

GLOBAL GADGETS DATABASE PROJECT

A level 7 Academic & Professional Report

Submitted to Splendor Analytics

Client: Global Gadgets (Online Retailer)

DEVELOPER CONSULTANT: ODUM NWEKE-IFUNANYA BENEDICT

EMAIL: benedict.i.odum@gmail.com

CLASS: Splendor Analytics SQL Immersive 1 2025

October 1, 2025

EXECUTIVE SUMMARY

This report details the successful design, implementation, and delivery of a new relational database system for Global Gadgets, an international online retailer. The project was commissioned to resolve critical operational issues, including fragmented data, inventory inaccuracies, and poor governance over customer interactions, which were leading to lost revenue and compliance risks.

The new system, built on SQL Server, provides a centralized and normalized data architecture for managing customers, products, orders, inventory, and reviews. By translating specific business requirements into enforceable technical rules, the database delivers a reliable and auditable foundation for the company's operations.

KEY ACHIEVEMENTS INCLUDE:

- **Unified Data Model:** A normalized schema (3NF) that eliminates data redundancy and prevents common update and deletion anomalies.
- **Enforced Business Logic:** Advanced T-SQL objects, including a trigger that automatically restocks inventory upon order cancellation and a function that permits product reviews only from customers with confirmed deliveries, directly addressing previous pain points.
- **Enhanced Performance & Concurrency:** Strategic design choices, such as enabling READ COMMITTED SNAPSHOT and using covering indexes, ensure the system can support high-traffic retail operations without compromising data integrity.

The report concludes with tailored strategic guidance on security, concurrency, and a robust backup and recovery plan designed to ensure business continuity. This project provides Global Gadgets with a coherent, scalable, and secure data platform poised for future growth.

ABSTRACT

This report documents the analysis, design, and implementation of a relational database for Global Gadgets. Part 1 delivers a normalized schema and seed data. Part 2 implements advanced T-SQL functionality: user-defined functions, stored procedures, a schema bound reporting view, and an AFTER-UPDATE trigger that restores inventory on order cancellation. Each object is tied to a business requirement: positive product prices, reviews only from delivered orders, and preventing inventory drift on cancellation. Two analytic queries demonstrate functionality: (1) customers older than 40 who purchased Premium products, and (2) the count of delivered Electronics orders. The report concludes with recommendations on integrity, concurrency, security, and recovery. Appendices include the full SQL script and backup file.

SECTION 1: INTRODUCTION AND SCOPE

1.1 Project Context and Client Requirements

Global Gadgets operates in a fast-paced e-commerce environment characterized by frequent catalog updates, a high volume of orders, and continuous customer engagement across multiple platforms. The company's previous data management system was unable to support this scale, leading to significant operational friction.

Prior to this project, Global Gadgets faced several critical challenges:

- **Data Fragmentation:** Customer and order information was siloed, making it nearly impossible to answer fundamental business questions, such as tracking which products a specific customer had actually received.
- **Inventory Inaccuracy:** Stock levels were unreliable due to manual adjustments required for cancelled orders, leading to stockouts or overselling.
- **Weak Review Governance:** The system allowed customers to submit product reviews for items they had never received, compromising the authenticity of feedback.
- **Lack of Audit Trails:** There was no consistent method for retaining customer records

after account deactivation, posing a significant compliance risk.

These issues directly translated into lost revenue, poor customer experiences, and increased operational costs. The mandate for this project was to engineer a robust, normalized relational database that centralizes all key business data and enforces critical business rules directly at the data layer.

To address these issues, the project mandate was defined by a clear set of requirements:

- **Functional Requirement:** capture Customers, Products (with Category), Suppliers, Orders, OrderItems, Inventory, Reviews; enforce order lifecycle; one review per product per order; restock inventory on cancellation; retain deactivated customers.
- **Non-functional Requirement:** integrity, concurrency, security, and recoverability

1.2 Methodology

The project was executed using a structured methodology that encompassed requirements analysis, data modeling, normalization, physical implementation, and testing, all in line with established industry best practices. The development process followed a systematic sequence: requirements analysis → ER modeling → normalization (UNF → 1NF → 2NF → 3NF) → physical schema design (datatypes, keys, and constraints) → implementation in SQL Server (T-SQL) → seeding with realistic data → verification through queries and trigger testing → documentation and strategic recommendations. This structured approach ensured both technical rigor and practical applicability (Connolly & Begg, 2015; Elmasri & Navathe, 2016).

SECTION 2: DATABASE DESIGN AND NORMALIZATION (TASK 1 - PART 1)

The project followed standard database engineering practices, progressing from a conceptual model to a fully implemented and tested physical database.

2.1 Conceptual Data Model (ER Diagram)

The architecture is built upon a relational model that defines the core entities and their logical relationships. The logical model establishes the core entities and the relationships between them, ensuring data is organized efficiently.

