within Claude Artifacts and ensuring production readiness.  
  
\*\*1. Authentication & Security:\*\*  
  
\* \*\*Google Ads API Access (Developer Token):\*\*  
 \* \*\*For Now (Prototype):\*\* The Google Ads Developer Token will be securely stored as an environment variable on the server hosting the prototype.  
 \* \*\*Future Plans (Iteration & Enhancement - Production Considerations):\*\* For a production deployment, a dedicated and robust secrets management solution such as HashiCorp Vault, AWS Secrets Manager, or Google Cloud Secret Manager will be implemented to ensure the secure storage and management of sensitive API credentials.  
\* \*\*Refresh Logic and Token Expiry Handling:\*\* As the initial prototype utilizes the Google Ads Basic API Access Developer Token, which does not expire, refresh token logic is not required at this stage. However, future iterations may explore OAuth 2.0 for authentication, which would necessitate the implementation of secure refresh token storage and expiry handling mechanisms.  
\* \*\*Access Control:\*\* Authorization to interact with the Google Ads API will be managed within the backend application (`google\_ads\_client.py`). The presence of a valid and correctly configured Developer Token will be the primary mechanism for granting access. Future enhancements might involve more granular access control based on user roles or permissions.  
  
\*\*2. API Integration & Data Flow:\*\*  
  
\* \*\*SDK Usage:\*\*  
 \* The Python MCP SDK (`modelcontextprotocol`) will be the interface for receiving requests from Claude Desktop and sending responses. This SDK will be utilized within the `mcp\_server.py` module.  
 \* The Google Ads Python SDK (`google-ads`) will be the primary tool for interacting with the Google Ads API to retrieve campaign performance data. This SDK will be used within the `google\_ads\_client.py` module.  
\* \*\*Data Retrieval and Chunking:\*\*  
 \* \*\*For Now (Prototype - Clarify Chunking Boundaries):\*\* To accommodate Claude's context window limitations and facilitate data visualization, the backend will implement a chunking strategy based on the segmentation parameters specified in the MCP request:  
 \* \*\*Campaign Type:\*\* If the request includes segmentation by `campaign\_type`, the data will be chunked such that each unique campaign type (e.g., Search, Display, Video) will constitute a separate data block within the MCP response.  
 \* \*\*Campaign Name:\*\* When segmentation by `campaign\_name` is requested, the data will be chunked by individual campaign names. To prevent excessively large responses, a configurable limit (e.g., 50) will be placed on the number of campaign names included in a single chunk.  
 \* \*\*Campaign Label:\*\* If segmentation by `campaign\_label` is present, each unique campaign label will define a data chunk.  
 \* \*\*Multiple Segmentation Dimensions:\*\* In cases where multiple segmentation dimensions are combined using an "AND" logic, the primary chunking will be performed based on the \*first\* segmentation dimension specified in the MCP request. More advanced multi-dimensional chunking strategies will be explored in future iterations.  
 \* The `mcp\_server.py` will process the segmented data received from `google\_ads\_client.py` and structure the MCP response to contain an array of `payload` sections, where each section represents a distinct chunk of data corresponding to a segment. Claude Desktop will be expected to handle a single MCP response that may contain multiple such data chunks for visualization purposes.  
 \* \*\*Future Plans (Iteration & Enhancement - Chunking Strategy Clarity):\*\* Future enhancements to the chunking strategy may include:  
 \* Dynamically adjusting chunk sizes based on estimated token usage.  
 \* Implementing explicit pagination controls within the MCP protocol.  
 \* Allowing Claude Desktop to specify preferred chunking parameters in the request.  
 \* Developing more sophisticated algorithms for handling complex multi-dimensional segmentation scenarios to optimize data presentation for visualization.  
  
\*\*3. Data Handling, Caching & Light DB Strategy:\*\*  
  
\* \*\*Local DB Implementation:\*\*  
 \* \*\*For Now (Prototype):\*\* SQLite will serve as the light database for caching Google Ads API responses. The database file (`cache.db`) will reside in the project's root directory for ease of setup.  
 \* \*\*Future Plans (Iteration & Enhancement - Scalability Planning):\*\* As data volumes and request frequencies are expected to grow beyond the prototype phase, more robust and scalable database solutions will be evaluated. Potential candidates include PostgreSQL or MySQL for relational data storage and Redis for high-performance in-memory caching. Assumptions regarding data volumes (potentially millions of campaigns, ad groups, keywords) and request frequencies (potentially hundreds or thousands per hour) will guide this transition.  
\* \*\*Table Structures:\*\*  
 \* \*\*`account\_kpi\_cache` Table:\*\*  
 | Column Name | Data Type | Primary Key | Notes |  
 | ------------------- | --------- | ----------- | --------------------------------------------------------------------- |  
 | `cache\_key` | TEXT | YES | Unique key based on account ID, date range, and segmentation parameters |  
 | `account\_id` | TEXT | NO | Google Ads Account ID |  
 | `start\_date` | TEXT | NO | Start date of the cached data (YYYY-MM-DD) |  
 | `end\_date` | TEXT | NO | End date of the cached data (YYYY-MM-DD) |  
 | `segmentation\_json` | TEXT | NO | JSON representation of the segmentation parameters |  
 | `kpi\_data\_json` | TEXT | NO | JSON representation of the cost and conversion data |  
 | `created\_at` | TIMESTAMP | NO | Timestamp of when the data was cached |  
 | `updated\_at` | TIMESTAMP | NO | Timestamp of the last time the cache entry was updated |  
 \* \*\*Future Consideration:\*\* A `visualization\_configs` table could be added to store user-specific preferences for data visualization (e.g., preferred chart types, color schemes) associated with particular Google Ads accounts or query patterns.  
\* \*\*Data Expiry and Freshness:\*\*  
 \* \*\*For Now (Prototype):\*\* A simple time-based cache invalidation strategy will be employed. Cached data will be considered valid for a configurable period (e.g., 15 minutes) based on the `updated\_at` timestamp. If a cached entry is older than this threshold, the backend will fetch fresh data from the Google Ads API and update the cache.  
 \* \*\*Future Plans (Iteration & Enhancement - Advanced Caching & Real-Time Data):\*\* More advanced cache invalidation techniques will be explored, including the potential implementation of a background process that periodically refreshes frequently accessed data. Strategies for handling near real-time data updates, if required, might involve exploring Google Ads API features for notifications or utilizing more sophisticated caching layers like Redis with time-to-live (TTL) settings.  
\* \*\*Chunked Data Storage:\*\* For the initial prototype, the `kpi\_data\_json` field in the `account\_kpi\_cache` table will store the segmented data, which will then be further processed into chunks for the MCP response. More explicit storage of individual chunks in a separate table might be considered in future iterations if the complexity of the chunking strategy increases significantly.  
  
