

FLY! FLY! FLY!



Team 3 – FiveGuys

Abel Macedo, Benayah Hamonangan, Joshua Cottrill, Rodrigo Tumani Soubhia Giesteira

MIS 3353 – Database Management

Professor Swetha Siripurapu

April 28, 2024

*“Stronger Together”*

# **Executive Summary**

Elysian Fly Company, a forward-thinking company dedicated to enhancing profitability and fostering growth, is embarking on a journey to recreate and refine their operational framework and decision-making processes. Upon recognizing the positive impact that technology has towards driving efficiency and customer value, Elysian Fly seeks to implement an enhanced relational database system to help streamline its operations.

This proposed project seeks to develop an Entity-Relationship Diagram (ERD) that is tailored specifically to the needs of Elysian Fly. By utilizing the advanced database system that we will create, Elysian aims to enhance operational efficiency, improve decision-making capabilities, and ultimately deliver greater customer value.

The important objectives of this project includes:

1. **Operational Efficiency Improvement**: The implementation of a relational database system will assist Elysian Fly to streamline its operational processes – both externally and internally – which therefore will help reduce redundancies and optimize resource utilization.
2. **Improved Decision-Making:** With a more refined database system, Elysian Fly will have access to more detailed data insights. This will help decision-makers within the company to make informed decisions, fueled by precise and detailed data. Doing this will help lead to more effective resource allocation, strategic planning, and business growth.
3. **Customer Value Creation:** By optimizing the distribution of service through the creation of the ERD, Elysian Fly aims to enhance the overall customer experience. With the hopes of improving customer experience and responsiveness, the company will be able to accommodate better to customers' needs, build stronger relationships, and develop customer loyalty and satisfaction.

Overall, the development of an ERD for Elysian Fly Company represents a strategic initiative to help develop operational excellence, facilitate data-driven decision-making, and enhance customer value.

Over the length of the project, the total cost is $967.

Contents

[Executive Summary 2](#_Toc338719663)

[Get to Know the Team: 5](#_Toc1358042784)

[Q&A During the Meeting & Information We Learned 9](#_Toc1127996762)

[Significant Assumptions 10](#_Toc458299715)

[What is an ERD? Why is it necessary? 11](#_Toc1611151310)

[Business Cycles Used 12](#_Toc420045012)

[Data Provided by the Client 12](#_Toc1783719392)

[ERD Created 13](#_Toc711974836)

[Changes made to generic ERDs (Query Feasibility) 14](#_Toc132309797)

[Logical Design 16](#_Toc1083011730)

[Normalization 17](#_Toc804963369)

[Normalized Relations 17](#_Toc1224270145)

[Differences between ERD and Normalized Relations 22](#_Toc293278447)

[Referential Integrity 22](#_Toc1616569502)

[Physical Design and Implementation 22](#_Toc992229281)

[Data Dictionary 23](#_Toc166081104)

[Denormalization 23](#_Toc1603870120)

[Implemented Physical Design 23](#_Toc768916949)

[Challenges Faced/Addressed During Implementation 24](#_Toc2114562314)

[Strengths and Weaknesses Encountered During Implementation 24](#_Toc1336931040)

[Specific SQL Statements Requested 25](#_Toc288261025)

[Three Additional Queries 31](#_Toc744277326)

[User Documentation 32](#_Toc1805087308)

[What We Learned Throughout This Process 46](#_Toc1179563424)

[Appendix 48](#_Toc1550397219)

[Team Contract 49](#_Toc968519154)

[Data Dictionary Model 49](#_Toc1862180359)

[Project Management 54](#_Toc1861919265)

# **Get to Know the Team:**

*Abel Macedo*



Hello! My name is Abel Macedo and I am a senior at the University of Oklahoma, studying Management Information Systems at the Price College of Business. During my free time, I really like to play sports at the Sarkey’s Fitness Center with my friends. During the week, we’d try to find time to play some pickup basketball, which I really enjoy. While I am currently searching for internships in the summer, I believe I will bring a hard-working ethic that will bring value to this team.

*Benayah Hamonangan*



My name is Benayah Hamonangan and currently, I am a Junior studying Management Information Systems (MIS) and Sports Business here at the Price College of Business, and while I am originally from Jakarta, Indonesia, I spent most of my life in Doha, Qatar. In my free time, I enjoy trying new places to eat, playing sports, and spending time with my loved ones. Currently, I work with the OU Men’s Basketball team, but I have gained prior work/internship experience in the social media space and various operations roles.

*Joshua Cottrill*



Hello, my name is Joshua Cottrill, and I am a junior at the Price College of Business pursuing a Bachelor of Business Administration degree with a specialization in Management Information Systems. I am originally from Dallas, Texas, and I am a member of Kappa Sigma Fraternity at the University of Oklahoma. While I am presently looking for internship opportunities, I bring a keen attention to detail and a strong work ethic that I believe will benefit our team!

*Rodrigo Tumani Soubhia Geisteira*



Hi, my name is Rodrigo, and I am currently a Junior, studying MIS at the Price College of Business. Originally from Brazil, I come from a very diverse cultural background. I enjoy watching football games during my free time as I am a big supporter of the football club Sao Paulo. With zero internship experience, I bring a great deal of knowledge with regards to the programming aspects of the information systems world and with the ability to utilize those skills to recognize similar patterns and applications within Databases, I believe that I provide great value to our team.

**Conceptual Design**

At this stage, the emphasis changes to creating a high-level abstract overview of the system, outlining the essential components and their connections. The conceptual design stage involves the creation of the ERD and the conceptual schema, which describes the database's logical structure. The primary output is the Entity-Relationship Diagram (ERD), which visually the system’s entities and relationship step and serves as the foundation for the succeeding stages of database architecture, outlining a clear path for future development and execution.

**The Client Meeting**

The client meeting was held via Zoom at 10:15 AM. The owner of the company, who was the interviewee, was accompanied by a group of interviewers, including Abel Macedo, Benayah Hamonangan, Joshua Cottrill, and Rodrigo Tumani Soubhia Giesteira. This meeting discussed goals, expectations, and ideas. The owner, whose unique needs and problems were carefully considered by the interviewers, effectively communicated the company’s value proposition throughout the session to promote a better understanding. As the meeting ended with gratitude from both parties, concrete future actions were outlined, opening the door for a cooperative relationship based on communication, trust, and shared objectives.

* Meeting Time: Our group met at 10:15 AM
* Location: Over Zoom
* Interviewers: Abel Macedo, Benayah Hamonangan, Joshua Cottrill, Rodrigo Tumani Soubhia Giesteira
* Interviewee: The Owner of the company

## **Q&A During the Meeting & Information We Learned**

1- How does the company manage the relationships with fly vendors and artisans? Is there any specific criteria that products must meet?  
● There are specific quality criteria that the products must meet because the vendors  
and artisans are notorious for their high quality, and the company requires the best  
quality.  
2- Regarding inventory management, are there specific requirements for tracking  
inventory levels and stock availability for each product?  
● No, there are no specific requirements, which means assumptions must be made.  
3. How are the customer types such as mentioned local, and tourist represented. Should they be separate?  
● No, they must be separate.  
4. It mentions that “Elysian also supplies do-it-yourself (DIY) fly tyers with a wide  
assortment of needed materials (e.g., hooks, thread, feathers, DIY books, etc.).” From that statement, would you recommend having the DIY supplies as a distinct product category in the database, allowing us to track inventory/ sales etc. for only those specific materials?  
● You can just label it how you want it is still just materials at the end of the day.  
5. With each customer being able to invite their friends to tag along on each trip, will it be the case that because only one customer pays, there can’t be a group payment and that instead, the people among the group will have to figure out by themselves how they are to pay that one person back?  
● Yes, so because there is only one person that pays, that group will have to figure out  
by themselves how they want to split the cost/pay the person back because the  
payment can only be conducted by one person.  
6. So, with there being a record of which lakes and streams each of the guides knows well, would it mean that each guide is a guide for one specific lake/stream, or can each guide also perform a tour for other lakes/streams?  
● Each guide has a specialty, but they are not limited to what they are knowledgeable  
about. It’s more so about the skills that they possess.