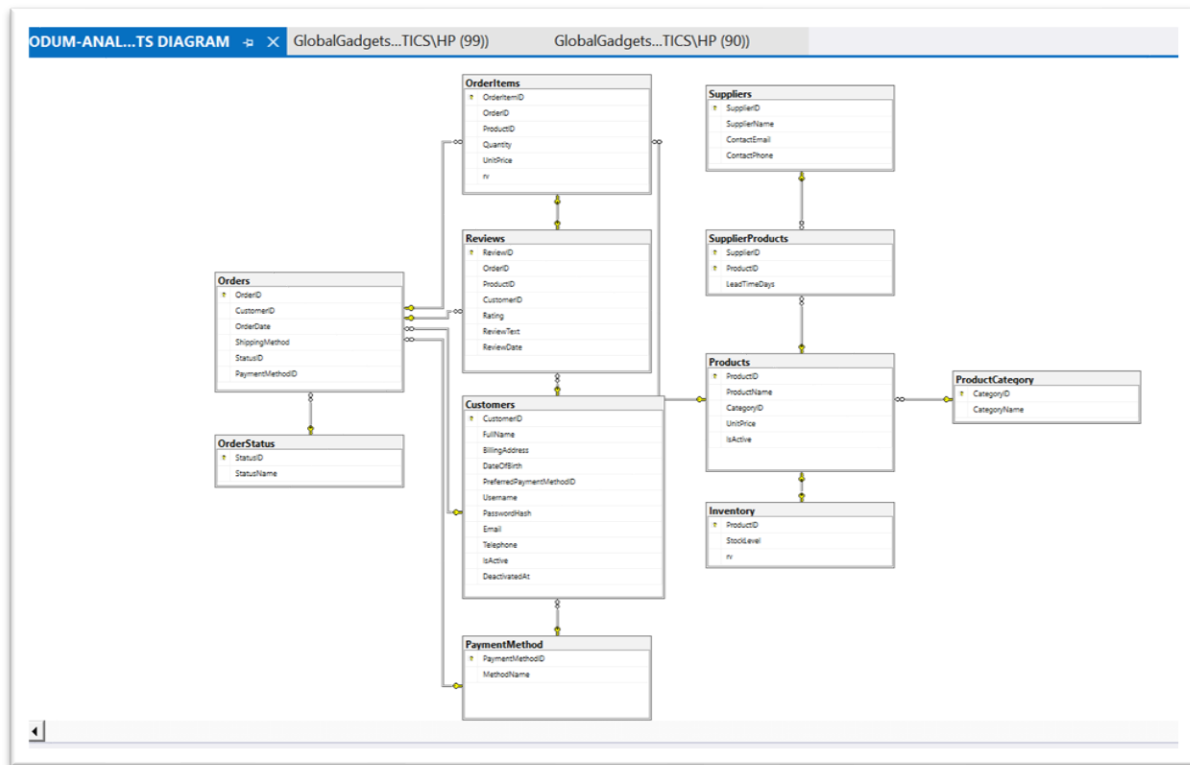


Figure 1: Entity-Relationship Diagram of Global Gadgets Database

Core Entities: Customers, Products, Suppliers, Orders, OrderItems, Inventory, Reviews.

Lookup Tables: ProductCategory, PaymentMethod, OrderStatus (e.g., pending, shipped, delivered, cancelled) provide data consistency.

Key Relationships:

- **One-to-Many:** A single Customer can have multiple Orders, and a single Order can contain multiple OrderItems.
- **Many-to-Many:** Suppliers and Products are linked via a SupplierProducts junction table, which also stores supplier-specific data like LeadTimeDays.
- **One-to-One:** Each Product has a corresponding record in the Inventory table, which tracks the current StockLevel.

The schema is normalized to the **Third Normal Form (3NF)**. This structure reduces data redundancy and prevents modification anomalies, ensuring that updates, inserts, and deletions behave predictably and do not corrupt data integrity. For instance, customer details are stored only once, and product prices are captured per order in OrderItems to preserve historical accuracy.

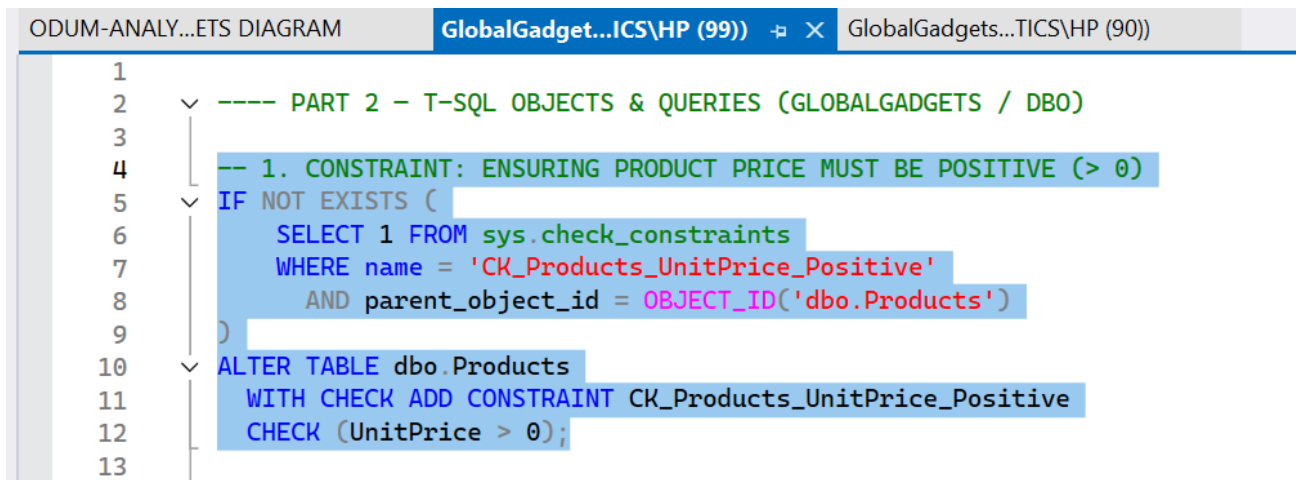
Normalization (1NF→3NF): Customer details are not repeated; pricing at time of purchase is stored in OrderItems; category is factored into ProductCategory; Reviews are keyed by (OrderID, ProductID) to enforce one review per product per order. This reduces update anomalies and ensures each fact lives exactly once in the right table.