\*\*4. Error Handling & Resilience:\*\*  
  
\* \*\*API Error Management:\*\*  
 \* \*\*For Now (Prototype):\*\* Basic error logging will be implemented to record any issues encountered during communication with the Google Ads API. The `google\_ads\_client.py` will also incorporate a basic exponential backoff retry mechanism for handling transient API errors.  
 \* \*\*Future Plans (Iteration & Enhancement - Monitoring and Alerting):\*\* A comprehensive monitoring and alerting system will be crucial for production. This will involve:  
 \* \*\*Centralized Logging:\*\* Utilizing a logging aggregation service (e.g., ELK stack, Splunk, or cloud-based solutions like AWS CloudWatch Logs or Google Cloud Logging) to consolidate logs from all backend components.  
 \* \*\*Performance Monitoring:\*\* Implementing dashboards (e.g., using Grafana, Kibana, or cloud monitoring services) to track key performance indicators such as API request latency, error rates, and cache hit ratios.  
 \* \*\*Automated Alerting:\*\* Setting up alerts based on predefined thresholds for error rates, latency, or other critical metrics to proactively notify the development team of potential issues.  
\* \*\*Fallback Mechanisms:\*\*  
 \* \*\*For Now (Prototype):\*\* In the event of Google Ads API unavailability or rate limiting, the backend will attempt to serve the most recent valid data from the cache. The MCP response might include an indication that the data could be slightly stale.  
 \* \*\*Future Plans:\*\* More robust fallback mechanisms, such as implementing circuit breaker patterns to prevent repeated failed calls to the Google Ads API, will be considered for enhanced resilience.  
  
\*\*5. Performance & Scalability Considerations:\*\*  
  
\* \*\*Efficient Data Access:\*\*  
 \* \*\*For Now (Prototype):\*\* Appropriate indexes will be created on the `account\_kpi\_cache` table (specifically on `account\_id`, `start\_date`, `end\_date`, and `segmentation\_json`) to optimize query performance during cache lookups. The logic within `google\_ads\_client.py` will be designed to retrieve only the necessary fields from the Google Ads API.  
 \* \*\*Future Plans (Iteration & Enhancement - Detailed Scalability Assumptions):\*\*  
 \* \*\*Data Volumes:\*\* As mentioned earlier, we anticipate potentially very large datasets within Google Ads accounts. The transition from SQLite to a more scalable database solution like PostgreSQL or MySQL will be necessary to handle these volumes efficiently.  
 \* \*\*Request Frequency:\*\* We expect the request frequency from Claude Desktop to increase as the tool gains adoption. Utilizing a high-performance in-memory cache like Redis in front of the primary database could significantly improve response times and reduce the load on the database.  
 \* \*\*Cache Invalidation:\*\* The frequency of cache invalidation will directly impact both data freshness and the number of API calls. Careful consideration and potentially dynamic adjustment of the cache invalidation period will be required based on monitoring data and user feedback.  
\* \*\*Modular Data Processing:\*\* The current modular architecture, separating API interaction, MCP server logic, and caching, will be maintained and further refined in future iterations to facilitate independent scaling and updates of individual components.  
  
\*\*6. Documentation & Extensibility:\*\*  
  
\* \*\*Schema Annotations:\*\* This document serves as the primary source of truth for the backend schema.  
\* \*\*Future Enhancements:\*\* The architecture will be designed with extensibility in mind to accommodate future requirements, such as:  
 \* Integration with additional third-party data sources.  
 \* Support for more complex Google Ads metrics and reporting.  
 \* Implementation of user-specific data filtering and manipulation options.  
 \* Potential integration with other LLM platforms or applications.  
\* \*\*Documentation Updates:\*\* As the backend evolves, meticulous updates to this schema document, as well as any other relevant developer documentation, will be paramount to ensure maintainability and facilitate future development efforts.  
  
\*\*7. Final Considerations:\*\*  
  
The backend schema for the Google Ads KPI MCP server prototype is strategically designed to balance the need for providing timely and relevant Google Ads performance data to Claude Desktop with the inherent limitations of context windows and the potential constraints of the Google Ads API. The caching mechanism, particularly with the planned chunking strategy, is intended to deliver data in manageable segments suitable for visualization within Claude Artifacts. The distinction between the prototype implementation and future plans allows for a pragmatic approach to initial development while keeping scalability and production readiness in sight for subsequent iterations.