

Google Cloud

Unlocking the interconnected enterprise

With graph analytics in Google Cloud.



Table of contents

Introduction	03
The 'why' – why build a graph?	06
From 'what' to 'why'	07
Agentic Data Cloud graph capabilities – Spanner and BigQuery Graph	08
Unified graph strategy with Agentic Data Cloud	11
Practitioner's guide – building your first graph	14
Building your first graph with Spanner Graph	15
Advanced insights: running built-in graph algorithms in Spanner Graph	19
Building your first graph with BigQuery Graph	22
Introduction to BigQuery measures	24
Practical guide – high-impact graph use cases	29
Anti-money laundering and circular fraud prevention	30
Customer 360 and hyper-personalization	33
Supply chain impact analysis in food and beverages	35
Healthcare: metabolic pathway search	37
Cybersecurity: real-time threat correlation and lateral movement hunting	40
Competitive intelligence: unstructured data to knowledge graphs (SEC filings)	43
Network IT operations: root-cause analysis and autonomous network digital twins	46
Identity resolution and access sprawl management	48
Social media: influencer hub detection	51

01



Introduction

Everything in the world is interconnected

By understanding these connections, we can discover powerful insights and drive better outcomes. In the traditional realm of data, we've spent decades forcing the world into neatly organized tables with rows and columns. But the world doesn't live in a grid – it's structured as a web. Whether it's how money moves between accounts, how customers browse a website, how raw materials are turned into finished products, or how infectious diseases spread through communities – the most valuable insights often live in the connections between entities, not the points themselves.

As we enter the agentic AI era, moving from human-scale insights to agent-scale action, AI agents need more than just raw data; they need context.

By explicitly modeling both entities and their relationships, graphs retain the rich context and connections of the real world – uncovering game-changing insights that power both humans and agents.

Welcome to the era of graph analytics powered by Google's Agentic Data Cloud – where we stop treating entities like people, places, and products, as 'tables', and relationships between them as 'joins', and start treating them as equally valuable, just as they are represented in the physical world.

Agentic Data Cloud provides a unified graph database and analytics solution on Google Cloud, leveraging **Spanner Graph** for operational workloads and **BigQuery Graph** for analytical workloads. By providing purpose-built interconnected graph solutions, Agentic Data Cloud helps you accelerate time-to-action and build intelligent applications without compromising price-performance, latency, scale, or creating data silos. Both graph solutions supercharge data analytics by delivering powerful insights without any data movement. They model existing tables and joins as entities and relationships and use familiar SQL and ISO Graph Query Language (GQL) queries to uncover insights, eliminating the need to move data or learn specialized and proprietary graph tools and query languages.

Finally, these solutions deliver insights at unprecedented scale and speed while keeping costs low, so you can successfully deliver on the 'find the needle in the haystack' use cases by cost-effectively analyzing millions of entities and billions of edges at the speed of thought.

This guide provides an overview of Spanner Graph and BigQuery Graph, including how these solutions work together to provide a comprehensive graph ecosystem. We cover use cases such as product recommendations and next best offer in retail, fraud detection and anti money laundering in financial services, drug discovery and infectious diseases outbreaks in healthcare, supply chain and logistics in manufacturing, and identity and access management in IT.



This guide provides practical guidance including step-by-step instructions, code samples, and demos to help you implement these graph analytics in real-world scenarios. You can get started with either solution and seamlessly add the other solution to get the best price-performance, latency, and functionality based on your use case.

02



The 'why' – why build a graph?



In the traditional data world, the messy, interconnected reality of business was forced into the rigid rows and columns of relational tables and the joins between them. Applying this process overlooks rich context, and subsequently relies on business users and analysts to retrospectively augment the data with context. However, as we enter the **agentic AI** era, businesses require more than just raw data – they require context and causal reasoning.

Because agents operate at a scale and speed that is an order of magnitude higher than humans, the agentic AI era requires graphs to provide the automated multi-hop context needed for agents to evolve from delivering insights to taking trusted actions.

2.1

From 'what' to 'why'

Relational databases are great at answering "what" questions (for example, "What was our total revenue in Q3?"). Meanwhile, graphs excel at answering "why" questions (for example, "Why did churn increase despite high sales?").

By tracing multi-hop relationships – linking a weather event to a factory shutdown, a shipping delay, and finally a customer complaint – graphs provide the reasoning layer that AI agents need to operate autonomously.

Agentic Data Cloud graph capabilities – Spanner and BigQuery Graph

Previously, modeling entities and analyzing relationships required exporting data from operational databases to standalone graph solutions. Additionally, it was necessary to pick an operational graph database or an analytical graph solution based on the operational or analytical nature of the use case. This created data silos, synchronization overheads, and data drift, and also forced latency, concurrency, cost, and scale compromises.

Developers had to learn new languages, data models, and programming constructs just to extract insights, and were often forced to build data pipelines for answers to business questions. Agentic Data Cloud provides a unified graph database and analytics solution on Google Cloud with Spanner Graph for low latency, high concurrency, real-time graph analytics and BigQuery Graph for petabyte-scale analytical workloads. This combined approach overcomes the challenges of managing disconnected proprietary graph solutions for different needs and eliminates data silos, complex integrations, and skill-set gaps.

Feature	BigQuery Graph (analytical)	Spanner Graph (transactional)
Graph model, query, and visualization	Unified graph modeling and graph query language, all powered by GoogleSQL, part of ISO SQL standard. Same interface for graph visualization.	Unified graph modeling and graph query language, all powered by GoogleSQL, part of ISO SQL standard. Same interface for graph visualization.
Primary workload	Analytical: aggregations over massive datasets.	Transactional: high concurrent volume of low latency reads/writes.
Query latency	Seconds to hours. Optimized for scanning terabytes/petabytes.	Milliseconds to seconds. Critical for user-facing apps.
Query pattern	Global or whole graph: "Who has a 'network reach' of more than 5,000 people?"	Local or neighborhood: "Who are friends with my friends, but aren't in my immediate circle?"
Scale	Petabyte-scale, virtually limitless, optimized for large-scale historical data.	Horizontally scalable, virtually unlimited scale, optimized for hot data.
Data freshness	Near real-time or batch. Can access data from different source (for example, Spanner Graph data using Data Boost for Spanner , Bigtable , Cloud Storage , or Amazon S3).	Real-time, strong consistency.
Data input	Data lakes, historical logs, archived transaction data.	Live application streams, user interactions.
Data movement	Spanner to BigQuery (forward extract, transform, and load (ETL)) with pre-built templates (batch, streaming). BigQuery to Spanner (reverse ETL) with EXPORT_DATA . Query federation (zero-ETL) with external schema.	Spanner to BigQuery (forward extract, transform, and load (ETL)) with pre-built templates (batch, streaming). BigQuery to Spanner (reverse ETL) with EXPORT_DATA . Query federation (zero-ETL) with external schema.

Table 1: Google Cloud offers two specialized engines to cover the full graph spectrum.

2.2.1

Spanner Graph for operational workloads

Spanner Graph is a capability built into Google Cloud Spanner that allows users to model and query their data as a property graph directly within their existing Spanner databases. By mapping relational tables to graph elements, like nodes and edges, Spanner Graph enables real-time, operational workloads to leverage graph traversal and pattern matching using syntax compatible with GQL, often embedded within standard SQL queries. This integration provides the benefits of Spanner's global scale, high availability, and strong consistency while avoiding the need for complex ETL processes associated with a standalone graph database. Typical use cases include real-time fraud detection, identity resolution, data lineage, and customer 360.

Key capabilities include:



Zero-ETL schema with ACID consistency:

Integrated table-to-graph mapping directly over existing relational data enforces strict transactional guarantees. Graph mutations occur simultaneously with underlying table updates, completely eliminating the risk of 'dangling edges' commonly found in eventual-consistency graph architectures.



Interoperable querying and algorithms:

Combines GQL with standard SQL and built-in analytical algorithms (such as PageRank). Users can execute complex community detection workflows and immediately query or write the results back to the graph in a single, seamless process.



Advanced search and AI integration:

Built-in vector search, full-text search, and native integration with the Gemini Enterprise Agent Platform.



Network-optimized traversals:

By mapping Spanner interleaved tables as graph edges, relationships are physically co-located on the exact same storage split as their source node. This ensures multi-hop graph traversals execute locally, eliminating latency spikes from cross-server network calls even at global scale.

2.2.2

BigQuery Graph for analytical workloads

BigQuery Graph brings large-scale graph analytics capabilities directly into BigQuery, allowing users to model and query relationships within their massive datasets. By defining property graphs over existing BigQuery tables and views without data movement, BigQuery Graph enables analytical workloads to uncover complex patterns and hidden connections using an interface compatible with GQL, fully interoperable with SQL.

Leveraging BigQuery's scalable and serverless architecture, BigQuery Graph is designed for in-depth analysis of historical data, supporting use cases like fraud-ring analysis, customer 360 insights, and supply chain optimization across petabytes of data.

Key capabilities include:



Zero-ETL graph schema:

Mapping existing BigQuery tables to graph schemas without data movement.



Familiar GQL/SQL querying:

Interoperable GQL/SQL querying for large-scale historical datasets.



Integration with BigQuery capabilities:

Integration with BigQuery AI, vector search, full-text search, geospatial functions, data agents, and Gemini Enterprise Agent Platform.



Graph visualization:

Visual modeling experience in BigQuery Studio provides an intuitive interface to easily build and manage the entities, relationships, and business logic that power your agentic context. Graph query results are displayed in a visually appealing graph format in BigQuery Studio that makes data exploration, investigation, and explanation much easier than writing and running queries for data exploration.



Integration with Spanner Graph:

BigQuery Graph and Spanner Graph share the same graph schema and query language. You can execute operational graph workloads in Spanner and run complex graph analytics in BigQuery without needing to remodel your data or translate your queries.