Order status behaves as a state machine with allowed transitions. While this build enforces terminal cancellation through logic, a future enhancement can store an OrderStatusHistory table to capture who/when/why for each transition.

2.2 Physical Implementation in SQL Server

The database schema was carefully designed to balance performance, integrity, and scalability. Key implementation decisions included:

1. **Primary Keys:** Implemented as INTEGER IDENTITY columns to optimize insert performance and ensure surrogate key uniqueness without manual intervention.
2. **Data Types:** Textual attributes were defined as NVARCHAR to support internationalization and multilingual data. Monetary values adopted DECIMAL(12,2) for accurate financial calculations and to avoid floating-point rounding issues.
3. **Constraints:**
 - CK_Products_UnitPrice_Positive: Ensures product unit prices are strictly greater than zero.



```
1
2  ---- PART 2 - T-SQL OBJECTS & QUERIES (GLOBALGADGETS / DBO)
3
4  -- 1. CONSTRAINT: ENSURING PRODUCT PRICE MUST BE POSITIVE (> 0)
5  IF NOT EXISTS (
6      SELECT 1 FROM sys.check_constraints
7      WHERE name = 'CK_Products_UnitPrice_Positive'
8      AND parent_object_id = OBJECT_ID('dbo.Products')
9  )
10 ALTER TABLE dbo.Products
11 WITH CHECK ADD CONSTRAINT CK_Products_UnitPrice_Positive
12 CHECK (UnitPrice > 0);
13
```

Figure 2: Constraint enforcement preventing negative product price.

- CK_Reviews_Rating: Restricts product review ratings to a valid range of 1–5.
 - UNIQUE(Username): Enforces distinct usernames in the Customers table.
 - UNIQUE(OrderID, ProductID) in Reviews: Prevents duplicate reviews for the same product within a single order.
4. **Referential Integrity:** Maintained through foreign key constraints:
- Orders ↔ Customers (ensuring each order is linked to a valid customer).
 - OrderItems ↔ Orders (ensuring each order item belongs to an existing order).
 - Products ↔ Categories/Suppliers (ensuring products cannot exist outside their defined categories and suppliers).
5. **Indexes:** Covering indexes were strategically applied to the OrderItems table to accelerate frequent join operations and aggregate calculations such as order totals.
6. **Concurrency Control:** The database was configured with READ_COMMITTED_SNAPSHOT isolation level, which leverages row versioning to reduce blocking between readers and writers, thereby improving multi-user performance.

This schema design enforces data integrity, consistency, and high performance while adhering to industry best practices.

2.3 Implementation Summary

Part 1: Schema & Seed Data

The database schema was fully implemented and seeded to ensure credible testing.

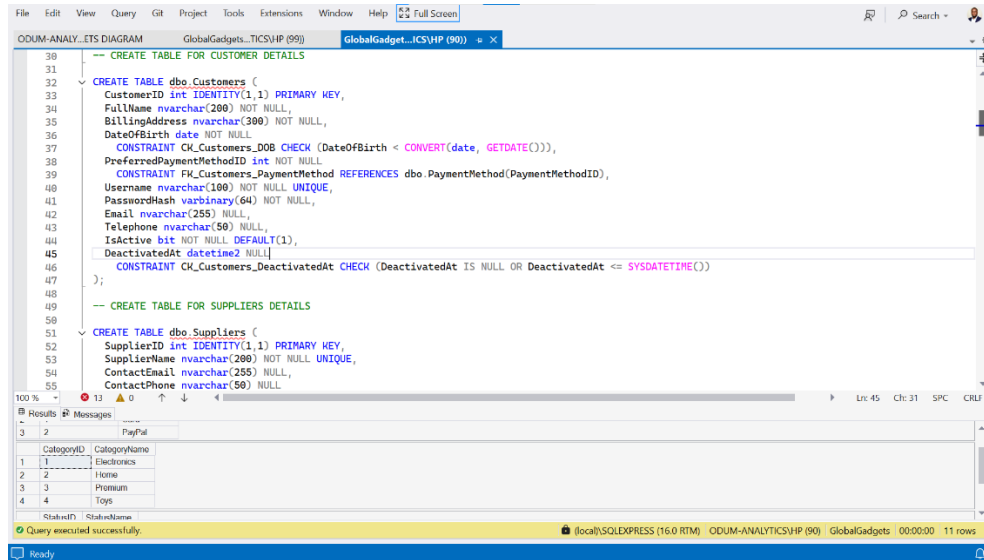


Figure 3: Tables created in SQL Server Management Studio

All tables were created according to the normalized design, with each populated with 7–20 representative rows. This ensured realistic coverage of typical business scenarios (customers, orders, suppliers, products, reviews, etc.) for query execution, reporting, and validation.