# **Significant Assumptions**

After conducting our interview, along with obtaining information from the reading, we realized that there is some information that would need to be further clarified when it came to creating our ERD. As a result, we have established the following assumptions:

**Assumption 1: Vendor Preference and Product Quality Assurance**

Considering that Elysian orders different types of raw materials from different vendors, we assumed the company has no preference for any vendors, apart from checking their product quality. Because of this, we only created an attribute for a vendor reliability rank in the vendor table to track which will help the company keep track of the preffered vendors to do business with.

**Assumption 2: Multi-Product Orders from Customers**

We assume that each order can contain multiple different products: Orders placed by customers may include multiple products, such as fly-tying materials, or guided trips.

**Assumption 3: Sales Transactions can involve one or more payment method**

When customers place orders, they may use different payment methods, such as credit cards, cash, or checks. Therefore, we created a reference table named [PayType], indicating that each order can have multiple payment methods.

**Assumption 4: Products differ from guided trips**

We assume that guided trips, which are services provided by the company, should be modeled separate from product, as those encompass different tracking needs. Because of this, we created a separate entity for Guided Trips that has a relationship with the tables Employee (As trip guides) and Sales Order, to record the financials of the trip.

**Assumption 5: Localized Employee Presence**

We assume that establishing a direct relationship between the Employee table and the State reference table is unnecessary due to all employees being situated within the State of Montana.

## **What is an ERD? Why is it necessary?**

A database schema, also referred to as an entity-relationship diagram (ERD), holds paramount importance for a wide range of professionals involved in database design, development, and management. This includes database designers, developers, business analysts, system architects, stakeholders, and data administrators. The ERD serves as a foundational blueprint that defines the structure, organization, and relationships of data entities within the database system. It is important because it shows how data items are connected visually, which helps stakeholders communicate and understand each other better. The Entity Relationship Diagram (ERD) facilitates informed decision-making processes inside businesses, optimizes database efficiency, and helps to ensure data integrity. Essentially, the ERD is an essential tool for developing, putting into practice, and managing database systems that effectively utilize and manage organizational data assets.

# **Business Cycles Used**

Upon the creation of our Entity-Relationship Diagram, we established that we would use all three cycles: Revenue, Production, and Expenditure.

Revenue Cycle - The revenue cycle in Elysian’s business operations focuses on money production and involves measures to improve income streams. It focuses on actions that enhance financial growth and profitability. In revenue cycle, we feel that Elysian Fly not only sells various varieties of flies but also provides guided tours across various lakes and rivers. As a result, integrating features like Customer, Employee, and a SalesOrderLine was critical in developing our ERD to guarantee that we had all of the necessary and relevant data.

Production Cycle - Elysian's production cycle is primarily focused on manufacturing activities, resulting in efficient production processes. Prioritizing resource and process optimization to meet production targets and generate high-quality products. Furthermore, the manufacturing cycle includes RawMaterials and Vendor, both of which are part of Elysian's operations sector. Because Elysian uses a local, family-run manufacturer, those components are incorporated into the systems that govern how Elysian runs, therefore the production cycle must be included.

Expenditure Cycle - The expenditure cycle includes the processes for obtaining products and services required for Elysian’s operations. Elysian is a retailer that sells fly-fishing gear and many types of flies. As a result, our ERD requires a product and employee component, which we may extract from the expenditure cycle. We anticipate better tracking of raw material procurement and vendor negotiations due to effective expenditure cycle management, improving operational efficiency and customer satisfaction.

# **Data Provided by the Client**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Vendor** | **Stellas Outdoor shop** |  |  |  |
| ProductName | Cost | PatternID | SizeID | ColorID |
| SuperFly | $8.99 | 2 | 101 | 212 |
| Flyalicious | $4.99 | 65 | 112 | 203 |
| BigFishFlier | $2.99 | 47 | 108 | 204 |
| FlyingHigh | $10.99 | 12 | 110 | 207 |
|  |  |  |  |  |
| **Vendor** | **BigRigFishing** |  |  |  |
| ProductName | Cost | PatternID | SizeID | ColorID |
| OhFlyGoodness | $1.99 | 37 | 103 | 211 |
| FlyFishinInTheDark | $6.99 | 1 | 108 | 205 |
| BackCountryFly | $7.99 | 93 | 111 | 201 |
| F-22 | $13.99 | 9 | 104 | 209 |
|  |  |  |  |  |
| **Vendor** | **BoomerSoonerFly** |  |  |  |
| ProductName | Cost | PatternID | SizeID | ColorID |
| ProgramFly | $3.99 | 98 | 123 | 216 |
| FlylerMurray | $4.99 | 45 | 117 | 212 |
| FlylySims | $2.99 | 23 | 119 | 214 |
| FlyLikeRoy | $5.99 | 6 | 121 | 207 |

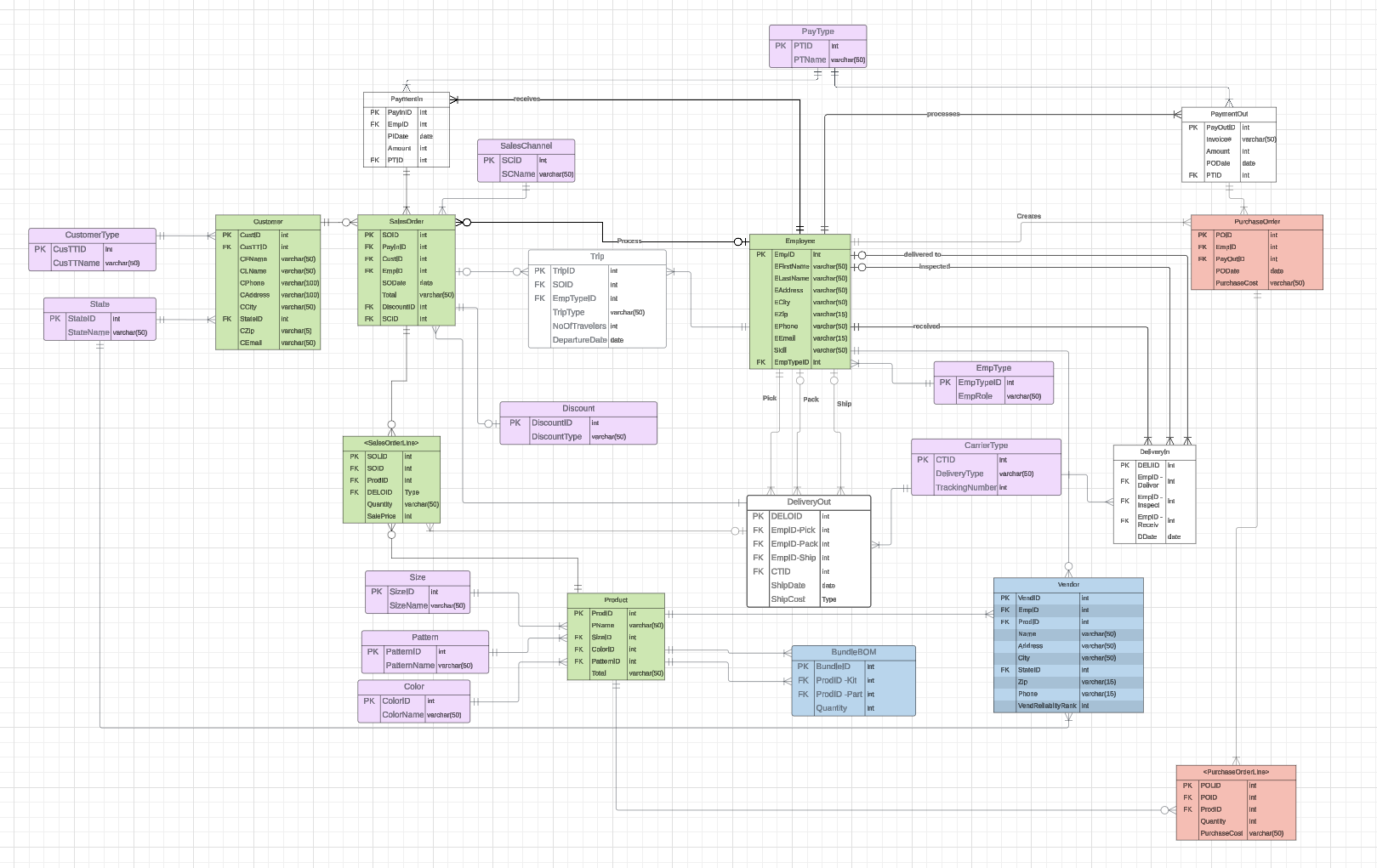
Displayed above is an example of the sample data provided by our client Elysian Fly Company. Within this table is a list of data that is related to the Vendor table, which showcases some of the different vendors that are used by Elysian Fly Company to help create their products, along with the costs associated with each product provided by the vendor and their respective IDs.