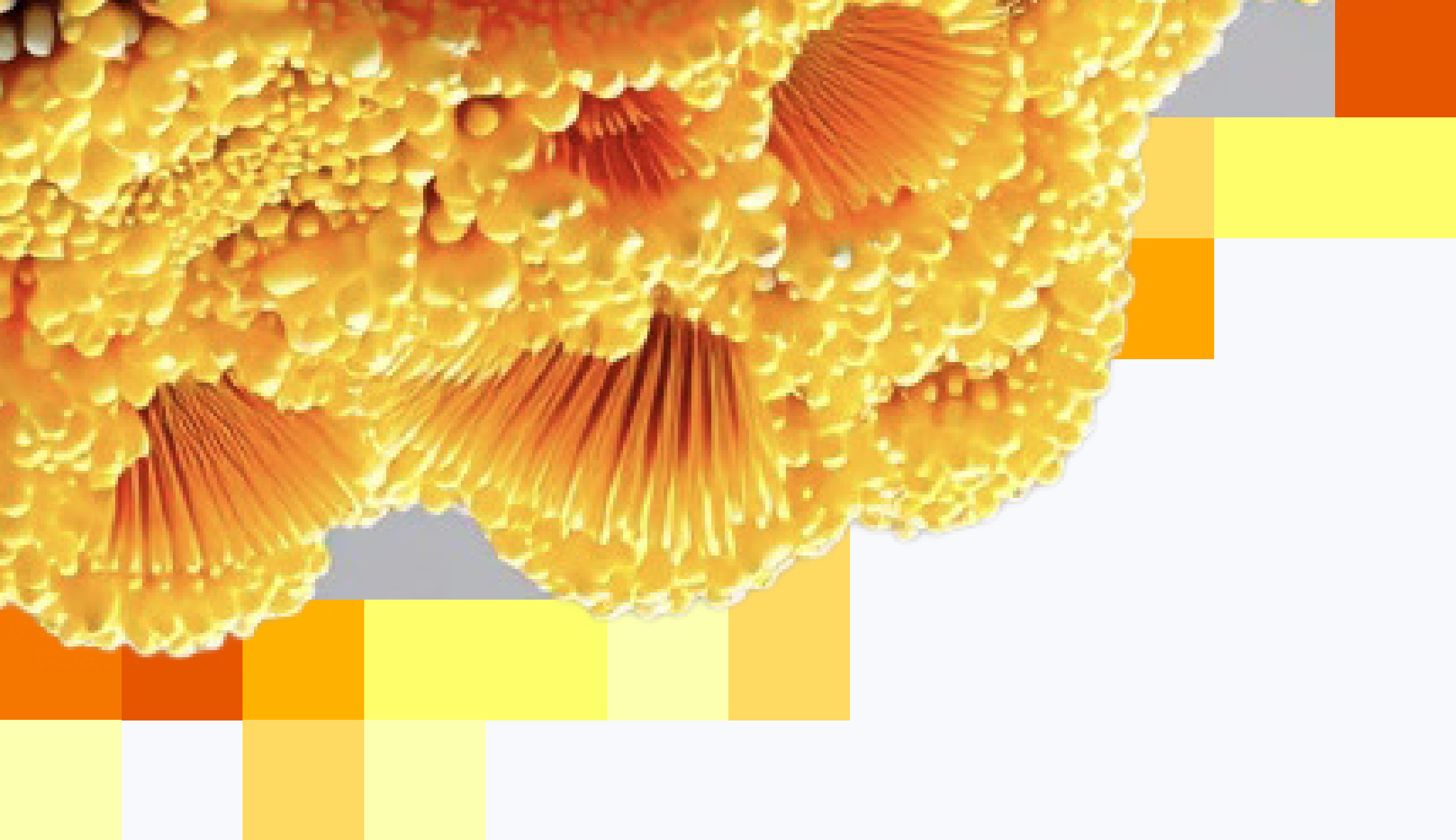
Integration with borderless Lakehouse:

BigQuery Graph works with borderless Lakehouse. This ensures that the rich graph capabilities are available across all data stored across clouds and on-premises in databases, data lakes, and SaaS.



Query using natural language:

Ask questions about your graph using [conversational analytics](#). Agents can write SQL and GQL queries and provide visualizations of your output. Agents can also use descriptions, synonyms, and [measures](#) defined on your graph to improve the quality of the results.



2.3

Unified graph strategy with Agentic Data Cloud

While Spanner Graph and BigQuery Graph are powerful on their own, their true value emerges when they are deployed together. By connecting operational and analytical environments and using common schema and query languages, you eliminate data silos and accelerate your time-to-insight without compromising price-performance, latency, or scale.



Spanner Graph enables Yahoo to unify our data into a connected foundation at a global scale, powering real-time, intelligent decision-making across our agentic advertising platform.

This enhances our AI-driven approaches that drive one of the largest digital advertising ecosystems, and we look forward to building on it with BigQuery Graph to unlock deeper analytics and predictive capabilities to power future innovation."

Gabriel DeWitt,

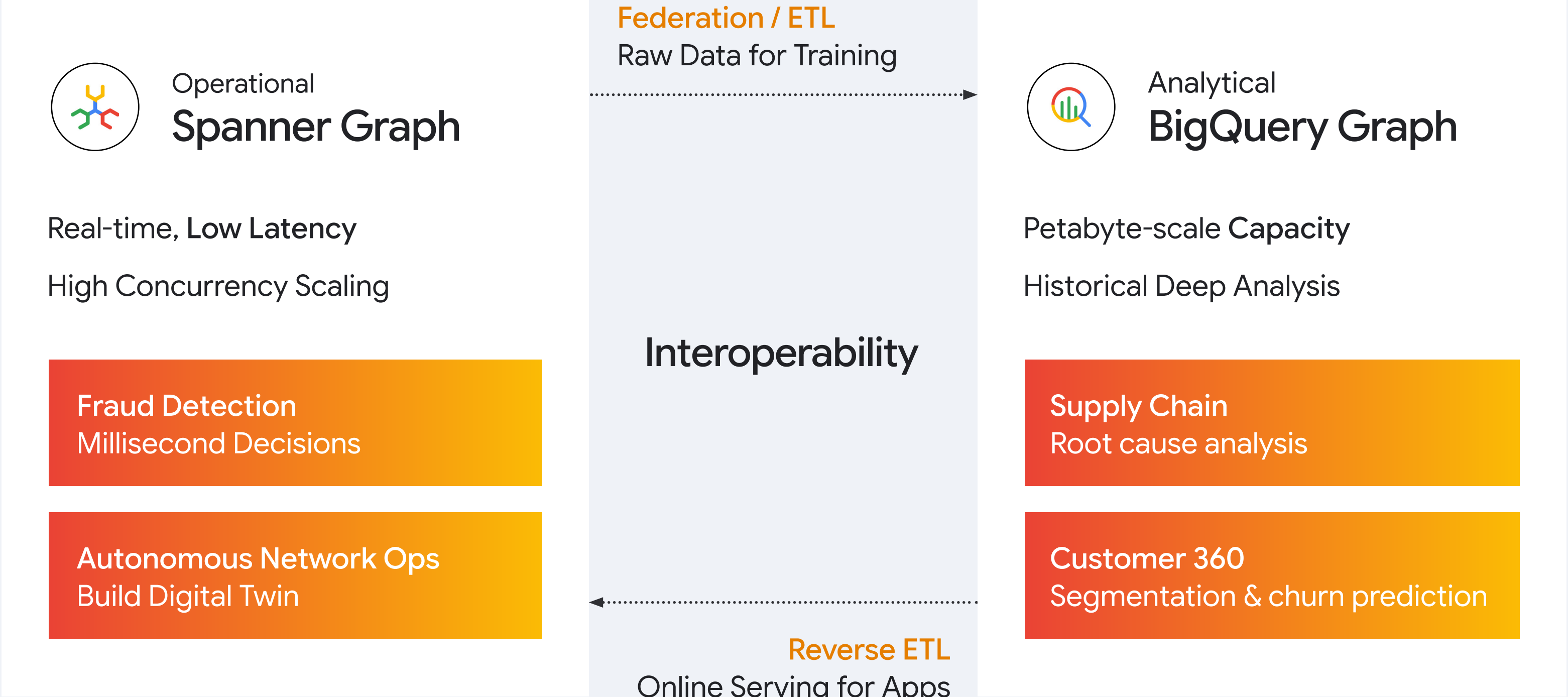
Head of Consumer Monetization,
Yahoo

yahoo!

Unified Experience Architecture

Unified Google Cloud Graph Solution

One Schema, One Language (ISO GQL, SQL/PGQ)



Take financial fraud detection as an example: your application uses Spanner Graph to instantly identify a suspicious connection and block a transaction at checkout. Meanwhile, BigQuery Graph analyzes petabytes of historical transaction data to expose the complex, long-term fraud ring that initiated it.

Figure 1: Better together: Unified graph strategy with Spanner Graph and BigQuery Graph.

Here is how these two engines integrate to create an end-to-end graph workflow:



Query Spanner data in BigQuery Graph through Data Boost:

With Data Boost, you can query Spanner Graph data directly from BigQuery without impacting the performance of your transactional workloads. This allows you to build a 'virtual graph' that combines real-time operational data from Spanner with historical analytics in BigQuery – all without moving any data. For instance, you can combine real-time Account and User nodes from Spanner Graph with historical LogIn edges from BigQuery to identify suspicious login patterns across different devices.



Export BigQuery data into Spanner Graph through reverse ETL:

When you need to bring analytical data back into Spanner for low-latency, real-time querying, you can use reverse ETL without additional pipelines. For example, you can import historical device data (IP addresses, device IDs) from BigQuery into Spanner Graph to enhance your real-time fraud detection operations.



A unified graph query and schema experience:

A core advantage of this solution is the consistent schema and GQL shared across both platforms – this shared language reduces development time and minimizes the friction of context-switching.



Visualize your graph data:

Visualizing connected data is core to data explorations, investigations, and analyses. With Spanner Studio and BigQuery Studio, you can instantly visualize your graph data without leaving your familiar environment or setting up external tools. For deeper programmatic exploration, you can also leverage Spanner Graph notebooks and BigQuery notebooks to render query results directly within your existing data science workflows.

03

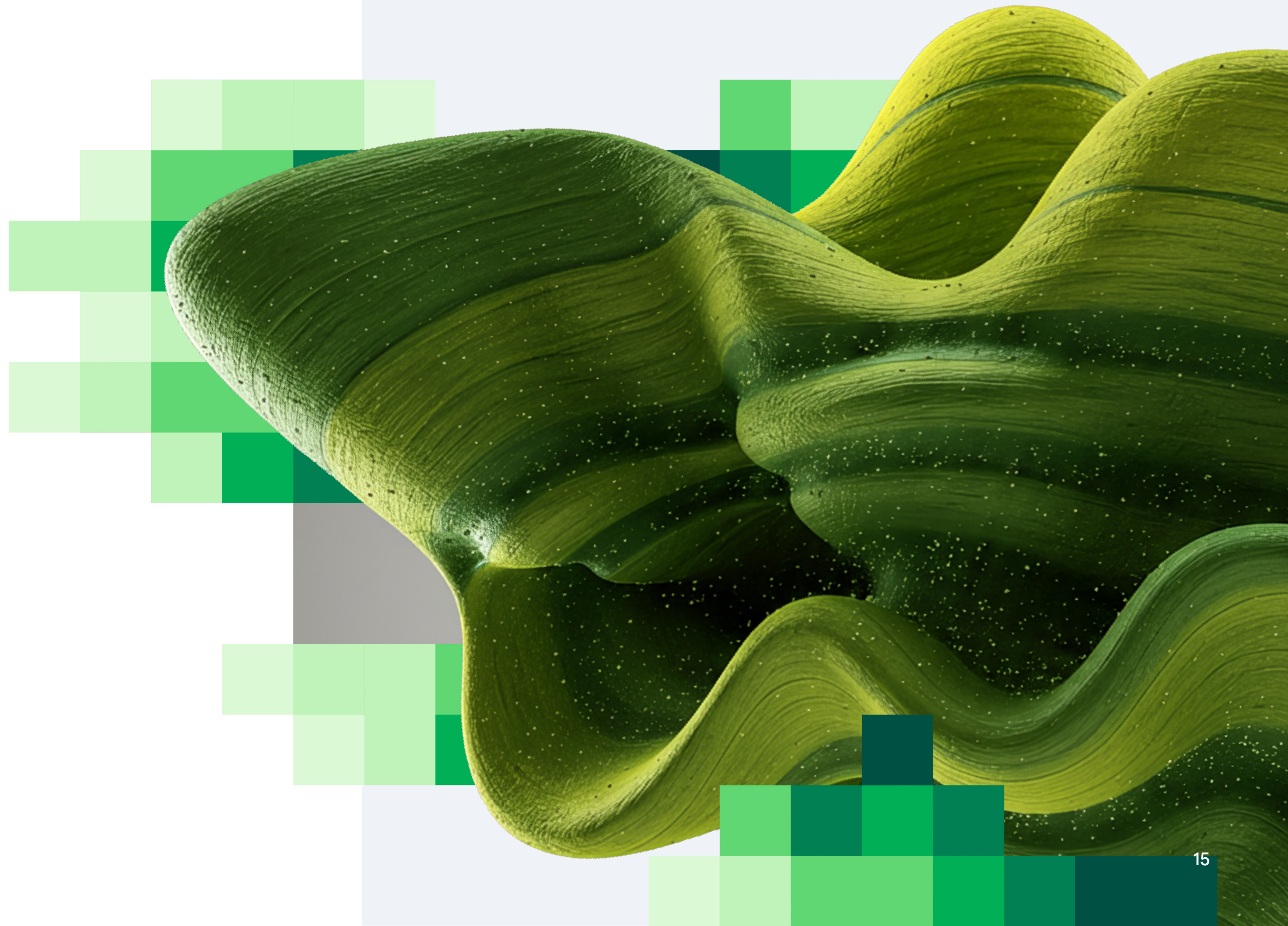


Practitioner's guide – building your first graph

Building your first graph with Spanner Graph.