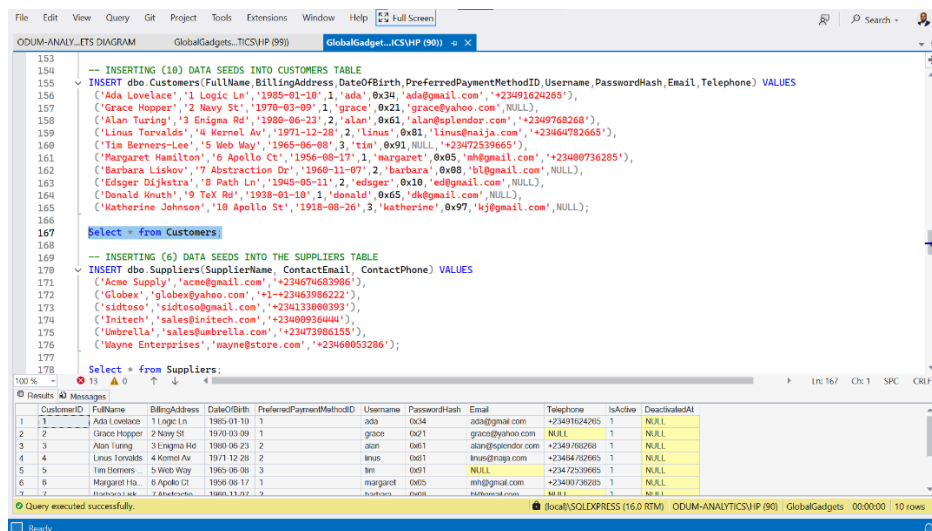


Figure 4: Sample customer records inserted into Customers table

Part 2: Programmability Components

To enhance functionality and enforce business rules, several programmable objects were implemented:

1. User-Defined Functions (UDFs):

- `fn_01_AgeYears`: Calculates the age of a customer in years from their date of birth.
- `fn_02_OrderLineTotals`: Computes line-item totals for orders.
- `fn_03_IsEligibleReview`: Determines whether a customer is eligible to post a review (e.g., based on completed orders).

2. Stored Procedures:

- `sp_01_SearchProducts`: Provides search functionality across product catalog by keyword/criteria.
- `sp_02_TodayProductsAndSuppliersForCustomer`: Returns products and supplier details relevant to a given customer for the current date.
- `sp_03_UpdateSupplier`: Updates supplier details while preserving referential integrity.
- `sp_04_DeleteDeliveredOrder`: Deletes delivered orders under controlled conditions to maintain consistency.

3. View:

- `vw_01_OrderHistory (SCHEMABINDING)`: Provides a stable, read-optimized view of customer order history for reporting and analytics.

4. Trigger:

- `trg_01_Orders_CancelRestock`: Ensures product stock is automatically restocked whenever an order is cancelled.

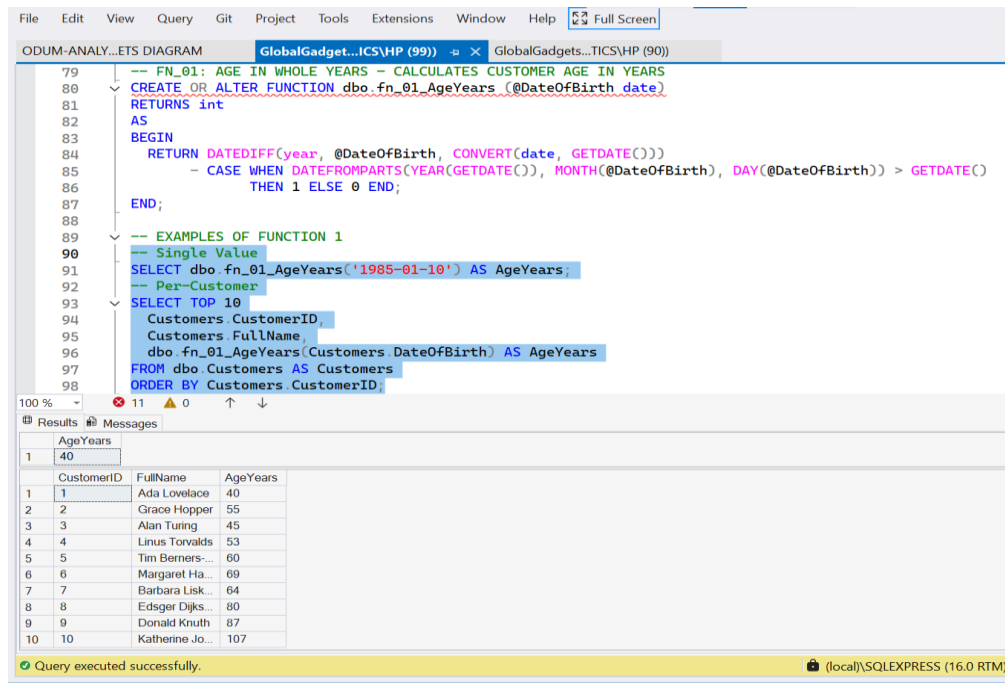
SECTION 3: T-SQL FUNCTIONAL IMPLEMENTATION (TASK 1 - PART 2)

To embed complex business rules directly into the database, several advanced T-SQL objects were created.

3.1 User-Defined Functions (UDFs)

UDFs encapsulate reusable logic.