Regarding this data's denormalization process, we believe all three tables are in 3NF form. What this means is that all the data included within these three tables are atomic – meaning that there are no multi-valued attributes and that inside each row, there is only one single entity in that specific row/column -- and that there no dependencies between each attribute. What that means is that no specific cell of data determines the outcome or result of another cell of data within another row or column.

# **ERD Created**

After conducting our interview, along with the research we conducted as a team, we have created our initial ERD which portrays some of the findings we have discovered. Since our ERD is founded upon all 3 business cycles, we established that our ERD must comprise of various “main ingredients.” We believe that these “main ingredients” (Product, Employee, Payment Processes, Vendor, as well as the delivery processes) act as pillars that serve as the foundation of the Elysian Fly Company.

Below, we have attached an image of our ERD thus far:



[Team 3 (FiveGuys ERD)](https://lucid.app/lucidchart/b5226175-4828-4f06-b081-7e2b034b7749/edit?viewport_loc=-2947%2C-1043%2C5813%2C2530%2C0_0&invitationId=inv_e826bccc-60db-45d7-a396-0fcb0f59684b)

## Changes made to generic ERDs (Query Feasibility)

After reviewing our initial ERD, we realized elements needed to be added to the ERD or further specified in more detail. Through utilizing reference tables and subtype-supertype relationships, these are the changes we have made to our initial ERD:

|  |  |  |
| --- | --- | --- |
| Change # | Original ERD | Updated ERD |
| 1. We removed the sub-type/super-type relationship for different types of employees. Instead of using that, we created a reference table for the different types of employees. |  |  |
| 2. We created a separate entity for the Trips, to record them differently than the products. | N/A |  |
| 3. We created a reference table for the different types of Sales channels (online, in person, phone). | N/A |  |
| 4. We changed the way we modeled the Vendor Managers. We previously had a reference table for this role from the Vendor Table, but we decided to remove that and keep it simple by creating a relationship between Vendor and Employee, as Vendor managers are a type of employee. |  |  |
| 5. We removed the raw materials table and created a table for BundleBOM with two relations with the product table (Kits and Parts) |  |  |
| 6. We created a reference table named ‘Discount’ to address the different discounts given to certain orders. | N/A |  |

# **Logical Design**

Logical Design entails defining the constraints, relationships, tables, and columns in accordance with the system requirements. Normalization techniques are used in logical design to guarantee data efficiency and integrity. The resulting logical schema directs the storage and retrieval of data and acts as a design for the database’s physical implementation. The goal of logical design is to provide a thorough and understandable representation of the data model, which serves as the foundation for developing the database system itself.

## **Normalization**

Normalization is a procedure in database design that organizes data effectively and reduces redundancy. It entails organizing the database tables in such a way that data dependencies are minimized. The basic purpose of normalization is to ensure data integrity, which refers to the accuracy and consistency of data recorded in the databases. Normalization contributes to the reliability and manageability of a database by eliminating or minimizing data redundancy and abnormalities.

Normalized relations, which are essential to the database’s design, are tables designed in accordance with normalization principles to decrease redundancy and ensure data integrity. This approach, essential for maintaining consistency, involves efficiently structuring data to eliminate anomalies during operations such as updates or deletions. By following normalization principles, database designers ensure that each table reflects a distinct entity, allowing for more effective data management and querying. In essence, normalized relations promote a well-organized and streamlined database design, which improves the system’s overall stability and efficiency.

# **Normalized Relations**

TCustomerType (CusTTypeID, CusTTName)

TState(StateID, StateName)

TSize (SizeID, SizeName)

TColor (ColorID, ColorName)

TPattern (PatternID, PatternName)

TSalesChannel (SCID, SCName)

TPayType (PTID, PTName)

TDiscount (DiscountID, DiscountType)

TEmpType (EmpTypeID, EmpRole)

TCarrierType(CTID, DeliveryType, TrackingNumber)

TCustomer (CustID, CCusTTID\*, CFName, CLName, Phone, Address, Street, City, CStateID\*, Zip, Email)

Foreign Key CCusTTID references TCustomerType

Not Null

On Delete Restrict

Foreign Key CStateID references TState

Not Null

On Delete Restrict

TEmployee (EmpID, EFirstName, ELastName, EAddress, ECity, EZip, EPhone, EEmail, Skill, EmpTypeID\*)

Foreign Key EEmpTypeID references TEmpType

Not Null

On Delete Restrict

TPaymentIn (PayInID, PIEmpID\*, PIDate, Amount, PIPTID\*)

Foreign Key PIEmpID references TEmployee

Not Null

On Delete Restrict

Foreign Key PIPTID references TPayType

Not Null

On Delete Restrict

TPaymentOut (PayOutID, Invoice#, Amount, PODate, POPTID\*)

Foreign Key POPTID references TPayType

Not Null

On Delete Restrict

TPurchaseOrder (POID, POEmpID\*, POPayOutID\*, PODate, Total)

Foreign Key POEmpID references TEmployee

Not Null

On Delete Restrict

Foreign Key POPayOutID references TPaymentOut

Not Null

On Delete Restrict

TDeliveryIn (DELIID, DEmpID-Deliver\*, DEmpID-Inspect\*, DEmpID-Receive\*, DDate)

Foreign Key DEmpID-Deliver references TEmployee

Not Null

On Delete Restrict

Foreign Key DEmpID-Inspect references TEmployee

Not Null

On Delete Restrict

Foreign Key DEmpID-Reciev references TEmployee

Not Null

On Delete Restrict

TDeliveryOut(DELOID, DOEmpID-Pick\*, DOEmpID-Pack\*, DOEmpID-Ship\*, DOCTID\*, ShipDate)

Foreign Key DOEmpID-Pick references TEmployee

Not Null

On Delete Restrict

Foreign Key DOEmpID-Pack references TEmployee

Not Null

On Delete Restrict

Foreign Key DOEmpID-Ship references TEmployee

Not Null

On Delete Restrict

Foreign Key DOCTID references TCarrierType

Not Null

On Delete Restrict

TSalesOrder(SOID, SOPayInID\*, SOCustID\*, SOEmpID\*, SODate, Total, SODiscountID\*, SOSCID\*)