Building a graph in Spanner Graph allows you to model, query, and traverse highly interconnected data with global consistency and millisecond latencies. Unlike traditional relational setups that rely on heavy runtime **JOIN** operations, Spanner Graph materializes relational tables directly into graph elements like nodes and edges.

For operational workloads that require high throughput and availability, Spanner Graph provides a fully structured approach, via interleaved tables, to guarantee that related nodes and edges are physically co-located on disk, eliminating distributed join overhead.



SQL

```
-- Create the base Node table
CREATE TABLE Node (
  label STRING(256) NOT NULL,
  key STRING(256) NOT NULL,
  detail JSON,
  created_at TIMESTAMP NOT NULL OPTIONS (allow_commit_timestamp = true)
) PRIMARY KEY (label, key);
-- Create the Edge table, interleaved within the Node parent table
CREATE TABLE Edge (
  label STRING(256) NOT NULL,
  key STRING(256) NOT NULL,
  edge_label STRING(256) NOT NULL,
  other_node_label STRING(256) NOT NULL,
  other_node_key STRING(256) NOT NULL,
  detail JSON,
  CONSTRAINT Fk_Src_Node FOREIGN KEY (label, key) REFERENCES Node(label, key)
) PRIMARY KEY (label, key, edge_label, other_node_label, other_node_key),
INTERLEAVE IN PARENT Node ON DELETE CASCADE;
```

Step 1: Define the underlying relational schema (DDL)

Spanner Graph builds a logical graph schema on top of strongly-typed relational tables. To achieve optimal performance under high-throughput write workloads, the **Edge** table is physically interleaved within the parent **Node** table.

Performance note: By declaring **INTERLEAVE IN PARENT Node ON DELETE CASCADE**, Spanner ensures that when an identity, user, or asset node is deleted, all of its outgoing relationship edges are automatically cleaned up in a single transaction without extra engine overhead.

Step 2: Declare the property graph

Once your relational storage layer is ready, map your tables to a property graph using standard Data Definition Language (DDL) syntax. This establishes your nodes and edges as first-class graph elements.

```
SQL
CREATE PROPERTY GRAPH SecurityGraph
  NODE TABLES (Node)
  EDGE TABLES (
    Edge
    SOURCE KEY (label, key) REFERENCES Node (label, key)
    DESTINATION KEY (other_node_label, other_node_key)
    REFERENCES Node (label, key)
  );
```

Step 3: Operational multi-hop traversals using GQL

```
SQL
GRAPH SecurityGraph

MATCH (bot:Node {key: 'Deep-Cover-Bot'})-[:Edge {edge_
label: 'IS_MEMBER_OF'}]->(g:Node)

      -[:Edge {edge_label: 'NESTED_IN'}]->{1, 5}
      (parent:Node)

      -[:Edge {edge_label: 'HAS_PERMISSION'}]->(res:Node)

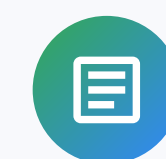
WHERE res.detail.sensitivity = 'High'

RETURN bot.key AS bot_name, g.key AS group_name, parent.key
AS inherited_group, res.key AS resource_name;
```

With the property graph declared, use GQL to perform fast pattern matching and deep traversals. Above is an operational query designed to audit a compromised automated bot account. It tracks down hidden nested privilege escalations by traversing up to five layers of nested groups.

Step 4: Visualizing operational output

Spanner Graph integrates directly into **Spanner Studio**. When queries are executed inside the console, you can instantly toggle from a tabular result view into an interactive node-edge graph visualization. This allows incident response or fraud-prevention teams to visually track blast radiuses, map connection lines, and drill directly into the properties embedded inside the **JSON** details of any given entity.



(Try this step-by-step [tutorial](#) for hands-on experience with Spanner Graph.)



3.2

Advanced insights: running built-in graph algorithms in Spanner Graph

While GQL pattern matching is perfect for local neighborhood traversals, extracting macro-level insights – like identifying an entire fraud ring or finding the most influential hub in a network – requires global graph algorithms. Developed in close collaboration with Google Research Graph Mining, Spanner Graph offers a built-in suite of high-performance algorithms that scale to ten of billions of edges.

The architecture: zero-impact analytical scaling

Running heavy graph computations on a live transactional database traditionally risked degrading application performance. Spanner Graph solves this by offloading algorithm execution to Spanner Data Boost.



Fully managed and serverless:

Algorithms run on isolated, on-demand compute resources. You pay only for the Serverless Processing Units (SPUs) consumed during execution, ensuring near-zero impact on your operational workloads.



Seamless GQL/SQL integration:

Algorithms are invoked directly as function calls inside standard queries. The outputs can be piped out to Cloud Storage or written straight back into your Spanner tables, enriching your existing graph elements.

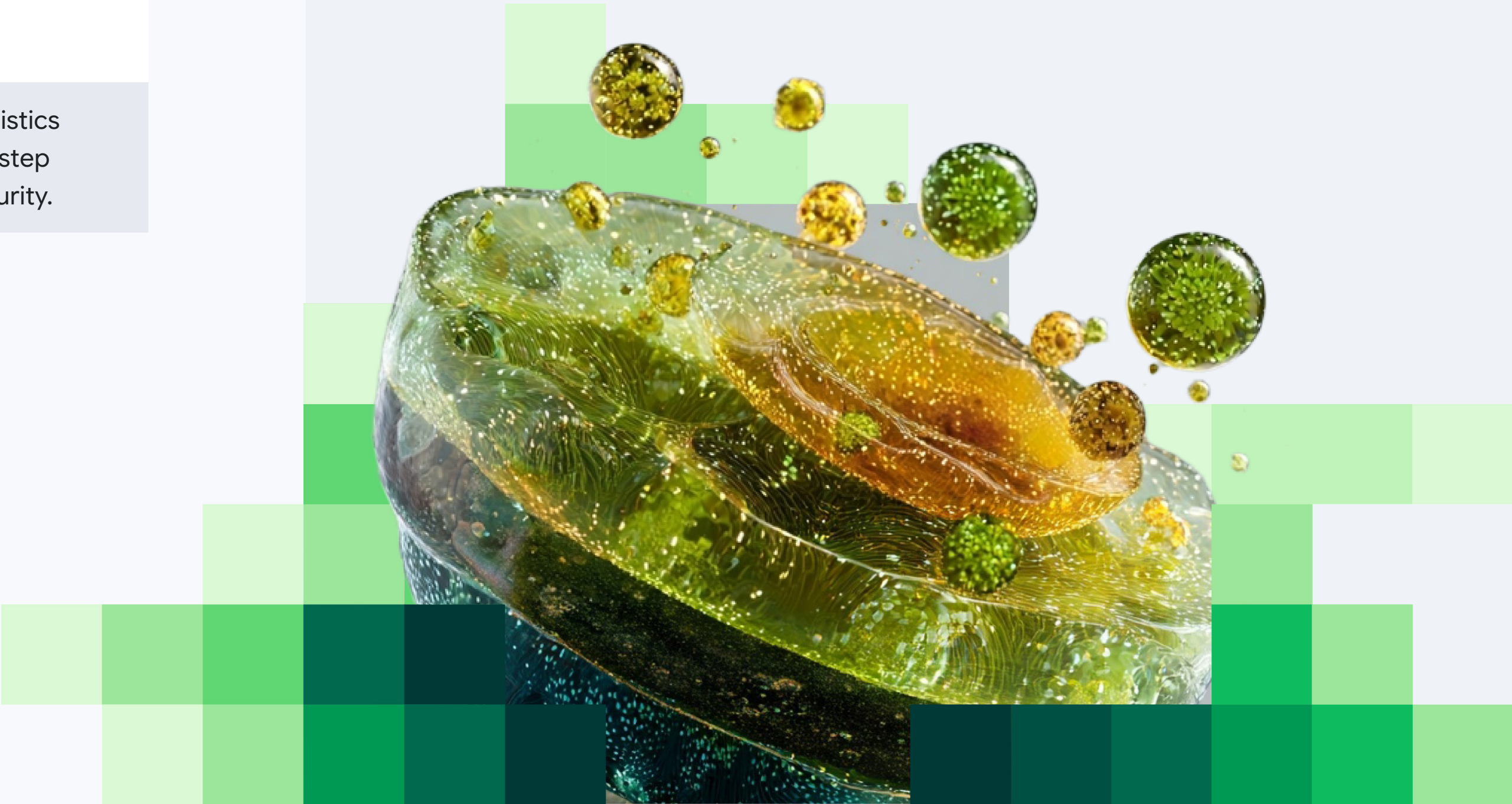
The Spanner Graph algorithm catalog

Spanner Graph groups its native intelligence capabilities into four essential categories:

Category	Supported Algorithms	Common Use Cases
Centrality	PageRank, BetweennessCentrality, ClosenessCentrality	Ranking influential social profiles, pinpointing critical telecom routers, or catching fraud ringleaders.
Clustering	WeaklyConnected Components, ModularityClustering, LabelPropagation, CliqueFinding	Detecting tightly-knit communities, isolating transaction fraud rings, and structural customer segmentation.
Similarity	JaccardSimilarity, CosineSimilarity, CommonNeighborsSimilarity	Driving lookalike recommendation engines based on shared behavioral neighborhoods.
Query pattern	ShortestPath	Supply chain optimization, logistics routing, and tracking step-by-step lateral movement in cybersecurity.

Step-by-step implementation: community detection and write-back

The power of this architecture lies in the **integrated workflow**. In this procedural example, we execute a community detection algorithm over a financial graph to group connected accounts and write the structural results natively back to the **Account** table in a single block.