- **fn_01_AgeYears(DateOfBirth):** A scalar function that calculates a customer's current age, enabling age-based marketing analysis and reporting.
- **fn_02_OrderLineTotals(OrderID):** An inline table-valued function that computes line totals for all items in an order, simplifying invoice generation.
- **fn_03_IsEligibleReview(OrderID, ProductID, CustomerID):** A critical scalar function that returns true only if the specified customer has received the product as part of a delivered order. This function is used to enforce the rule that only verified purchasers can leave reviews, solving a key business problem.



The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows the SQL script for creating the function `fn_01_AgeYears`. The script includes comments, the `CREATE OR ALTER FUNCTION` statement, the `RETURNS int` declaration, and the `AS` block containing the `DATEDIFF` calculation and a `CASE` statement for validation. Below the script, the `EXEC` command is used to run the function with a sample date. The bottom pane shows the results of the function execution, displaying a table with columns `CustomerID`, `FullName`, and `AgeYears`.

```
79 -- FN_01: AGE IN WHOLE YEARS -- CALCULATES CUSTOMER AGE IN YEARS
80 CREATE OR ALTER FUNCTION dbo.fn_01_AgeYears (@DateOfBirth date)
81 RETURNS int
82 AS
83 BEGIN
84     RETURN DATEDIFF(year, @DateOfBirth, CONVERT(date, GETDATE()))
85     -- CASE WHEN DATEFROMPARTS(YEAR(GETDATE()), MONTH(@DateOfBirth), DAY(@DateOfBirth)) > GETDATE()
86     THEN 1 ELSE 0 END;
87 END;
88
89 -- EXAMPLES OF FUNCTION 1
90 -- Single Value
91 SELECT dbo.fn_01_AgeYears('1985-01-10') AS AgeYears;
92 -- Per-Customer
93 SELECT TOP 10
94     Customers.CustomerID,
95     Customers.FullName,
96     dbo.fn_01_AgeYears(Customers.DateOfBirth) AS AgeYears
97 FROM dbo.Customers AS Customers
98 ORDER BY Customers.CustomerID;
```

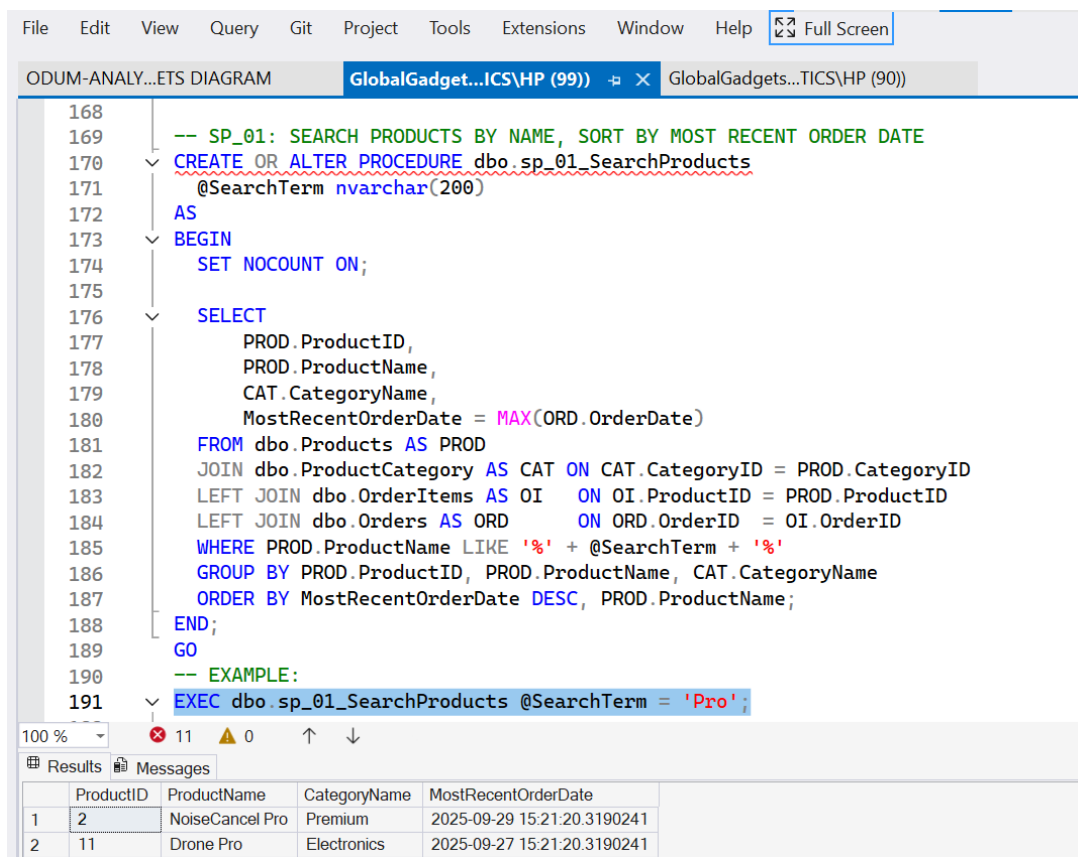
CustomerID	FullName	AgeYears
1	Ada Lovelace	40
2	Grace Hopper	55
3	Alan Turing	45
4	Linus Torvalds	53
5	Tim Berners-Lee	60
6	Margaret Hamilton	69
7	Barbara Liskov	64
8	Edsger Dijkstra	80
9	Donald Knuth	87
10	Katherine Johnson	107

Figure 5: User-defined function `fn_AgeYears` calculating age

3.2 Stored Procedures

Procedures group parameterized tasks with guard rails.