Foreign Key SOPayInID references TPaymentIn

Not Null

On Delete Restrict

Foreign Key SOCustID references TCustomer

Not Null

On Delete Restrict

Foreign Key SOEmpID references TEmployee

Not Null

On Delete Restrict

Foreign Key SODiscountID references TEmployee

Not Null

On Delete Restrict

Foreign Key SOSCID references TSalesChannel

Not Null

On Delete Restrict

TProduct (ProdID, PName, PSizeID\*, PColorID\*, PPatternID\*, Total)

Foreign Key PSizeID references TSize

Not Null

On Delete Restrict

Foreign Key PColorID references TColor

Not Null

On Delete Restrict

Foreign Key PPatternID references TPattern

Not Null

On Delete Restrict

TBundleBOM (BundleID, BProdID-Kit\*, BProdID-Part\*, Quantity)

Foreign Key BProdID-Kit references TProduct

Not Null

On Delete Restrict

Foreign Key BProdID-Part references TProduct

Not Null

On Delete Restrict

TTrip(TripID, TSOID\*, TEmpTypeID\*, NoOfTravelers, DepartureDate)

Foreign Key TSOID references TSalesOrder

Not Null

On Delete Restrict

Foreign Key TEmpTypeID references TSalesOrder

Not Null

On Delete Restrict

TVendor(VendID, VEmpID\*, VProdID\*, Name, Address, City, VStateID\*, Zip, Phone, VendReliablityRank)

Foreign Key VEmpID references TEmployee

Not Null

On Delete Restrict

Foreign Key VProdID references TProduct

Not Null

On Delete Restrict

Foreign Key VStateID references TState

Not Null

On Delete Restrict

T<PurchaseOrderLine>(POLID, POLPOID\*, POLProdID\*, Quantity, PurchaseCost)

Foreign Key POLPOID references TPurchaseOrder

Not Null

On Delete Restrict

Foreign Key POLProdID references TProduct

Not Null

On Delete Restrict

## **Differences between ERD and Normalized Relations**

Entity-Relationship Diagrams (ERDs) and normalized relations differ primarily in how they depict database systems and serve different objectives. ERD’s are essentially conceptual tools that give an overall overview of a database’s structure by visualizing the entities, attributes, and relationships inside it. Normalized relations, on the other hand, represent the finer points of the database’s implementation; they are real database tables organized in accordance with normalization standards. While normalized relations delve deeply into the details and provide a clear depiction of individual tables, ERDs just provide us with an overview of the database. Normalized relations ensure that the database functions properly by reducing redundancy and maintaining data integrity by adhering to normalization rules. Overall, ERDs assist us understand the structure of the database, whereas normalized relations provide a practical blueprint for database creation and management.

# **Referential Integrity**

Referential integrity guarantees the accuracy of data and relationships in a database. When properly implemented, it ensures that all references between the tables are correct. This integrity constraint generates associations across entity types by correlating a foreign key from one table to a primary key from another, or it can be null if the mandatory side is optional. These constraints must be met to maintain correct connections between tables, confirming they are always present. They are essential to relational databases as they construct rules that regulate relationships, so preserving the database’s integrity and usability. Without referential integrity, relationships between entities would break down, resulting in a disconnected database incapable of properly linking data.

# **Physical Design and Implementation**

The step of physical design involves connecting each table to the database and storing physical design data in the database. This step begins with the establishment of a data dictionary that explains the purpose of every field. During this procedure, we analyze both physical and the operations to guarantee that the database runs properly. A database developer can choose from a variety of platforms to create the physical design on; in our example, we used: MSSQL.

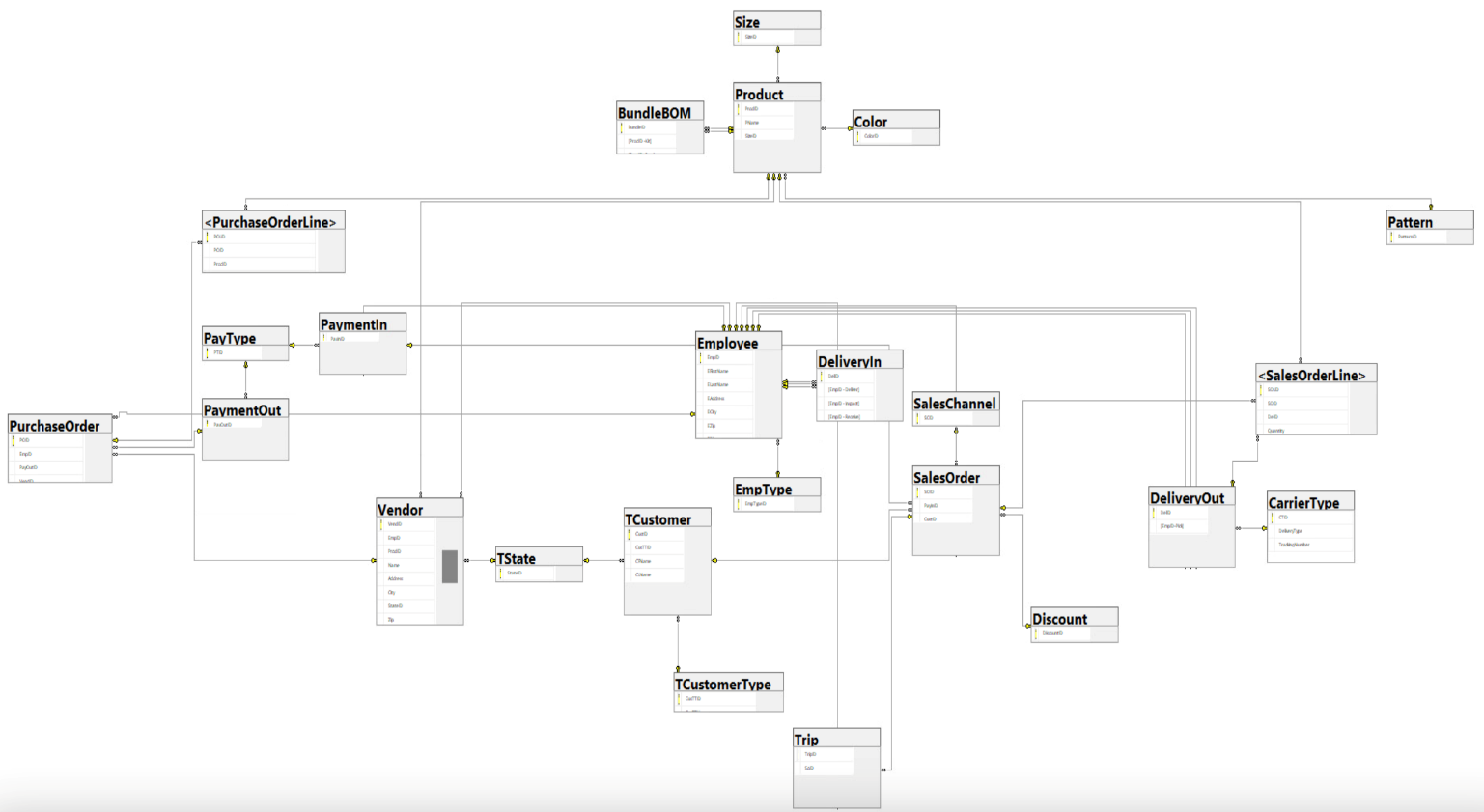
# **Data Dictionary**

A comprehensive reference table for the whole database is the data dictionary. It aids in maintaining the database’s credibility and in explaining how fields are used. For database administrators, developers, and users, the data dictionary is an essential tool that promotes effective data management, comprehension of data structures, and data integrity. The following below is an example of the data dictionary.