Step 1: Run community detection and mutate the graph

We call the `ModularityClustering` algorithm and use the `EXPORT DATA` statement with `write_mode = 'update_ignore_all'` to dynamically append the discovered cluster IDs directly to our operational schema:

```
EXPORT DATA OPTIONS(  
  format = 'CLOUD_SPANNER',  
  table = 'Account',  
  write_mode = 'update_ignore_all'  
) AS  
GRAPH FinGraph  
CALL ModularityClustering(  
  node_labels => ['Account'],  
  edge_labels => ['Transfer']  
)  
YIELD node, cluster  
RETURN node.id AS id, cluster AS community_id;
```

Step 2: Isolate and query the suspicious subgraph

Once the graph is enriched with `community_id` values, operational teams can run standard GQL queries to filter on anomalous communities. For instance, once a high-risk community is identified, you can run a targeted `PageRank` algorithm over that specific subgraph to unmask the central ringleader moving the funds.

```
-- Find the community with the highest concentration of  
-- flagged fraud  
GRAPH FinGraph  
MATCH (a:Account)  
WHERE a.community_id IS NOT NULL AND a.fraud_flag = TRUE  
RETURN a.community_id AS community_id, COUNT(*) AS fraud_  
count  
ORDER BY fraud_count DESC  
LIMIT 1;
```

Best practice: To invoke these graph algorithms via the Cloud Console, CLI, or client libraries, your IAM principal must explicitly be granted the `spanner.databases.runGraphAlgorithms` permission, which comes natively bundled inside the `roles/spanner.graphIntelligenceUser` role.

3.3

Building your first graph with BigQuery Graph

Building a graph in BigQuery is a three-step workflow that bridges the gap between relational storage and graph reasoning.

Step 1: Define the property graph (DDL)

You don't move data; you define a metadata layer over existing tables.

You can also use the [visual graph modeler](#) in BigQuery Studio to define nodes and edges from your BigQuery tables, and how to edit and query graph data.

SQL

```
CREATE OR REPLACE PROPERTY GRAPH graph_db.FinGraph
  NODE TABLES (
    graph_db.Account,
    graph_db.Person
  )
  EDGE TABLES (
    graph_db.PersonOwnAccount
      SOURCE KEY (id) REFERENCES Person (id)
      DESTINATION KEY (account_id) REFERENCES Account (id)
      LABEL Owns,
    graph_db.AccountTransferAccount
      SOURCE KEY (id) REFERENCES Account (id)
      DESTINATION KEY (to_id) REFERENCES Account (id)
      LABEL Transfers
  );
```


Step 2: Query and visualize

Use **GQL** to traverse patterns. For example, the following query uses a **MATCH** statement to find information about who Dana transferred money to:

```
SQL
GRAPHgraph_db.FinGraph
MATCH
  (person:Person {name: "Dana"})-[own:Owns]->
  (account:Account)-[transfer:Transfers]-
  >(account2:Account)<-[own2:Owns]-(person2:Person)
RETURN
  person.name AS owner,
  transfer.amount AS amount,
  person2.name AS transferred_to
ORDER BY person2.name;
```

Visualizations can then be generated natively in **BigQuery Studio Notebooks** using `%%bigquery --graph` commands or directly in the **BigQuery Studio** console. Once rendered, you can click elements to view their specific properties and neighbor connections, customize layout styles and node colors, or switch directly to a schema view to inspect the overall graph structure.

For deeper exploratory analyses, BigQuery natively connects to robust third-party visualization platforms like **Graphistry** (offering GPU-accelerated layouts and AI-assisted anomaly detection for high-scale relational data), **Linkurious Enterprise**, and **Kineviz GraphXR**.

3.4

Introduction to BigQuery measures

In traditional relational databases, joining tables with one-to-many or many-to-many relationships often leads to a common analytical pitfall: data fan-out.

When a single row from one table matches multiple rows in another (for example, one department at a university having multiple courses), the parent row is duplicated in the output. If you attempt to calculate an aggregation like `SUM(Budget)` on this joined output, the budget is counted multiple times, resulting in wildly inflated metrics. In addition, businesses need a single definition of metrics such as revenue, costs, and budgets to drive alignment and consistent reporting and analysis with a single version of truth.

Consider a university dataset where we want to report the total budget for the 'College of Engineering.'

Table: Department

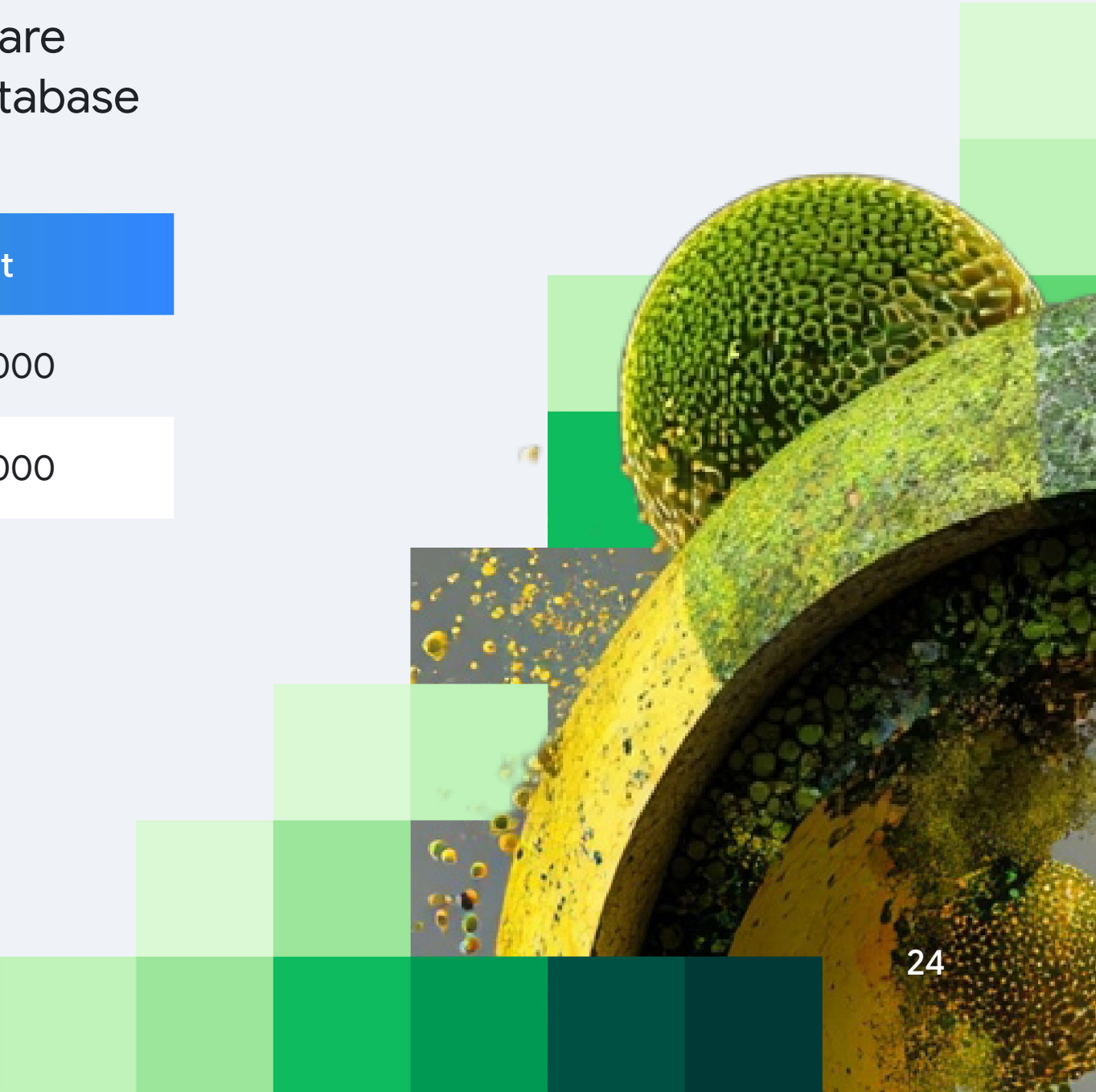
Dept_ID	Dept_ID	Dept_ID
1001	Computer Science	\$500,000

Table: Course

Course_ID	Course_Name	Dept_ID
1	Intro to CS	1001
2	Data Structures	1001

If you perform a standard `JOIN` to see which courses are offered by the Computer Science department, the database creates a flattened result set:

Dept_Name	Course_Name	Budget
Computer Science	Intro to CS	\$500,000
Computer Science	Data Structures	\$500,000



If you run `SELECT SUM(Budget)`, the database returns \$1,000,000 instead of the actual \$500,000. Because the "Computer Science" department appears in two rows, its budget is essentially 'fan-out' duplicated in your report.

BigQuery measures solve this overcounting and inconsistent metrics definition problem by embedding computational definition and aggregate logic for a metric – locked to a specific entity's primary key – directly into your graph schema, thereby providing a governed, single source of truth for business metrics. By defining the computational logic and aggregation at the schema level, AI agents and BI dashboards can query complex relationships with the guarantee that aggregations are performed exactly once per key and the entire organization operates from a single version of the truth.

Step 1: Defining a measure in your graph schema

A measure is defined within the `PROPERTIES` clause of a node or edge table using this exact format:

```
MEASURE (aggregate_function  
(property_reference))  
AS measure_alias.
```

The `MEASURE()` keyword wraps around a supported aggregate function (such as `SUM`, `AVG`, `COUNT`, `MIN`, or `MAX`) applied to a specific property.

Before defining a property graph, you need underlying relational tables that store your entities and their foreign key relationships.

In this example, we assume three existing base tables: `university.College`, `university.Department` (which references a `college_id`), and `university.Course` (which references a `dept_id`). Then, we define a property graph for a university. Notice how the `total_budget` measure is locked to the Department table's key (`dept_id`).