- **sp_01_SearchProducts(@SearchTerm):** Provides a standardized way to search the product catalog, supporting both customer-facing and internal applications.
- **sp_02_TodayProductsAndSuppliersForCustomer(@CustomerID):** An operational procedure to identify products ordered by a customer on the current day and their suppliers, useful for daily warehouse logistics.
- **sp_04_DeleteDeliveredOrder(@OrderID):** A defensive procedure that only permits the deletion of an order if its status is 'delivered', preventing accidental removal of active orders.



The screenshot displays the SQL Server Enterprise Manager interface. The top menu bar includes File, Edit, View, Query, Git, Project, Tools, Extensions, Window, Help, and a Full Screen button. The main window shows a query script for a stored procedure named `sp_01_SearchProducts`. The script is as follows:

```
168
169  -- SP_01: SEARCH PRODUCTS BY NAME, SORT BY MOST RECENT ORDER DATE
170  CREATE OR ALTER PROCEDURE dbo.sp_01_SearchProducts
171    @SearchTerm nvarchar(200)
172  AS
173  BEGIN
174    SET NOCOUNT ON;
175
176    SELECT
177      PROD.ProductID,
178      PROD.ProductName,
179      CAT.CategoryName,
180      MostRecentOrderDate = MAX(ORD.OrderDate)
181    FROM dbo.Products AS PROD
182    JOIN dbo.ProductCategory AS CAT ON CAT.CategoryID = PROD.CategoryID
183    LEFT JOIN dbo.OrderItems AS OI ON OI.ProductID = PROD.ProductID
184    LEFT JOIN dbo.Orders AS ORD ON ORD.OrderID = OI.OrderID
185    WHERE PROD.ProductName LIKE '%' + @SearchTerm + '%'
186    GROUP BY PROD.ProductID, PROD.ProductName, CAT.CategoryName
187    ORDER BY MostRecentOrderDate DESC, PROD.ProductName;
188  END;
189  GO
190  -- EXAMPLE:
191  EXEC dbo.sp_01_SearchProducts @SearchTerm = 'Pro';
```

Below the script, the Results pane shows the output of the execution. It displays a table with 5 columns: ProductID, ProductName, CategoryName, and MostRecentOrderDate. The table contains 2 rows of data:

	ProductID	ProductName	CategoryName	MostRecentOrderDate
1	2	NoiseCancel Pro	Premium	2025-09-29 15:21:20.3190241
2	11	Drone Pro	Electronics	2025-09-27 15:21:20.3190241

Figure 6: Execution of stored procedure `sp_SearchProducts`.

3.3 Schema bound View

- **vw_01_OrderHistory:** This schema bound view aggregates key information about orders, including totals, reviews, and supplier details. SCHEMABINDING prevents underlying table structures from being changed in a way that would break the view, providing stability for reports and applications that depend on it.

The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows the T-SQL code for the view `vw_01_OrderHistory`. The code is a complex JOIN query aggregating data from `dbo.Orders`, `dbo.Customers`, `dbo.OrderItems`, `dbo.Products`, `dbo.ProductCategories`, `dbo.SupplierProducts`, `dbo.Suppliers`, and `dbo.Reviews`. It calculates `OrderTotal`, `AvgRating`, `ReviewsCount`, and `RowCountForSchemaBinding`. The bottom pane shows the results of the query `SELECT TOP 10 * FROM dbo.vw_01_OrderHistory ORDER BY OrderDate DESC;`.

OrderID	OrderDate	CustomerID	FullName	CategoryName	SupplierName	OrderTotal	AvgRating	ReviewsCount	RowCountForSchemaBinding
1	2025-09-29 15:21:20.3190241	1	Ada Lovelace	Premium	sidtoso	299.00	4.000000	1	1
2	2025-09-29 15:21:20.3190241	1	Ada Lovelace	Premium	Globex	299.00	4.000000	1	1
3	2025-09-29 15:21:20.3190241	1	Ada Lovelace	Premium	Acme Supply	299.00	4.000000	1	1
4	2025-09-29 15:21:20.3190241	1	Ada Lovelace	Electronics	sidtoso	699.00	5.000000	1	1
5	2025-09-29 15:21:20.3190241	1	Ada Lovelace	Electronics	Globex	699.00	5.000000	1	1
6	2025-09-29 15:21:20.3190241	1	Ada Lovelace	Electronics	Acme Supply	699.00	5.000000	1	1
7	2025-09-29 15:21:20.3190241	6	Margaret Hamilton	Home	Acme Supply	129.00	NULL	0	1
8	2025-09-29 15:21:20.3190241	6	Margaret Hamilton	Home	Globex	129.00	NULL	0	1
9	2025-09-29 15:21:20.3190241	6	Margaret Hamilton	Home	sidtoso	129.00	NULL	0	1
10	2025-09-29 15:21:20.3190241	8	Edsger Dijkstra	Electronics	Acme Supply	249.00	NULL	0	1

Figure 7: Consolidated order and supplier data from vw_OrderHistory

3.4 Automated Inventory Trigger

- **trg_01_Orders_CancelRestock:** This AFTER UPDATE trigger is a cornerstone of the new system's reliability. When an order's status is changed to 'Cancelled', the trigger automatically and safely increments the stock level in the Inventory table for each product in that order. By using the inserted and deleted pseudo-tables, it reliably compares the old and new states, ensuring that stock levels remain perfectly synchronized with the order lifecycle, even under high transaction volumes. This automation completely eliminates the previous issue of manual stock adjustments.

