Database Management System: Google BigQuery

Description:

Google BigQuery is a fully managed, cloud-based, and serverless data warehouse and analytical database management system developed by Google Cloud. It is designed for high-performance, scalable, and cost-effective data analysis and querying, making it particularly well-suited for data warehousing and analytics tasks.

How Google BigQuery Different from Other Databases?

Google BigQuery is distinct from traditional relational databases and other data warehousing solutions in several ways:

- **Serverless and Scalable:** Unlike traditional databases that require hardware and capacity planning, BigQuery is serverless and automatically scales to handle query workloads.
- **Pay-as-You-Go:** BigQuery's pricing model is based on data processed, which is different from traditional databases that often involve upfront hardware and software costs.
- **Separation of Storage and Compute:** BigQuery stores data separately from computational resources, which enhances scalability and minimizes data movement.
- **Designed for Analytical Workloads:** BigQuery is optimized for analytical and data warehousing tasks, making it an excellent choice for organizations with heavy data analysis needs.
- **Real-Time Streaming:** BigQuery supports real-time data streaming, a feature not commonly found in traditional databases.
- **Integration with Google Cloud:** It seamlessly integrates with other Google Cloud services, allowing for a comprehensive cloud-based data ecosystem.
- **Ease of Use:** BigQuery is designed for ease of use and offers a user-friendly interface, making it accessible to both data engineers and data analysts.

Data Source Information:

- **Data Origin:** This dataset was originally sourced from KDNuggets and is available on Kaggle at the following link: Sales and Customer Data on Kaggle. It is a publicly available dataset that contains sales and customer information, making it suitable for various analytical and research purposes.

Database Schema: Sales and Customer Data

Sales Table

Column Name	Data Type	Description
invoice_no	STRING	A unique identification number for each sales invoice
customer_id	STRING	ID of the customer associated with the transaction
category	STRING	General item category for the product sold
quantity	INTEGER	Number of products or items sold
price	FLOAT	Price of each individual product
invoice_date	DATE	Date of the purchase
shopping_mall	STRING	Location or name of the mall where the transaction took place

Customer Table

Column Name	Data Type	Description
customer_id	STRING	A unique identification number for each customer
gender	STRING	Gender of the customer (e.g., "Male", "Female")
age	INTEGER	Age of the customer
payment_method	STRING	Payment method used by the customer during transactions

Why This Schema and Database is Suitable for Analytical Databases (Google BigQuery):

- **Structured Data for Analysis:** The schema is structured and well-organized, aligning with Google BigQuery's capabilities for handling structured data efficiently.
- **SQL Compatibility:** Google BigQuery supports SQL, the standard language for querying and analyzing structured data. The schema is SQL-friendly, enabling complex queries for insightful analysis.
- **Data Integration:** The schema combines sales and customer data, showcasing its ability to handle data integration and support comprehensive analysis by joining datasets.

- **Performance and Scalability:** Google BigQuery offers high performance and scalability for analyzing large datasets, making it well-suited for this schema.
- **Data Visualization and Reporting:** The structured data can be used with data visualization and reporting tools for creating meaningful reports and visualizations.
- **Query Performance Optimization:** Google BigQuery provides features like caching, partitioning, and clustering to optimize query performance, enhancing the schema's effectiveness when working with substantial datasets.
- **Collaboration and Sharing:** Google BigQuery facilitates collaboration and sharing among users, making it a suitable platform for teams working on data analysis projects using this schema.

Database Schema Data Structures: Sales and Customer Data

Sales Table (sales_data):

- **invoice_no (STRING):** This field serves as the unique identification number for each sales invoice. It provides a reference point for associating products with specific transactions.
- **customer_id (STRING):** The customer identification number is stored in this field, linking each sale to a specific customer. This facilitates customer-related analytics and segmentation.
- **category (STRING):** This field categorizes products into groups based on general item categorization. It aids in product segmentation and analysis.
- **quantity (INTEGER):** The number of products or items sold in each transaction is recorded in this field. It's crucial for calculating total quantities and performing item-level analysis.
- **price (FLOAT):** The price of each product is stored in this field. It is essential for calculating total sales amounts and performing price-related analyses.
- **invoice_date (DATE):** The date of the purchase transaction is captured in this field. It allows for time-based analysis, including sales trends, seasonality, and historical insights.
- **shopping_mall (STRING):** The location or name of the shopping mall where the transaction occurred is stored here. It's useful for location-based analysis and understanding which malls are generating the most sales.

Customer Table (customer_data):

- **customer_id (STRING):** This field serves as a unique identification number for each customer, allowing for individual customer profiling and linking customer data with sales transactions.
- **gender (STRING):** The customer's gender is recorded in this field, helping to segment customers by gender and analyze gender-based preferences.

- **age (INTEGER):** The customer's age is stored here, facilitating age-related analysis and segmentation.
- **payment_method (STRING):** The payment method used by the customer during transactions is recorded in this field. It's essential for understanding payment preferences and trends.