SQL

```
CREATE OR REPLACE PROPERTY GRAPH university.SchoolGraph
  NODE TABLES (
    university.College
      KEY(college_id)
      PROPERTIES(college_id, college_name),
    university.Department
      KEY(dept_id)
      PROPERTIES(
        dept_id,
        dept_name,
        college_id,
        budget OPTIONS(description="Department budget in
USD"),
        MEASURE(SUM(budget)) AS total_budget -- Locks
aggregation to dept_id
      ),
    university.Course
      KEY(course_id)
```

```
    PROPERTIES(
      course_id,
      course_name,
      dept_id,
      MEASURE(AVG(credits)) AS avg_credits,
      MEASURE(COUNT(course_id)) AS course_count
    )
  )
  EDGE TABLES (
    university.Department AS CollegeDept
      SOURCE KEY (college_id) REFERENCES College (college_
id)
      DESTINATION KEY (dept_id) REFERENCES Department
(dept_id),
    university.Course AS DeptCourse
      SOURCE KEY (dept_id) REFERENCES Department (dept_id)
      DESTINATION KEY (course_id) REFERENCES Course
(course_id)
  );
```


Step 2: Querying measures with GRAPH_EXPAND and AGG()

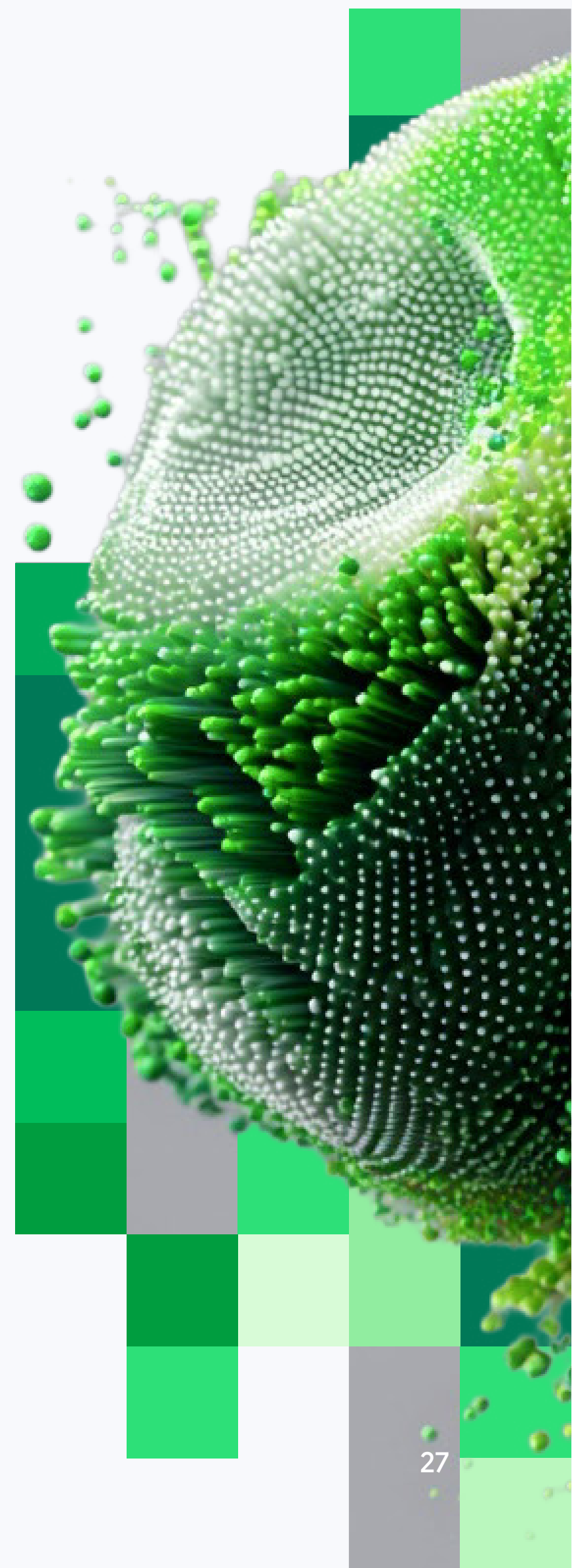
You cannot directly reference a property defined by a measure using a standard GQL traversal query. Instead, you access measures by calling the `GRAPH_EXPAND` table-valued function (TVF). `GRAPH_EXPAND` flattens your graph into a single table representation. You cannot directly `SELECT` a measure column from the flattened graph. To evaluate the measure, you must wrap it in the `AGG()` function.

This mandatory syntax tells the engine to execute the schema-defined aggregation exactly once per key, which inherently prevents data fan-out.

```
SQL
SELECT
  College_college_name,
  AGG(Department_total_budget) AS college_budget,
  AGG(Course_course_count) AS total_courses
FROM
  GRAPH_EXPAND("university.SchoolGraph")
GROUP BY
  College_college_name;
```

Why this matters

If a college has one department offering 10 courses, traditional SQL would duplicate the department's budget 10 times in the join. By using `GRAPH_EXPAND` combined with `AGG()`, BigQuery knows to calculate the department's budget exactly once based on the node's underlying key, returning the correct `college_budget` mathematically.



Best practices and schema constraints

To ensure accurate aggregations and successful **GRAPH_EXPAND** execution, follow these structural guidelines when modeling your graph:



Adhere to strict topology constraints:

Because **GRAPH_EXPAND** relies on a strict **LEFT JOIN** cascade, it does not accept all property graphs. Your schema must form a fully keyed, strictly acyclic hierarchy where every node is reachable from a single root node via many-to-one or one-to-one relationships moving outward.



Define nodes and edges within a single table:

Whenever possible, use the same table for a node definition and its connecting edge definitions. This enforces an exact 1:1 relationship, preventing the function from treating the edge as ambiguous.



Model many-to-many (M:N) relationships as root nodes:

If you have an intermediate junction table (for example, an Enrollment table connecting Students and Courses), do not model it solely as an edge. **GRAPH_EXPAND** will ignore ambiguous many-to-many edges. Instead, promote the junction table to a root node table, and use it to define two many-to-one (N:1) edge tables connecting outward to your dimension nodes.



Inspect your flattened schema:

Before writing aggregation queries, preview the exact column names, types, and descriptions of the flattened table using the system procedure:

```
CALL BQ.SHOW_GRAPH_EXPAND_SCHEMA('your_dataset.YourGraph', schema);
```

Practitioner's guide – high-impact graph use cases

4.1

Anti-money laundering and circular fraud prevention



(Try the step-by-step [tutorial](#).)



(Try the Spanner and BigQuery: [Real-Time Fraud Defense Shield](#).)

Challenge

Businesses face significant challenges from financial crimes like money laundering and circular fraud, leading to substantial financial losses, regulatory penalties, and reputational damage.

Combating money laundering is hampered by legacy systems that generate numerous false positives that burden compliance teams and obscure truly illicit activities within an estimated \$2 trillion global flow.

Meanwhile, circular fraud schemes are particularly difficult to detect because they involve complex, closed loops of transactions that, in isolation, appear legitimate and so require advanced analytical techniques, often leveraging graph-based analysis, to uncover the hidden networks and patterns of deception.



Solution

A modern data platform with native graph capabilities and vector search, like BigQuery, solves money laundering and circular fraud challenges by performing Graph Retrieval-Augmented Generation (GraphRAG). Traditional platforms perform keyword or purely semantic searches, missing complex multi-hop financial trails.

An integrated graph paradigm detects illicit activity via three mechanics:

01

Multi-hop traversal (zero-ETL):

Rather than duplicating data or performing resource-heavy relational JOIN operations, a property graph creates a semantic layer over structured data. It traverses cyclic or deep layers (for example, entity A -> entity B -> entity C -> entity A) to spot circular loops instantly.

02

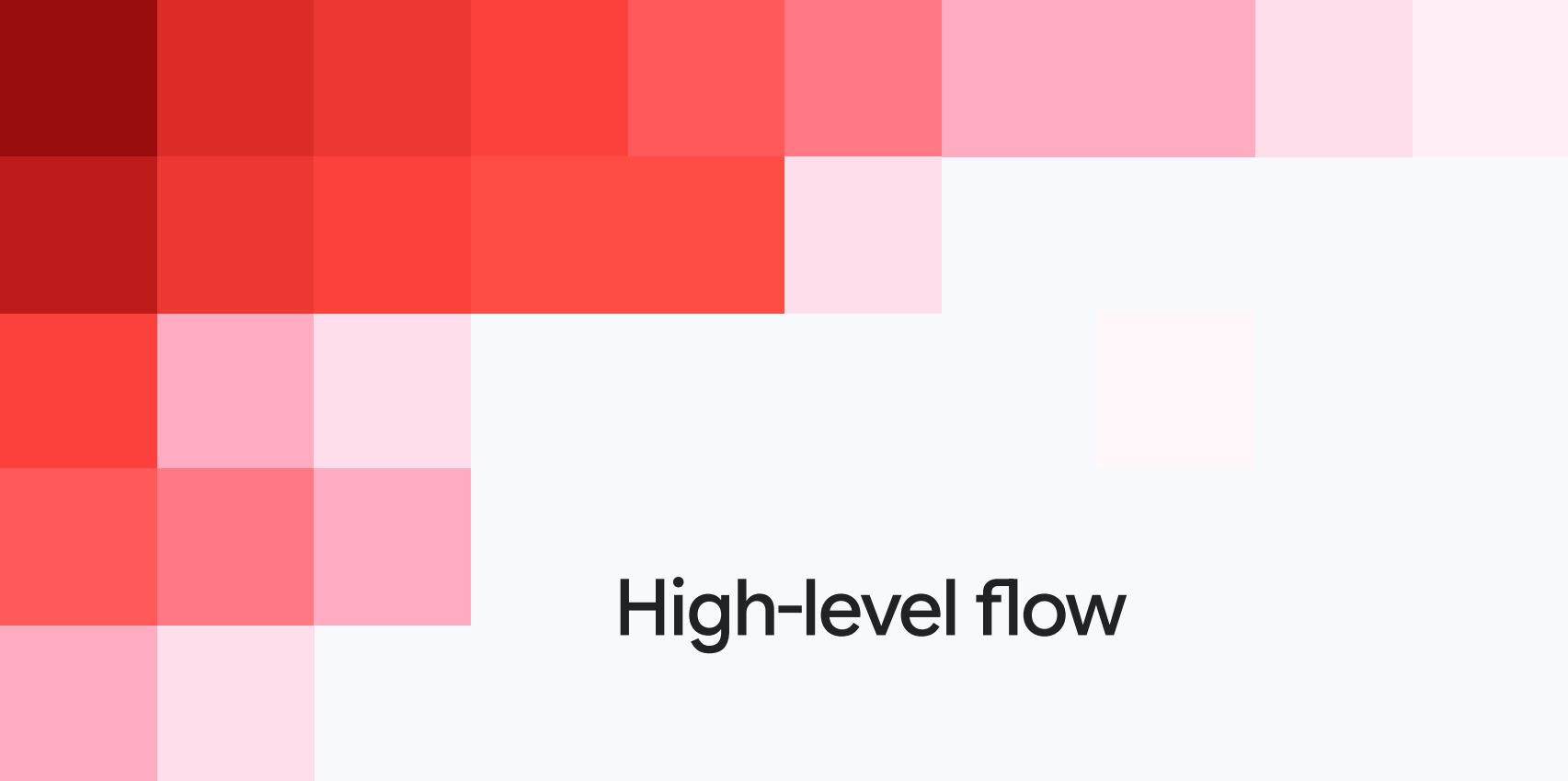
Hybrid retrieval (BigQuery Graph+ Vector Search):

Unstructured logs (for example, Know Your Customer (KYC) audit flags or suspicious notes) are converted into vector embeddings. Vector Search identifies the start node semantics while graph queries pull the topology of interconnected accounts.

03

False positive reduction:

By passing relationship structures and transaction magnitudes to LLMs, reasoning frameworks discern between high-risk cycles and normal behaviors (for example, small monetary gifts vs. high-volume shell routing).



High-level flow

Create a BigQuery dataset and schema to consolidate data across customers, accounts, and loans. Create unstructured data in Google Cloud Storage to track audits in files.

Create a property graph in BigQuery that connects these entities.

Visualize customers, accounts, and loans using graph queries to identify multi-hop relationships.

Generate vector embedding for audit logs stored in Google Cloud Storage. Run Vector Search and GQL traversal queries to pull the audit logs for every entity found in the path, providing the LLM with a 360° view of the syndicate.

Execute LLM chains to identify the fraudsters and eliminate false positives.



By implementing BigQuery Graph, we have successfully moved away from the previous limited SQL-based approach to a more scalable solution for fraud detection network analysis. This has allowed us to detect sophisticated fraud networks by uncovering hidden connections between seemingly unrelated accounts and transactions.