# **Denormalization**

Denormalization speeds up database retrieval by consolidating redundant data into a single table, resulting in improved read performance. Denormalization can also indicate that we reduce atomicity in a field. It is primarily used to reduce the time required for queries to run, as more tables referenced make the query take longer to be completed. For a further explanation and clarification, Denormalization could entail adding state information directly to the TCustomer table through adding the StateID from TState. From that statement, by denormalization, we can include the StateName directly in TCustomer. The solution is denormalization, which streamlines queries that include customer state information, potentially boosting query performance by decreasing the need for joins, therefore having faster performance.

## **Implemented Physical Design**



# **Challenges Faced/Addressed During Implementation**

Getting the correct generated data implemented for our tables was a major challenge while setting up our MSSQL database. To find what we needed, we had to refine our queries and investigate a variety of data sources. Trial and error were the procedure, but it became clear how crucial it is to use MSSQL wisely to increase data precision and ensure the success of our implementation.

Initially, we used a Sub/Super-Type structure for the Employee Entity. However, with additional assistance, we shortened the procedure by eliminating the Sub/Super-Type and instead creating a reference table based on the Employee Entity. This reference table uses EmpTypeID as the primary key, along with the attribute Role, for a more straightforward and efficient method.

One more difficulty we ran into was with debugging. When trying to enter the sample data, we ran into problems, which meant we had to keep looking at error messages and figuring out where they were coming from. This procedure required close inspection to find and fix mistakes as soon as possible. Thankfully, persistence allowed us to identify the typical errors, reducing their likelihood in the future and optimizing our debugging procedure.

# **Strengths and Weaknesses Encountered During Implementation**

Our team continuously showed that we could solve problems well while implementing our database. When we found mistakes in our code, we worked together right away to identify the problems and fix them so that everything ran smoothly. Furthermore, our team’s tenacity and cooperation were apparent at every stage of the procedure. Even though we originally faced many obstacles, we came together and took a step back to fully comprehend the underlying reasons for our problems and methodically address them. One weakness we did see, though, was a frequent unwillingness to speak up. There have been times when team members have been reluctant to confess that they were unfamiliar with tasks. However, as our friendship grew stronger, we encouraged an atmosphere of transparency where anyone felt free to ask questions and get help from the team. For instance, we had trouble finding our trip table when building the actual design that was put into practice. Rather than letting this error continue, we brought the addressed problem up early and cooperated to make sure each part was accurately portrayed in the implemented physical design.

# **Specific SQL Statements Requested**

In the following sections, we will carefully outline the specific programs that the client has commissioned us to perform within the database. We have also created additional questions that we believe are very helpful for Elysian’s monitoring needs, along with detailed justifications of their possible advantages. Included are tables detailing the client’s demands, the necessary SQL code to carry them out, and sample photos showing the results of these programs.

|  |  |  |  |
| --- | --- | --- | --- |
| **Query #** | **Question** | **SQL** | **Partial Output** |
| 1 | Total sales (in dollars) by customer state per year. | Select SUM(so.Total) as total\_sales, s.StateName  from TState s join TCustomer c on s.StateID = c.StateID join SalesOrder so on c.CustID = so.CustID  where SODate between '2023-4-28' and '2024-4-28'  group by s.StateName |  |
| 2 | Total sales (in dollars) by vendor per year. We must Be able to calculate profit (sale price – purchase price). | SELECT v.vendid, v.VName, YEAR(so.SODate) AS sale\_year, SUM(p.[Sales Cost]) AS total\_sales, SUM(p.[Sales Cost] - p.[purchase cost]) AS total\_profit  FROM vendor v JOIN product p ON v.prodid = p.prodid join [<SalesOrderLine>] sol on p.ProdID = sol.ProdID join SalesOrder so on sol.soid = so.soid  GROUP BY v.vendid, v.Vname, YEAR(so.sodate); |  |
| 3 | The ten highest selling (in dollars) (a) patterns, (b) sizes, (c) pattern-size-color combinations each year. | Select top 10 PatternID, SizeID, ColorID, so.Total as Total\_sales  from product p join [<SalesOrderLine>] sol on p.ProdID = sol.ProdID join SalesOrder so on sol.SOID = so.SOID  group by PatternID, SizeID, ColorID, so.Total  order by Total\_sales Desc | Inserting image... |
| 4 | The number of times each product (fly) was sold. We want to see also those flies that have never been sold so that we can discontinue them. | select p.PName, COALESCE(COUNT(so.SOID), 0) As Num\_Times\_Sold  from Product p join [<SalesOrderLine>] sol on p.ProdID = sol.ProdID join SalesOrder so on sol.SOID = so.SOID  group by p.PName  order by Num\_Times\_Sold asc |  |
| 5 | Total sales (in dollars) for each channel per month. | Select sc.SCName, sum(so.Total) as Total\_sales  from SalesChannel sc join SalesOrder so on sc.SCID = so.SCID  where year(SODate) = 2024 and month(SODate) = 2  group by SCName  order by Total\_sales desc |  |
| 6 | 10% of the products that have the highest margin. | Select top 10 PERCENT p.ProdID, p.Pname as Product\_Name  From [<PurhcaseOrderLine>] pol join Product p on pol.ProdID = p.ProdID |  |
| 7 | The ten most popular (units sold) DIY fly-tying materials. | Select top 10 (pol.Quantity), p.ProdID, P.Pname as ProductName  From [<PurchaseOrderLine>] pol join Product p on pol.ProdID = p.ProdID  Where p.Pname like ‘%DIY%’  Order by pol.Quantity desc |  |
| 8 | The number of distinct products managed by each vendor manager. | SELECT e.EmpID, CONCAT(e.EFirstName, '' ,e.ELastName) as Employee\_name, COUNT(DISTINCT v.ProdID) AS distinct\_products\_managed  FROM Vendor v JOIN Employee e ON v.EmpID = e.EmpID join EmpType et on et.EmpTypeID = e.EmpTypeID  WHERE et.EmpRole = 'Vendor Manager'  GROUP BY e.EmpID, e.EFirstName, e.ELastName |  |
| 9 | The upcoming, scheduled guided trips (i.e., the guided trips that have already been sold) for each guide, including the guide’s name, the customer name, and the number in the customer’s party. | SELECT e.empID AS guide\_id, CONCAT(e.EFirstName, ' ', e.ELastName) AS guide\_name, CONCAT(c.CFName, ' ', c.CLName) AS customer\_name, t.nooftravelers  FROM Employee e JOIN EmpType et ON e.empTypeID = et.empTypeID JOIN Trip t ON e.empID = t.empID  JOIN SalesOrder so ON t.soid = so.soid JOIN TCustomer c ON so.custID = c.custID  WHERE et.EmpRole = 'Guide'  Group by: EmpID, |  |
| 10 | Number of trips and the number of customers taken on fishing trips by each guide in the past 6 months. | SELECT e.empID AS guide\_id, CONCAT(e.efirstName, ' ', e.elastName) AS guide\_name, COUNT(DISTINCT t.tripID) AS trips\_count,  COUNT(DISTINCT so.CustID) AS customers\_count    FROM Employee e JOIN Trip t ON e.empID = t.empID JOIN SalesOrder so ON t.soid = so.soid    WHERE t.TripType = 'Fishing'    GROUP BY e.empID, e.EFirstName, e.ELastName; |  |
| 11 | Names and email addresses of all customers who made purchases in a given month. We need to be able to enter the month. | SELECT CONCAT(c.CFName, ' ', c.CLName) AS customer\_name, c.CEmail AS customer\_email, s.Total  FROM SalesOrder s JOIN TCustomer c ON s.custID = c.CustID  WHERE MONTH(s.SODate) = 05 AND YEAR(s.SODate) = 2023  Group by c.CFName, c.CLName, c.CEmail, s.Total |  |
| 12 | Number of times used and dollars spent on each shipping vendor and shipping type by vendor. | SELECT COUNT (CT,CTID) AS TimesUsed, SUM(DO.ShipCost) AS MoneySpent  FROM DeliveryOut DO JOIN CarrierType CT ON CT.CTID = DO.CTID  GROUP BY CT.CTID |  |
| 13 | Invoice lines for a given sales invoice number and given customer name. | select pyo.Invoice#, CONCAT(c.CFName, ' ',c.CLName) as Customer\_Name  from TCustomer c join SalesOrder so on c.custid = so.CustID join PaymentIn py on so.PayInID = py.PayInID  join Employee e on py.EmpID = e.EmpID join PurchaseOrder po on po.EmpID = e.EmpID join PaymentOut pyo on pyo.PayOutID = po.PayOutID  group by pyo.Invoice#, CFName, CLName |  |
| 14 | A number of times a discount was applied to a sales order. | SELECT d.DiscountID, d.discounttype, COUNT(so.discountid) AS times\_discount\_applied  FROM salesorder so  JOIN discount d ON so.discountid = d.discountid  GROUP BY  d.discountid,  d.discounttype; |  |