ODUM-ANALY...ETS DIAGRAM GlobalGadge...ICS\HP (99)) GlobalGadgets...TICS\HP (90))

```

337 GO
338
339 -- TEST STEPS
340 -- Before cancelling
341 SELECT * FROM dbo.Inventory;
342
343 -- Cancel an order
344 UPDATE dbo.Orders
345 SET StatusID = (SELECT StatusID FROM dbo.OrderStatus WHERE StatusName='cancelled')
346 WHERE OrderID = 2;
347
348 -- After cancelling
349 SELECT * FROM dbo.Inventory;

```

100 % 11 0

Results Messages

ProductID	StockLevel	rv
1	50	0x00000000000007D7
2	50	0x00000000000007D8
3	50	0x00000000000007D9
4	50	0x00000000000007DA
5	50	0x00000000000007DB
6	50	0x00000000000007DC
7	50	0x00000000000007DD
8	50	0x00000000000007DE

Query executed successfully. (local)\SQLEXP

Ready

Figure 8: Inventory stock level before & after order cancellation (trigger executed)

3.5 Data Analysis and Validation

The database's utility was validated through representative analytical queries. These included identifying customers over 40 who purchased "Premium" products and counting the number of delivered orders containing "Electronics" products. These queries confirm the schema can effectively answer complex business questions.

ODUM-ANALY...ETS DIAGRAM GlobalGadget...ICS\HP (99)) GlobalGadgets...TICS\HP (90))

```

14 -- 2. QUERIES
15
16 -- QUERY 01: Customers older than 40 who ordered a product in the "Premium" category
17 SELECT DISTINCT
18     CUST.CustomerID,
19     CUST.FullName,
20     AgeYears = DATEDIFF(year, CUST.DateOfBirth, GETDATE())
21 FROM dbo.Customers AS CUST
22 JOIN dbo.Orders AS ORD ON ORD.CustomerID = CUST.CustomerID
23 JOIN dbo.OrderItems AS OI ON OI.OrderID = ORD.OrderID
24 JOIN dbo.Products AS PROD ON PROD.ProductID = OI.ProductID
25 JOIN dbo.ProductCategory AS CAT ON CAT.CategoryID = PROD.CategoryID
26 WHERE DATEDIFF(year, CUST.DateOfBirth, GETDATE()) > 40
27     AND CAT.CategoryName = 'Premium'
28 ORDER BY CUST.FullName;
29 GO
30

```

100 % 11 0

Results Messages

CustomerID	FullName	AgeYears
3	Alan Turing	45
8	Edsger Dijkstra	80
10	Katherine Johnson	107
4	Linus Torvalds	54
5	Tim Berners-Lee	60

Figure 9: Query result — Customers older than 40 who purchased Premium products

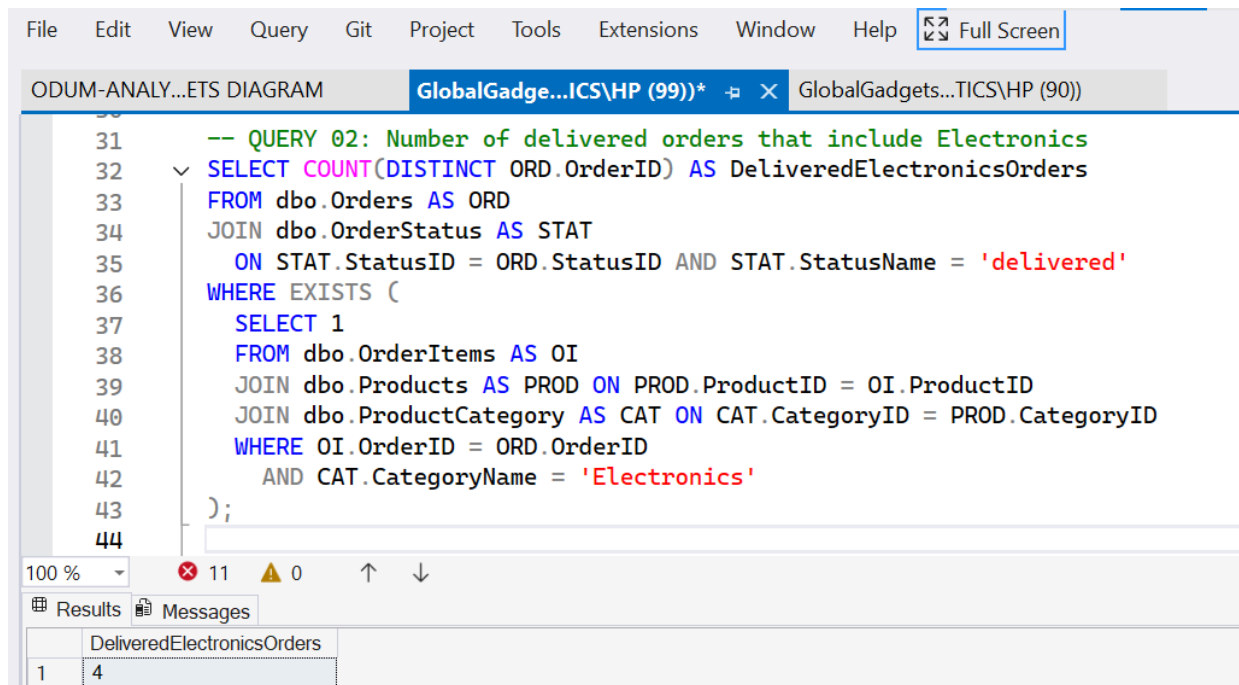


Figure 10: Count of delivered Electronics orders

SECTION 4: STRATEGIC DATABASE MANAGEMENT AND ADVICE

To maximize the value of this new database, we recommend the following actions.