This transition from traditional relational queries to graph-based analytics has showcased measurable business impact with ~£9.1M of savings. This shift has not only improved the precision of fraud detection but has also provided a scalable foundation for protecting the ecosystem without adding significant operational overhead."

Francis Darby,
VP Data and ML,
Curve



Customer 360 and hyper-personalization



(Try the step-by-step [tutorial](#).)

Challenge

Businesses face a significant challenge in truly understanding and engaging with their customers on an individual level. Often, valuable customer data is scattered across numerous disparate systems, such as CRM, marketing platforms, sales records, and support logs, resulting in isolated data silos.

This fragmentation prevents organizations from building a comprehensive 'Customer 360' view, which is essential to grasping each customer's journey, preferences, and behaviors across all touchpoints. Without this unified perspective, companies struggle to provide consistent and relevant experiences, frequently leading to disjointed interactions and customer frustration. The absence of a holistic customer view directly impedes the ability to deliver effective hyper-personalization at scale. Generic marketing campaigns and one-size-fits-all sales approaches yield suboptimal results, as they fail to address the unique needs and contexts of individual customers.

Solution

A modern data platform, such as BigQuery powered by technologies like BigQuery Graph, AI, Vector Search, and multimodal data analytics, directly addresses the challenges of fragmented data and inconsistent customer experiences.

By acting as a central hub, it ingests and unifies customer data from all sources – CRMs, websites, mobile apps, marketing tools, and more – into a single, comprehensive 'Customer 360' profile. This unified view breaks down silos and provides a consistent, real-time understanding of each customer's interactions and preferences.

BigQuery Graph complements this by excelling at modeling and querying the complex relationships inherent in customer data.

It visualizes and analyzes the connections between customers, products, channels, and behaviors. For instance, it can easily identify customers who are influenced by other customers, or discover purchase patterns within specific communities.

High-level flow

Create a BigQuery dataset and schema to consolidate data across customers, products, orders, and stores to break down data silos.

Create a property graph in BigQuery that connects these entities.

Visualize customer purchase history using graph queries.

Build a product recommendation system using Vector Search and past purchase histories.

Enhance recommendations using 'bought together' graph relationships and Jaccard similarity.



Open Intelligence is our foundational intelligence layer that securely connects trillions of live data points from clients, partners and WPP in a privacy-first way and is now integrated and powers WPP's agentic marketing platform, WPP Open.

Enabled by Google Cloud's Spanner Graph, Open Intelligence is a significant advancement in AI-driven marketing and we are excited about extending the use case for analytical graph workloads on BigQuery Graph."

Rob Marshall,
Head of Strategy, Data and Intelligence
WPP

Supply chain impact analysis in food and beverages



(Try the step-by-step [tutorial](#).)

Challenge

Ensuring food safety and consistent quality is a paramount concern within the intricate and often opaque supply chains that restaurants depend on. Ingredients traverse a complex web of suppliers, processors, and distributors before reaching the kitchen, with each step introducing potential risks to food integrity.

This lack of end-to-end visibility and traceability across the network leads to a significant business challenge: without a unified view of where ingredients come from, how they've been handled, and the quality checks performed at each stage, restaurants are vulnerable to lapses in safety and inconsistencies in product quality.

This opacity makes it incredibly difficult to react swiftly and effectively when food safety issues or quality deviations occur.

Solution

A robust data platform is foundational to addressing the visibility and quality control challenges in restaurant supply chains. By integrating data from every node in the network – from farms and ranches to processors, distributors, and ultimately, individual restaurant kitchens – a comprehensive data repository can be built.

This platform consolidates diverse datasets, including supplier certifications, batch-level tracking, temperature logs during transport, quality assurance reports, and even sensor data.

Storing this information in a scalable system like BigQuery creates a single source of truth that replaces fragmented records and provides an end-to-end view of the entire food journey, which is crucial for proactive food safety and quality management.

Building on this unified data foundation, BigQuery Graph offers a powerful way to model the complex interdependencies within the supply chain.

By representing each entity – like a specific farm, a batch of chicken, a distribution center, or a prepared dish – as a node, and the relationships between them – such as "supplied," "shipped via," or "is an ingredient in" – as edges, a dynamic graph network emerges.

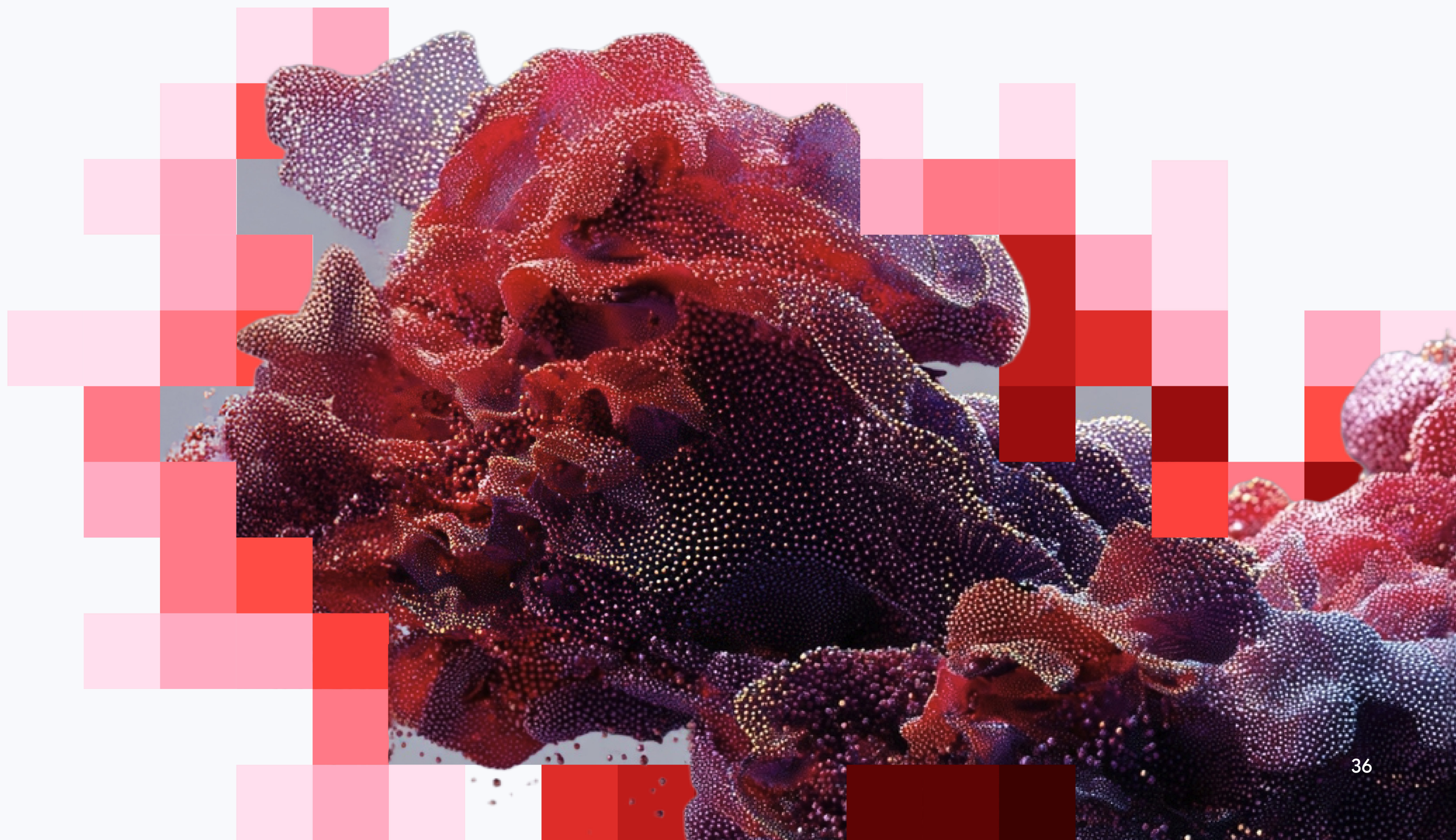
This graph structure allows for rapid and intuitive traversal. For example, in the event of a food safety incident, graph queries can instantly trace a contaminated ingredient back to its source or forward to identify all affected menu items and restaurant locations, enabling surgical recalls and minimizing the blast radius far more efficiently than traditional relational queries.

High-level flow

Define a graph model on top of existing BigQuery tables with entities such as items (such as tomatoes or chicken), facilities (such as suppliers, distribution centers, or cafes), and bill of materials (which defines weight relations, for example item A goes into item B).

Create a property graph inside BigQuery to model the entities and relationships, for example 'item is stored_at facility.'

Run traversal queries to trace upstream and downstream impacts.



Healthcare: metabolic pathway search



(Try the step-by-step [tutorial](#).)

Challenge

In oncology and drug discovery, identifying effective therapies requires understanding complex tumor microenvironments where cancer thrives in low-oxygen conditions that resist treatments and immune responses. Mapping these mechanisms in-silico is extremely challenging because tumor metabolism functions as an intricate network with a massive combinatorial explosion of connected chemicals and enabling enzymes. Legacy databases and traditional raw SQL approaches struggle to navigate these deep relationships, often failing or timing out due to complex recursive joins across millions of potential pathways. Consequently, critical biochemical routes remain obscured, delaying the identification of paths to making tumors vulnerable to treatment and repurpose existing drugs effectively.

Solution

A modern data platform like BigQuery with native graph capabilities solves complex metabolic modeling challenges through an agent-driven Graph Retrieval-Augmented Generation (GraphRAG) workflow.

Rather than forcing researchers to write complex query logic, scientists can formulate a natural language hypothesis.

An autonomous AI agent then translates this theory into precise graph queries using GQL to interrogate the underlying graph topology and factual data and verify whether the biological pathway exists.

This integrated paradigm accelerates drug discovery via three mechanics:

01

Agentic query translation:

A scientist constructs a biological hypothesis (For example, "Can suppressing enzyme X sever the metabolic supply chain between L-tryptophan and indoxyl sulfate?").

An AI agent uses vector search to map the unstructured medical concepts to exact graph entities, automatically generating the optimized GQL syntax required to test the theory.

02

Multi-hop traversal (zero-ETL):

The agent-generated query executes directly against a semantic property graph layered over structured chemical and reaction tables natively within BigQuery.

It instantly traverses deep, multi-hop pathways across thousands of nodes and edges to check the validity of the hypothesized route without moving data.

03

Hypothesis validation and false positive reduction:

The retrieval framework returns the exact factual pathway structures to the LLM agent. By evaluating the returned graph topology against specific organ environments or biological constraints, the agent confirms or refutes the scientist's hypothesis, filtering out dead ends and grounding the final therapeutic target in verified warehouse data.

High-level flow

Establish zero-copy access:

Securely store unstructured clinical research, trial protocols, and medical literature (such as PDFs or TXTs) in Google Cloud Storage, exposing these directly to BigQuery using **object tables** so the warehouse can access files natively, instead of duplicating data.

Native parsing and generative extraction:

Use the native **AI.PARSE_DOCUMENT** function directly within standard SQL queries to extract chunks and parse complex document layouts. Leverage **AI.GENERATE** (calling Gemini models via remote connections) inside **SELECT** statements to extract structured entities – like chemicals, target enzymes, and mechanisms of action – directly into denormalized tables.