## **Three Additional Queries**

A couple of additional questions that the customer did not specifically request but that would be useful to know the answers to for future operations are presented in the table below.

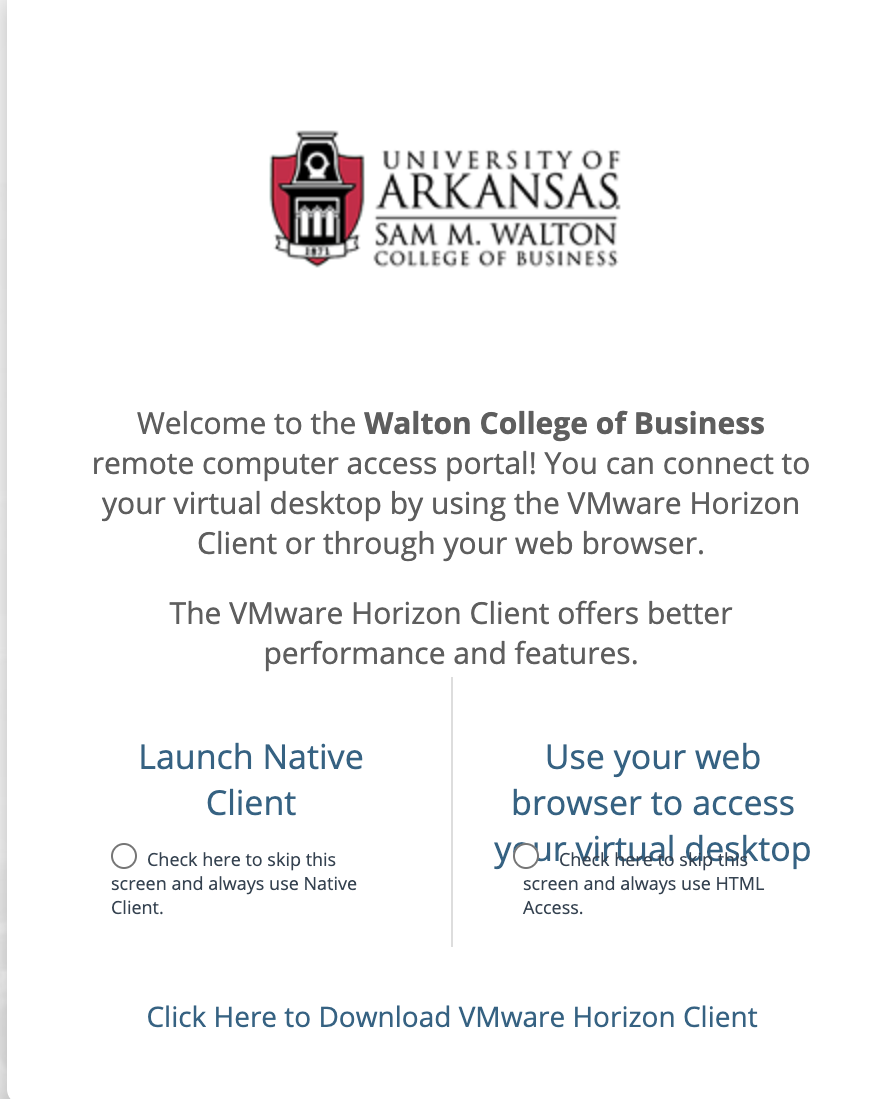
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Query # | Question | Why is this important | SQL | Partial Output | Recap of Findings |
| 1 | 10% of the products that have the lowest margin. | Businesses can identify opportunities for improvement by identifying the bottom 10% of items with the lowest margin. This allows for strategic adjustments to improve profitability and operational efficiency. | Select top 10 PERCENT p.ProdID, p.Pname as Product\_Name  From [<PurhcaseOrderLine>] pol join Product p on pol.ProdID = p.ProdID  Order by ProdID desc; |  | Wet flies are the product with the lowest profit margin, as indicated by the query. |
| 2 | The ten least popular (units sold) DIY fly-tying materials. | The business wants to be able to find out what are their ten least popular DIY fly-ties so they can maybe look at replacing getting rid of these items for products that will sell more. | Select top 10 (pol.Quantity), p.ProdID, P.Pname  From [<PurchaseOrderLine>] pol join Product p on pol.ProdID = p.ProdID  Where p.Pname like ‘%DIY%’  Order by pol.Quantity asc |  | After running the code, the least popular DIY materials sold between 22 and 6 times. |
| 3 | List the first and last names, along with the city, of customers who are locals. | Listing local customers and cities helps businesses determine where their core client base is located, allowing them to better respond to their preferences and needs. | Select C.CFName as FirstName, C.CLName as LastName, C.City as City, CT.CusTTName as CustomerType  From TCustomer C join TCustomerType CT on c.CusTTID = CT.CusTTID  Where CT.CusTTName like ‘%Local%’ |  | The majority of local clients are more than an hour away from Red Lodge. |