These data structures form the foundation for analytical work in the database schema. The Sales Table captures detailed transaction information, while the Customer Table stores customer attributes. The combination of these tables allows for comprehensive analysis, including customer segmentation, sales trends, and product-related insights. The structured data types (STRING, INTEGER, FLOAT, DATE) used in the schema enable efficient data analysis and querying.

Data structures that are extensively used in Google BigQuery:

- **Databases and Projects:** BigQuery uses a project-based organization. Projects are containers for datasets, which can be thought of as equivalent to databases. You can have multiple datasets within a project, each serving different purposes or containing different sets of data.
- **Datasets:** Datasets in BigQuery are used to group related tables and views. They provide a way to organize and manage data within a project. Datasets do not have schemas themselves; schemas are defined at the table level.
- **Tables:** Tables are the core data structure in BigQuery. Each table represents a collection of structured data, and it has a schema that defines the fields (columns) and their data types. BigQuery tables are typically used for storing analytical data.
- **Partitions:** BigQuery supports partitioned tables, which can significantly improve query performance and reduce costs for certain types of queries. You can partition tables by date or timestamp, and queries can be more efficient when filtering data within a specific date range.
- **Nested and Repeated Fields:** BigQuery supports nested and repeated fields within tables. Nested fields allow you to have hierarchical data structures within a column, while repeated fields let you store arrays of values within a column. This is useful for dealing with semi-structured data.
- **Materialized Views (BI Engine):** BigQuery supports materialized views for caching and accelerating query performance. The BigQuery BI Engine can be used to create and manage these views.
- **User-Defined Functions (UDFs):** You can create user-defined functions in JavaScript, SQL, or Python to perform custom transformations and calculations within your SQL queries.
- **Table Decorators:** Table decorators allow you to query a table as it existed at a specific point in time, making it easy to analyze historical data.

ETL process

The interface we utilized is Google BigQuery, where we managed two tables: "customers" and "sales."

We initiated the process by importing these tables into the tool. To do so, we simply uploaded the respective CSV files, and Google BigQuery's auto-schema detection feature came in handy. This means we didn't need to manually specify data types for each column; the tool automatically recognized and assigned them.

Our imported tables were denoted as "Customer staging" and "Sales staging" for your reference. Our primary objective was to denormalize the data for more efficient querying and analytical purposes.

To achieve this, we proceeded to create a final table named "customer sales data." We used an SQL query for this purpose and then employed an INSERT statement to load the data into this final table. This involved joining the "customers" and "sales" staging tables.

Subsequently, we harnessed SQL queries to answer various business questions. For example, we utilized a SQL code to identify the most sold category. Notably, BigQuery employs standard SQL language, making it quite familiar and accessible.

SQL to create final table –

```
CREATE TABLE `customer-sales-404221.CUSTOMER_SALES.customer_sales_data` (

 customer_id STRING,

 category STRING,

 quantity INT64,

 price FLOAT64,

 total_price FLOAT64,

 invoice_date DATE,

 shopping_mall STRING,

 gender STRING,

 age INT64,

 payment_method STRING

);
```

SQL to insert data into final table –

```
INSERT INTO `customer-sales-404221.CUSTOMER_SALES.customer_sales_data` (
 customer_id,
 category,
 quantity,
 price,
 total_price,
 invoice_date,
 shopping_mall,
 gender,
 age,
 payment_method)

SELECT
 s.customer_id AS customer_id,
 s.category AS category,
 s.quantity AS quantity,
 s.price AS price,
 s.quantity * s.price AS total_price,
 s.invoice_date AS invoice_date,
 s.shopping_mall AS shopping_mall,
 c.gender AS gender,
 c.age AS age,
 c.payment_method AS payment_method
```

FROM `customer-sales-404221.CUSTOMER_SALES.sales_staging` AS s

INNER JOIN `customer-sales-404221.CUSTOMER_SALES.customer_staging` AS c

ON c.customer_id = s.customer_id;

SQL - What is the most popular product category in terms of sales?

SELECT SUM (quantity) AS total_quantity, category

FROM customer-sales-404221.CUSTOMER_SALES.customer_sales_data

GROUP BY category

ORDER BY total_quantity DESC.

SQL - What is the total revenue generated in the year 2023?

SELECT SUM(total_price) AS total_revenue

FROM customer-sales-404221.CUSTOMER_SALES.customer_sales_data

WHERE EXTRACT(year FROM invoice_date) = 2023;

Running these queries yields results, which can be conveniently visualized by creating graphs with a single click. By clicking on the "chart preview" icon, we can access the query results in a visually informative format. Moreover, we have the option to save these results in CSV, JSON, or Google Sheets for further analysis.

For deeper data exploration and analysis, you can employ tools like Google Sheets, Looker Studio, or Python. In fact, if we intend to utilize machine learning algorithms or conduct more advanced data analysis, we can effortlessly redirect the data to a Python notebook using Google Colab.

In summary, this encapsulates the ETL process we executed for our project.