Construct the property graph:

Build a BigQuery property graph over the unified structured and extracted datasets to map the complex biological ecosystem (defining distinct nodes for drugs, trials, disorders, and enzymes connected by directed multi-hop relationships).

Embed and map via native search:

Use the **AI.EMBED** function to automatically generate high-dimensional vector representations of unstructured text. When a scientist submits a natural language hypothesis, an AI agent runs **AI.SEARCH** to resolve abstract medical concepts directly to specific graph nodes using hybrid retrieval.

Agentic GQL execution and validation:

The agent dynamically compiles and executes a GQL traversal query to pull the exact multi-hop topology connecting the targeted entities. By passing this comprehensive biological map back to the reasoning framework, the agent validates or refutes the hypothesis natively inside the warehouse.



Understanding disease isn't about collecting more data; it's about understanding the relationships within that data. By leveraging pathway search in BigQuery Graph at a massive scale, reaching depths of 7+ hops, we are finally able to see more of the human metabolism map.

This level of scale is what allows us to move beyond trial and error, identifying the precise biological levers that need to be pulled to cure complex diseases. We aren't just guessing anymore, we're emulating life at the speed of compute."

Nik Sharma,
CEO,
BioCortex

biocortex

4.5

Cybersecurity: real-time threat correlation and lateral movement hunting



(Try the step-by-step [tutorial](#).)

Challenge

Cybersecurity companies process massive volumes of security telemetry across distributed systems to detect, correlate, and respond to threats in near real time.

Because security alerts, incidents, and infrastructure logs are inherently connected across users, devices, IPs, and applications, attackers that breach a network perimeter rarely stay in one place; they stealthily hop from one server to another using compromised credentials – a technique known as lateral movement.

Because each individual hop can look entirely benign on its own (for example, a single login log), traditional flat security logging fails to trigger an alarm.

This is solved by stitching these isolated incidents together into an incredibly complex web at a multi-tenant, terabyte scale, but this causes severe operational bottlenecks when using traditional relational databases or legacy standalone graph databases:

01

The latency trap

Complex multi-hop queries in native pointer-based architectures exhibit severe tail-latency, rendering real-time threat correlation impossible.

02

Resource exhaustion

At TB-scale, standalone graph engines suffer from skyrocketing CPU utilization, degraded efficiency, and heavy infrastructure management overhead.

By moving to a unified graph architecture on the Agentic Data Cloud, enterprises transition from a flat, tabular log layout to an operational, globally distributed, and strongly consistent graph model.

Spanner Graph serves as the real-time operational engine, explicitly modeling node and edge structures into relational tables to replace native pointer traversals with high-performance table joins. Relationships are materialized and indexed for specific multi-hop patterns, dropping computation latency to milliseconds.

Nodes (entities):

User, Device, IPAddress, Server, and SecurityAlert.

Edges (relationships):

LOGS_IN_TO, ACCESS_REQUESTED, CORRELATES_WITH, and DEPENDS_ON.

[User Node] ---->
(LOGS_IN_TO) ---->
[Device Node] ----> (ACCESS_REQUESTED)
----> [Server Node]

Flag anomaly:

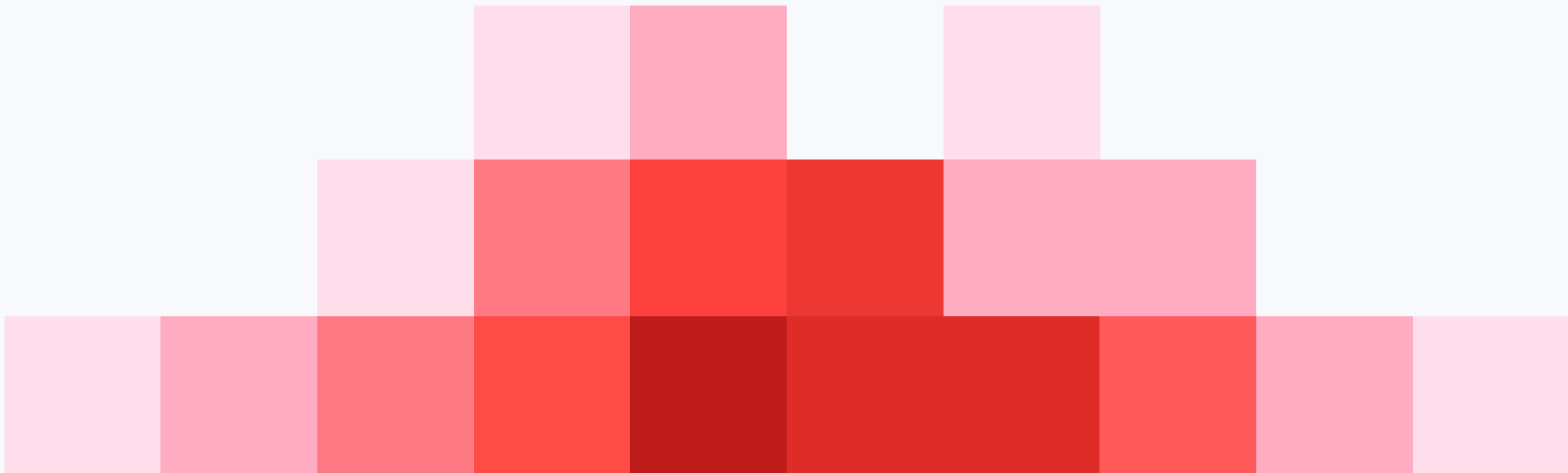
A User Behavior Analytics (UBA) tool or an anomaly detection model flags an unusual login event (for example, an engineer logging in from an unfamiliar IP address at 3:00 AM).

Map blast radius (Spanner Graph):

Instead of sweeping flat text logs sequentially, analysts execute an optimized GQL query using standard, native query constructs. Spanner Graph utilizes highly specific indices aligned with the multi-hop traversal pattern to pull the entire blast radius in tens of milliseconds. This evaluation untangles exactly what other devices the identity touched, whether secondary devices hit high-value assets, and which associated user accounts are exposed.

Isolate the breach:

Security Orchestration, Automation, and Response (SOAR) tools ingest the low-latency graph traversal output to automatically freeze the compromised network path, resetting credentials and isolating affected servers.





Historical deep-dive (BigQuery Graph):

Simultaneously, Change Data Capture pipelines stream real-time telemetry from Spanner into BigQuery Graph without impacting the transactional workload performance. BigQuery Graph runs large-scale analytical workloads over petabytes of historical logs to expose complex, long-term fraud rings or systemic security syndicates that engineered the initial breach.

4.6

Competitive intelligence: unstructured data to knowledge graphs (SEC filings)



(Try the step-by-step [tutorial](#).)

Challenge

In corporate strategy and investment research, analyzing a company's market position requires extracting critical signals from massive, dense unstructured documents like annual SEC 10-K filings.

Uncovering complex competitive landscapes is extremely challenging because vital insights regarding market moves, emerging risks, and direct competitors are buried deep within unstructured text.

Legacy analytics pipelines struggle to manage this process efficiently, typically requiring a highly fragmented chain: raw object storage, bespoke parsing services, separate AI extraction layers, standalone graph databases, and disconnected business intelligence tools.

Consequently, strategy teams face ongoing maintenance friction, expensive data duplication, synchronization overhead, and multiple points of failure that leave valuable corporate intelligence trapped in silos.

Solution

A modern data platform like BigQuery combined with integrated visualization tools solves complex unstructured analytics challenges by creating an evidence-first knowledge graph pipeline directly inside the data warehouse. Traditional pipelines require constant data movement between specialized systems to connect text extractions to graph topologies.

An integrated warehouse paradigm unlocks unstructured corporate intelligence via three mechanics:

01

Unified AI inference and extraction:

Utilizing native SQL functions, like **AI.GENERATE_TEXT** powered by Gemini models, the warehouse processes highly targeted unstructured document sections directly where they reside. It reliably extracts structured entities (such as risks, market actions, and competitors) and outputs clean JSON arrays without requiring external orchestration.

02

DDL graph declaration (zero data movement):

Structured extractions are saved into standard node and edge tables, which are subsequently bound into a fully traversable property graph using a single Data Definition Language (DDL) statement (**CREATE PROPERTY GRAPH**). This establishes direct relational mapping without complex JOIN operations or exporting data to standalone graph engines.

03

Evidence-first traceability and exploration:

Every generated node and relationship preserves essential source metadata, grounding extracted elements with exact evidence text and direct URLs back to the initial filing document. Low-code graphical environments like **Kineviz GraphXR** connect natively to the BigQuery Graph, empowering analysts to visually navigate subgraphs, verify multi-hop relationships, and trace strategic insights back to their original context interactively.



High-level flow

Ingest and parse:

Retrieve raw SEC 10-K filings from EDGAR, transform the source formatting into clean Markdown while preserving hierarchical structure, and load the text directly into BigQuery via Google Cloud Storage.

Targeted signal filtration:

Isolate text processing to high-signal regulatory sections – specifically the Business, Risk Factors, and Management's Discussion and Analysis (MD&A) sections – preserving core identifiers like CIK, company name, filing year, and direct source URLs.

Native generative extraction:

Execute standard SQL queries calling Gemini models to extract structured entities and competitive mentions from the filtered text blocks, resolving tens of thousands of raw mentions into normalized company nodes.

Declare the graph:

Use native BigQuery DDL to declare a unified property graph over the extracted node tables (companies, competitors, risks, opportunities) and directional edge tables (**COMPETES_WITH**, **FACES_RISK**).

Interactive visual exploration:

Connect third-party visualization platforms like Kineviz GraphXR directly to the BigQuery graph layer, enabling compliance and strategy teams to self-serve deep visual analyses of the competitive landscape without writing custom query logic.

4.7

Network IT operations: root-cause analysis and autonomous network digital twins



([Click here](#) to learn more.)

Challenge

Modern telecommunications networks and enterprise IT infrastructures are massive, multi-layered webs of physical and logical dependencies spanning Radio Access Networks (RAN), transport backhaul, core networks, and distributed cloud applications. When a service degradation or localized outage occurs, finding the true culprit is a massive 'needle in a haystack' problem.

Legacy performance management systems fail because they treat these highly interconnected systems as flat, disconnected tabular entries.

This creates a dual-pronged problem for large enterprises:



01

Siloed topologies and alarm fatigue:

Splitting live topology and telemetry into flat relational tables requires multi-way SQL joins across dozens of massive tables. When a single physical asset (for example, a physical fiber line or interface card) fails, it triggers a massive cascade of downstream logical alerts, blinding network operations centers with alert fatigue due to the lack of a unified graph perspective.

02

The silent degradation trap:

Traditional tabular baselines rely on static threshold alerts for isolated nodes, completely missing structurally propagating degradations that bottleneck traffic several hops away.

03

High Mean Time to Resolution (MTTR):

Without real-time topological mapping, human operators spend hours manually tracing connection hops, resulting in a high MTTR of up to 5 hours as well as costly SLA violations.

Solution

The Agentic Data Cloud solution builds a graph-first network digital twin that continuously updates with real-time data from the physical network, supported by an end-to-end operational and analytical loop.

Nodes (entities):

NetworkElement (routers, cell towers), **LogicalInterface**, **Location**, and **CustomerService**.

Edges (relationships):

DEPENDS_ON, **CONNECTED_TO**, and **ROUTES_THROUGH**.