# **User Documentation**

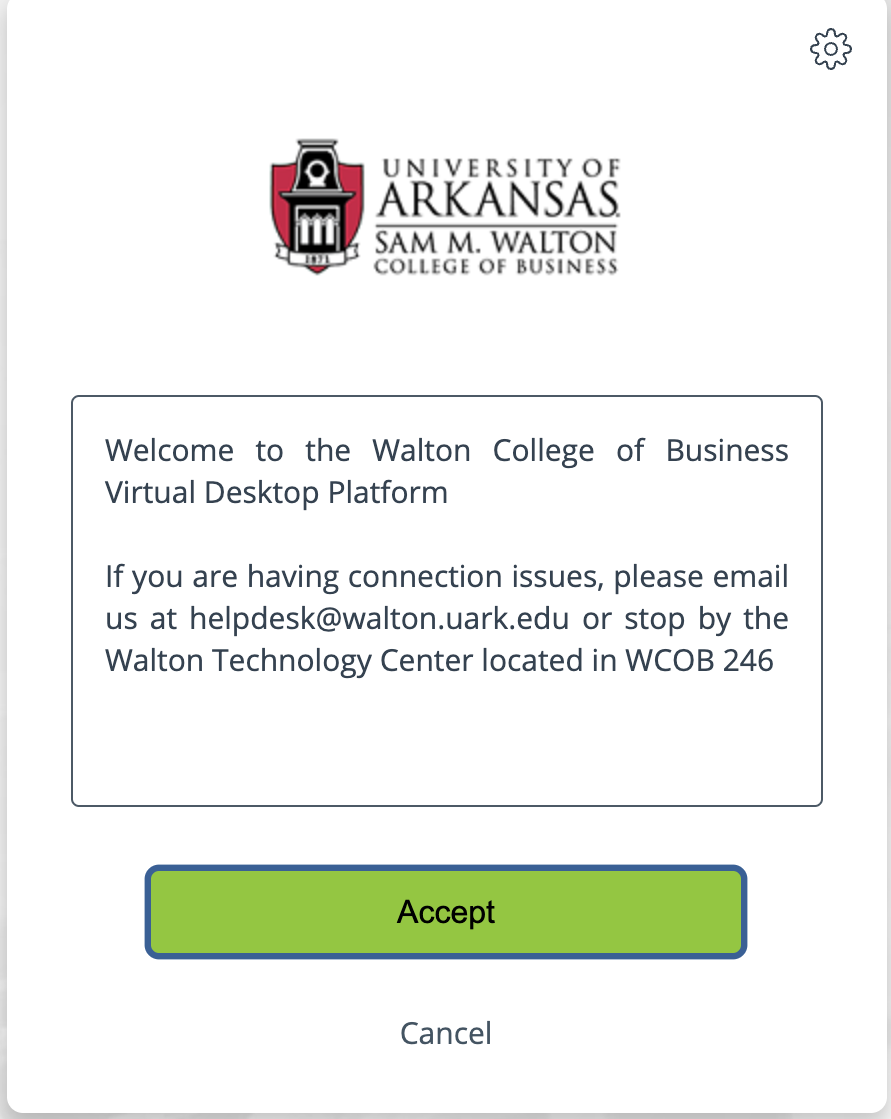
This virtual desktop is our go-to platform for database operations, reducing the load on our own PC’s. To get into the database we’ve been working on, take these steps:

1. Type <https://waltonlab.uark.edu/> into your search box and hit enter.

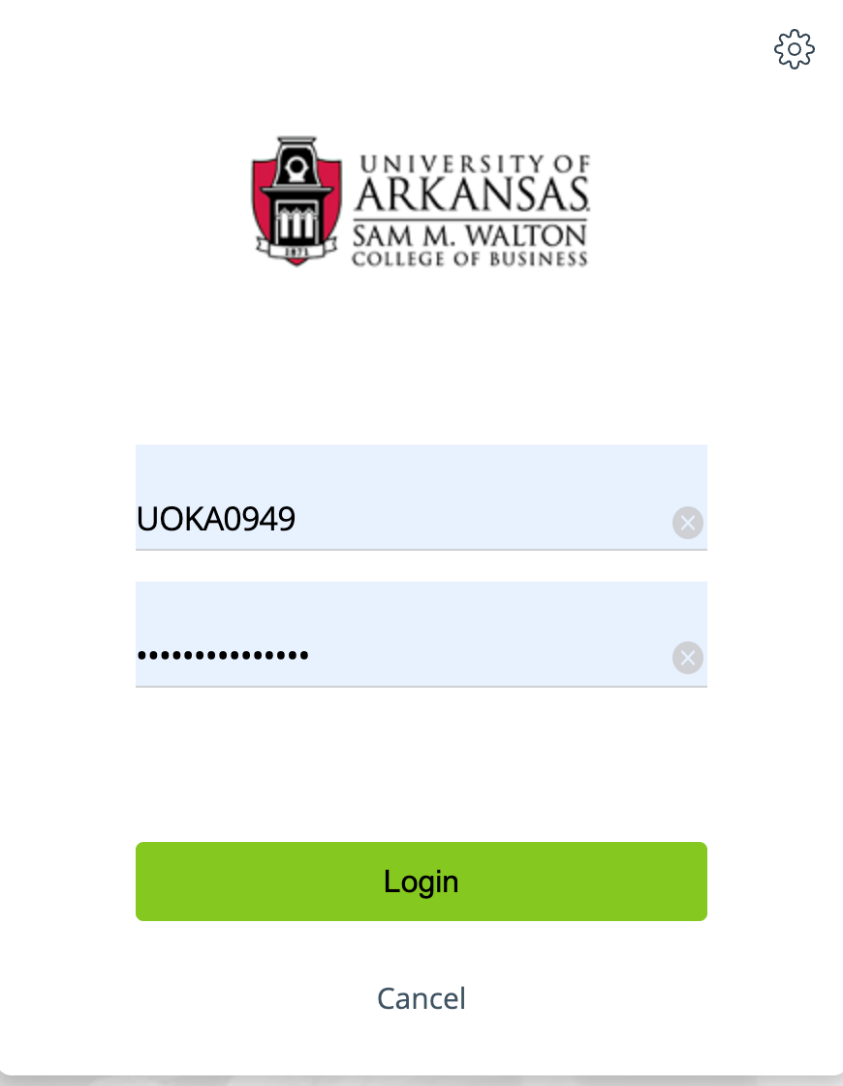
2. Choose the “Access Virtual Desktop with Web Browser” option.



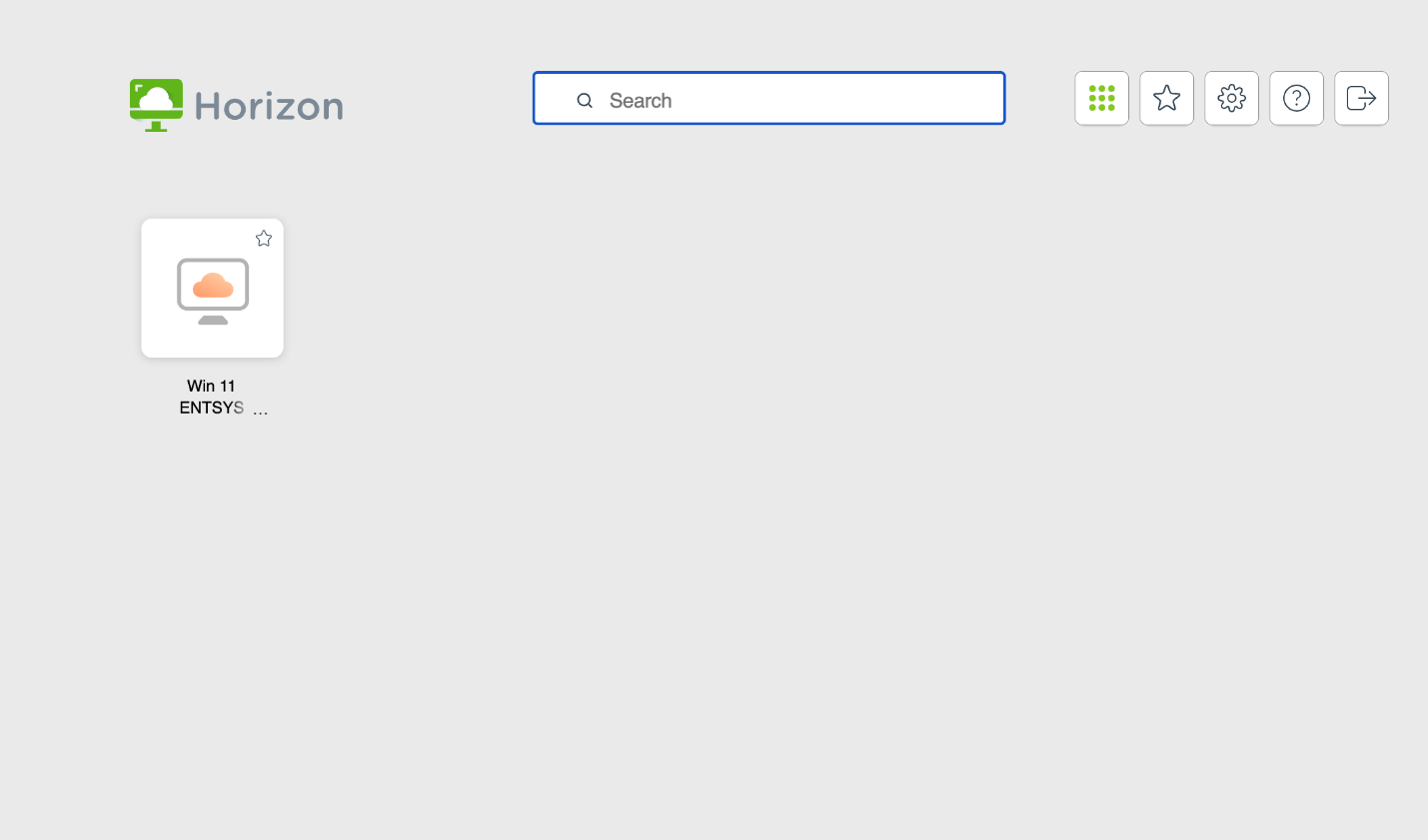
3. Select the “Accept” button.