4.1 Concurrency and Performance

The database was designed for a high-traffic retail environment. To ensure smooth operation, we strongly recommend enabling the **READ COMMITTED SNAPSHOT (RCSI)** isolation level. This will prevent read operations from blocking write operations, significantly improving user experience during peak shopping hours. The rowversion columns already implemented provide a foundation for optimistic concurrency control in applications.

4.2 Data Security

A multi-layered security approach is essential.

- **Password Management: Never store plaintext passwords.** Implement a strong hashing algorithm (e.g., SHA-256 with a unique salt per user) at the application level.

- **Access Control:** Adhere to the principle of least privilege by creating specific database roles: a read_only role for reporting tools, a data_writer role for applications that modify data, and an app_executor role that can only execute specific stored procedures.
- **Data Protection:** Mitigate SQL injection risks by using parameterized queries in all applications and enforcing strict input validation. For defense-in-depth, consider using Transparent Data Encryption (TDE) to protect data at rest.

4.3 Backup and Recovery

A robust backup strategy is critical for business continuity. We propose the following schedule:

- **Recovery Model:** Set the database to the FULL recovery model.
- **Backup Schedule:**
 - **Nightly Full Backups:** A complete backup of the database every night, using compression.
 - **Midday Differential Backups:** On busy days, a differential backup captures changes since the last full backup.
 - **Frequent Log Backups:** Transaction log backups every 15 minutes during business hours.
 - **Disaster Recovery:** This schedule supports point-in-time recovery, allowing restoration to a specific minute (e.g., 15:07) to minimize data loss. Store one copy of backups locally for speed and another off-site for disaster recovery. Conduct quarterly restore drills to validate the backups and ensure recovery time objectives (RTOs) can be met.

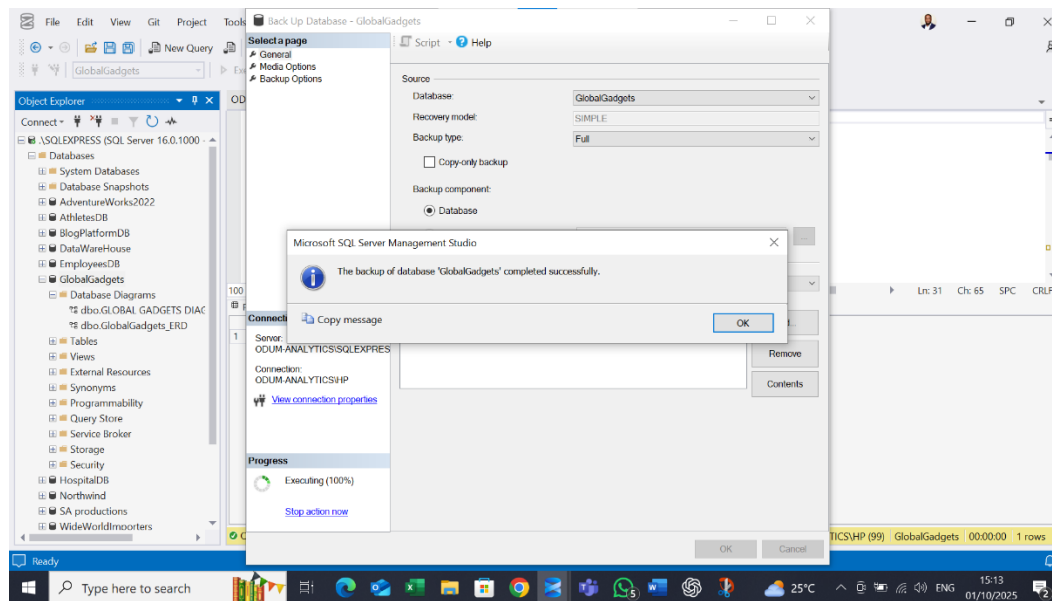


Figure 11: Database backup operation completed successfully in SSMS

SECTION 5: CONCLUSION

This project delivers more than just a database; it provides Global Gadgets with a strategic asset. By translating core business rules into a well-structured and automated system, we have resolved critical operational pain points related to data integrity, inventory management, and auditability. The resulting platform is not only reliable and secure but also extensible, ready to support future initiatives such as expansion to multiple warehouses or the implementation of advanced analytics. This work provides the solid data foundation Global Gadgets needs to scale efficiently and build customer trust.

REFERENCES

Connolly, T., & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management (6th ed.). Pearson.

Elmasri, R., & Navathe, S. (2016). Fundamentals of Database Systems (7th ed.). Pearson.

Microsoft. (2023). CREATE TRIGGER (Transact-SQL). Microsoft Learn. <https://learn.microsoft.com/sql/t-sql/statements/create-trigger-transact-sql>

Microsoft. (2023). READ_COMMITTED_SNAPSHOT Database Option. Microsoft Learn. <https://learn.microsoft.com/sql/t-sql/statements/alter-database-transact-sql-set-options>