By leveraging Spanner Graph as the transactional, globally consistent core, the network digital twin handles massive high-throughput read/write workloads with 'five nines' (99.999%) availability, ensuring the operational network view is never stale or lagged.

High-level flow

Ingest and map relationships: Live network events, SNMP traps, and real-time gNMI telemetry stream directly into the transactional Spanner Graph to maintain the live state of the digital twin at millisecond accuracy. Concurrently, massive long-term performance logs and counters are saved in open Apache Iceberg formats.

Model and train: BigQuery Graph queries the Iceberg lakehouse data in-place to build a deep historical baseline without complex ETL pipelines. Large-scale graph algorithms (such as PageRank and community detection) are run over petabytes of structural network metadata to train graph neural networks that learn the structural laws of fault propagation.

Detect and trace (Spanner Graph): When a service-layer drop occurs, operator teams or automated monitoring tools execute low-latency GQL queries to traverse multi-hop physical paths in milliseconds, tracing upstream **DEPENDS_ON** dependency edges directly from the degraded customer service endpoint back to the root hardware interface.

Enrich graph nodes: Built-in BigQuery AI functions run Vector Search and clustering on unstructured syslog events directly over the localized root-cause node, instantly matching active node anomalies to historical graph resolution paths.

Remediate and self-heal: The exact identified graph path and blast-radius analysis are passed to a Gemini-powered autonomous network agent. Grounded in the semantic graph structure, this agent runs a code interpreter to calculate safe failover routing and triggers an automated webhook to self-heal the network. This collapses root-cause analysis and remediation time from 5 hours to between 1 and 5 minutes.

Customer Identity Resolution (Entity Matching)



(Try the step-by-step [tutorial](#).)



(Try the Spanner [IAM Graph example](#).)

Challenge

In modern enterprise environments, customer identity resolution is a foundational capability for building a Customer 360 view, detecting fraud, and consolidating multi-system data. However, accurately connecting duplicate records across disparate datasets is extremely difficult due to noisy real-world data, including phonetic variations, spelling typos, and missing values.

This creates a dual-pronged problem for large enterprises:

01

Fragile Rule-Based Matching:

Traditional entity matching relies heavily on strict deterministic rules (e.g., matching exact SSNs or names). These rigid rules fail when confronted with complex typos, OCR errors, or semantic differences, resulting in missed links (under-merging) and fragmented customer profiles.

02

Computational Bottlenecks ($O(N^2)$):

Evaluating all possible customer record pairs to find matches results in quadratic computational growth. Comparing millions of records against one another without smart blocking and efficient clustering techniques becomes computationally prohibitive at scale.

Solution

A modular, end-to-end Customer Identity Resolution engine built directly inside BigQuery. It combines rule-based fuzzy matching (Soundex, Levenshtein distance) with semantic vector search (AI. EMBED), and utilizes BigQuery Property Graphs with ISO GQL path traversals to accurately cluster customer records.

```
[Customer Node] ---->
(MATCHED_TO) ---->
[Customer Node]
```

Nodes (entities):

Customer profiles (users)

Edges (relationships):

MATCHED_TO (representing a hybrid rule-based and vector similarity match between two customer records).

Key schema transformations and ingestion:

Instead of duplicating data or relying on external graph databases, BigQuery creates a logical Property Graph directly over your relational tables.

```
CREATE OR REPLACE PROPERTY GRAPH `identity_resolution.customer_
identity_graph`
  NODE TABLES (
    `identity_resolution.customer_nodes_cleaned` AS `Customer`
    KEY (rec_id)
  )
  EDGE TABLES (
    `identity_resolution.final_matched_edges`
    KEY (source_id, target_id)
    SOURCE KEY (source_id) REFERENCES `Customer`(rec_id)
    DESTINATION KEY (target_id) REFERENCES `Customer`(rec_id)
    LABEL MATCHED_TO
  );
```


01

Feature Engineering & Candidate Generation:

Raw customer data is ingested and preprocessed. A Cloud Function is deployed as a BigQuery Remote UDF to normalize street addresses. The pipeline then computes SOUNDEX phonetic keys and uses BigQuery's built-in generative AI functions (AI.EMBED) to generate text embeddings of the customer profiles by offloading the compute to Vertex AI.

02

Hybrid Edge Scoring & Feature Fusion:

To avoid quadratic scaling, the engine applies blocking rules (e.g., records sharing a SSN or DOB + Soundex key) to generate candidate pairs. It fuses rule-based match scores (using **EDIT_DISTANCE** and Jaccard similarity) with semantic similarity scores generated by **VECTOR_SEARCH**. The combined results are saved into a unified candidate edge table.

03

Property Graph Construction & GQL Clustering:

The unified edge table and cleaned nodes are used to construct the BigQuery Property Graph. To group multiple duplicates into a single canonical master ID, the engine executes an ISO GQL query that traverses variable-length paths:

```
GRAPH_TABLE(  
  `identity_resolution.customer_identity_graph`  
  MATCH (c1:Customer)-[e:MATCHED_TO]->{1, 2}(c2:Customer)  
  COLUMNS[...](asc_slot://start-slot-13) (c1.rec_id AS source_node_id,  
    c2.rec_id AS target_node_id)  
)
```

By specifying {1, 2}, the engine performs a k-hop path traversal to easily resolve transitive connections (e.g., if A matches B, and B matches C, A is clustered with C) into resolved canonical customer clusters.

04

Graph-Based Householding with Adamic-Adar:

Beyond resolving individual duplicate identities, the graph expands its scope to "householding"—intelligently grouping distinct individuals (e.g., spouses, dependents) who share a living situation or family unit. By applying node similarity and link prediction algorithms like the Adamic-Adar index, the engine evaluates the proportion and rarity of common neighbors between nodes. For instance, two distinct individuals sharing a highly specific physical address or a rare surname (rare shared neighbors) receive a much higher Adamic-Adar score than two people merely living in the same zip code (a common shared neighbor). This allows the system to confidently infer and link household relationships for unified household-level analytics, risk assessment, and marketing. (Note: clustering can be further enhanced by hierarchical clustering algorithms such as louvain).

05

Incremental Resolution & Persistent Stability:

For real-world production, the engine doesn't re-run the full graph resolution every day. Daily batch intakes are processed via incremental delta matching that compares new incoming records against the existing baseline clusters, either assigning them an existing `canonical_customer_id` or creating a new customer entity on the fly.

4.9 Social media: influencer hub detection

Challenge

As more marketing budgets shift towards social, identifying the best content creators for a specific brand or campaign becomes a highly complex process. While agencies possess immense historical data covering past brand collaborations, creator categories, and campaign metrics, discovering net-new or 'lookalike' influencers remains a manual, imprecise process.

Legacy workflows rely on rigid keyword searches and audience demographics, completely missing topical nuances and complex relationship networks.

Consequently, strategy teams struggle to find relevant creators who organically post about a brand's specific sub-category or align with highly targeted creative concepts (for example, "Best barefoot-style trail shoes").

Without robust audience data or advanced mapping techniques, agencies cannot efficiently traverse creator ecosystems to spot adjacent influencers who cover similar products or share overlapping campaign events.

Furthermore, onboarding new creators carries inherent brand risk. Manually vetting years of backlogged posts and videos for controversial past statements, toxic sentiment, or conflicting brand sponsorships is virtually impossible at scale, leaving campaigns vulnerable to sudden PR crises and reputational damage.

A modern data platform like BigQuery combined with multi-modal AI unlocks an intelligent influencer discovery hub directly within the data warehouse. By combining historical brand context with deep content analysis, agencies can better understand how creators are interacting with or representing brands in an organic way. Rather than relying solely on explicit past sponsorships, the platform allows teams to traverse the many different ways creators could be related through subtle signals extracted directly from their multimodal social media content.

An integrated graph and vector paradigm streamlines creator discovery and risk mitigation via three mechanics:

01

Multi-modal metadata extraction:

Utilizing native generative AI techniques, incoming daily and monthly batches of creator images and videos are automatically analyzed directly within the pipeline. Deep visual and textual metadata – such as background product placements, spoken sentiment, visual aesthetics, and brand logos appearing organically in videos – are extracted and converted into high-dimensional vector embeddings stored in a vector database.

02

Hybrid search discovery:

Rather than relying solely on exact hashtag or brand name matches, strategy teams execute hybrid search queries combining direct keyword relevance with abstract semantic meaning. This allows the platform to instantly return highly ranked creator profiles for granular, conceptual briefs while filtering results by category or past brand affinity.

03

Multi-hop traversal and automated safety scoring:

By modeling profiles, campaigns, and extracted content metadata as a property graph, the system maps the broader creator ecosystem. Graph queries effortlessly traverse multi-hop paths to connect lookalike creators based on shared organic behaviors. Simultaneously, the framework evaluates the last three years of extracted content history and network proximities to generate an automated **brand safety score**. This instantly flags toxic sentiment, controversial past brand interactions, or risky network associations before any outreach occurs.

High-level flow

Ingest multi-modal batches:

Securely ingest continuous daily chunks and monthly batches of influencer profiles, posts, images, and videos into Google Cloud Storage, formatting the profile and brand context data into BigQuery tables optimized for querying and dashboarding.

AI video and image analysis:

Apply native generative AI techniques to extract descriptive metadata, embedded text, organic brand appearances, and deep semantic concepts directly from the visual media assets.

Generate vector embeddings:

Compile vector embeddings from the extracted visual metadata alongside profile and post text, storing them natively in BigQuery Vector Search to index the true semantic meaning of creator content.

Declare the creator graph:

Build a BigQuery property graph mapping historical brand relationships, past campaign events, product categories, and newly extracted multimodal content links to establish an interconnected influencer topology.

Execute hybrid discovery, similarity, and vetting:

When discovery teams query the database for custom lookalikes, the engine merges semantic hybrid search results with multi-hop graph traversals. It applies node similarity algorithms directly within the query logic – utilizing metrics like Jaccard similarity to evaluate shared categories and hashtags, or Adamic-Adar to weight rare, highly specific common connections between creators. Finally, it dynamically evaluates the last three years of profile history to compute a real-time brand safety score – filtering out high-risk creators – and returns the fully vetted candidates directly to the internal brand AI platform.

Ready to take your next steps?

To get started on your graph journey:

 Learn more about
Spanner Graph

 Learn more about
BigQuery Graph

Contact us for expert advice

Contributors

Manan Goel, Group Product Manager, Google Cloud	Michelle Liu, Custom Engineer, Google Cloud	Neha Bhatnagar, Custom Engineer, Google Cloud	Candice Chen, Product Manager, Google Cloud
Gopikrishnan Anilkumar, Senior Product Manager, Google Cloud	Guru Rangavittal, Cloud Transformation Technical Lead, Google Cloud	David Ng, Cloud Database Engineer, Google Cloud	Greg Cheng, Staff Technical Solutions Consultant, Google Cloud
Bei Li, UTL for Graph (BigQuery, Spanner), Google Cloud	Piyush Mathur, Group Product Manager, Google Cloud	Shan Kulandaivel, Group Product Manager, Google Cloud	Deepak Dayama, Group Product Manager, Google Cloud
Geeta Banda, Head of Outbound Product Management, Google Cloud	Manoj Gunti, Product Marketing Manager, Google Cloud		