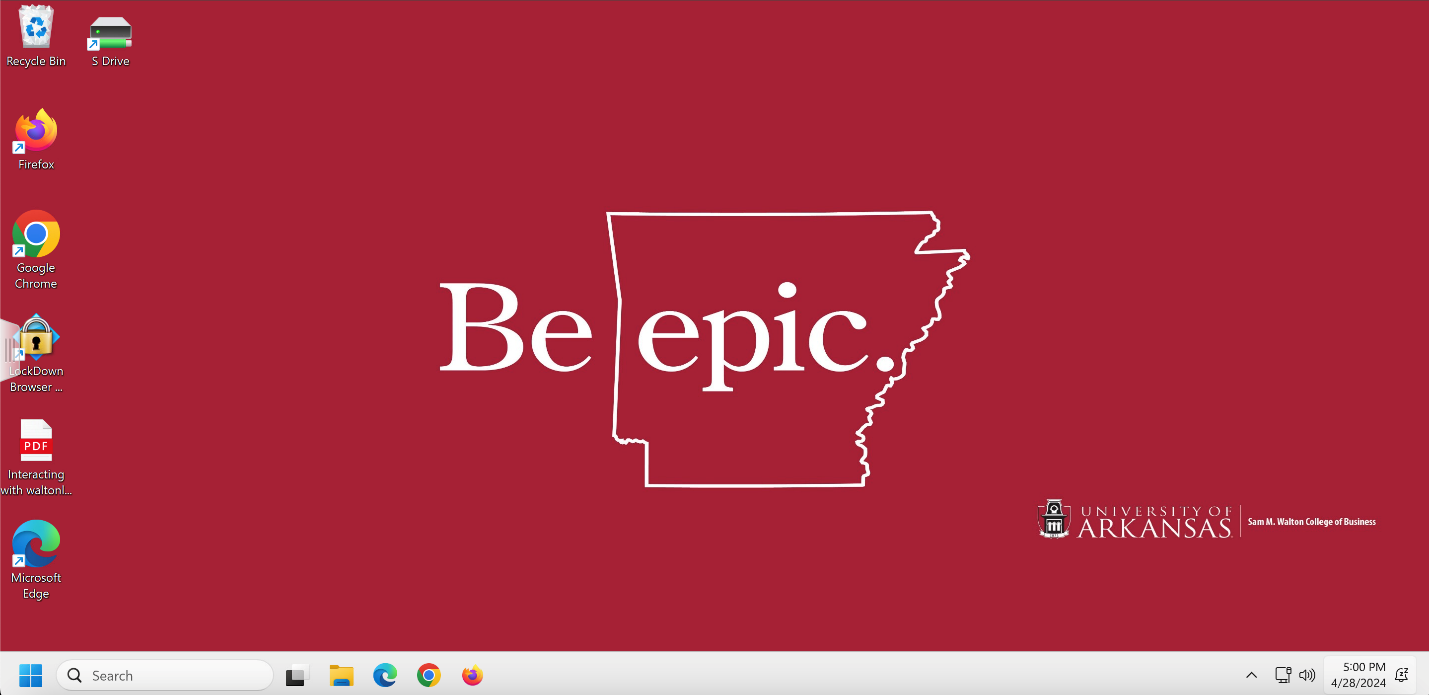
4. Kindly input the password and username you have provided. Take ours, for example, and click “Login”.

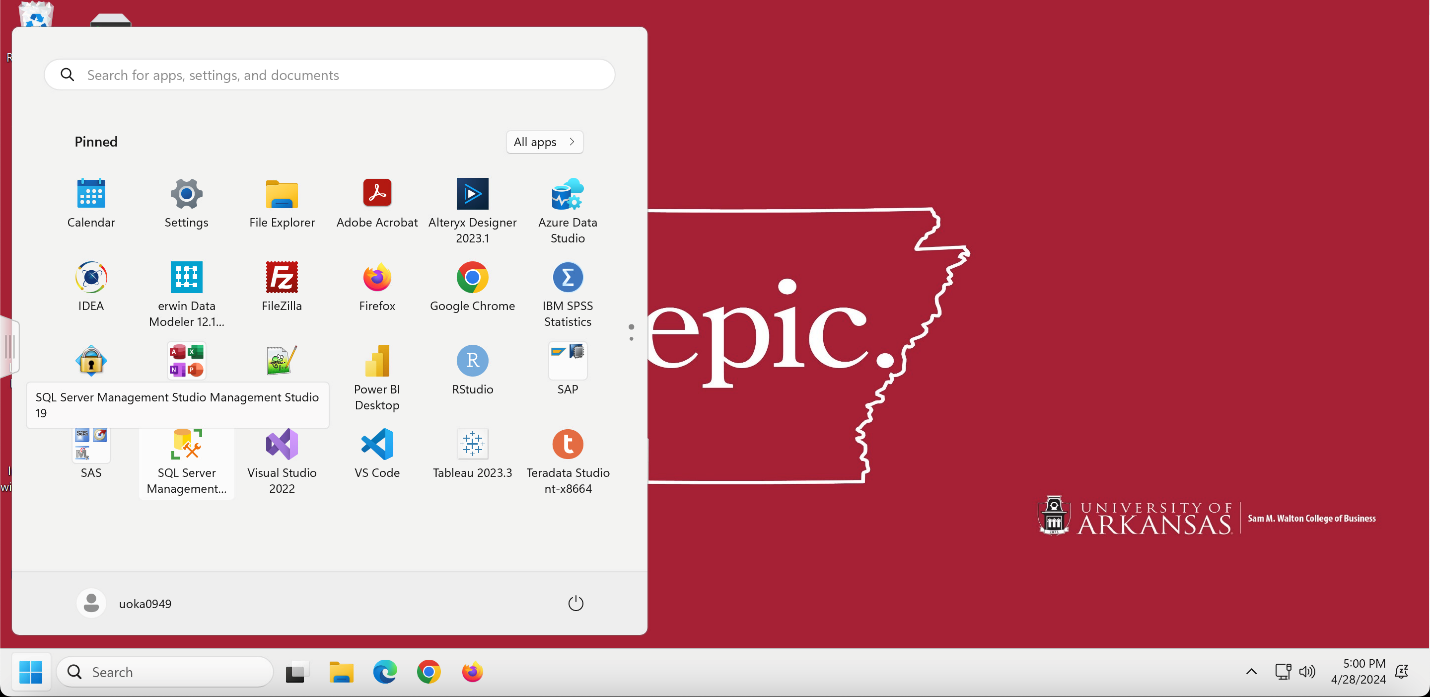


**5.** After logging in, select the “WIN 11 Enterprise Systems” monitor icon.

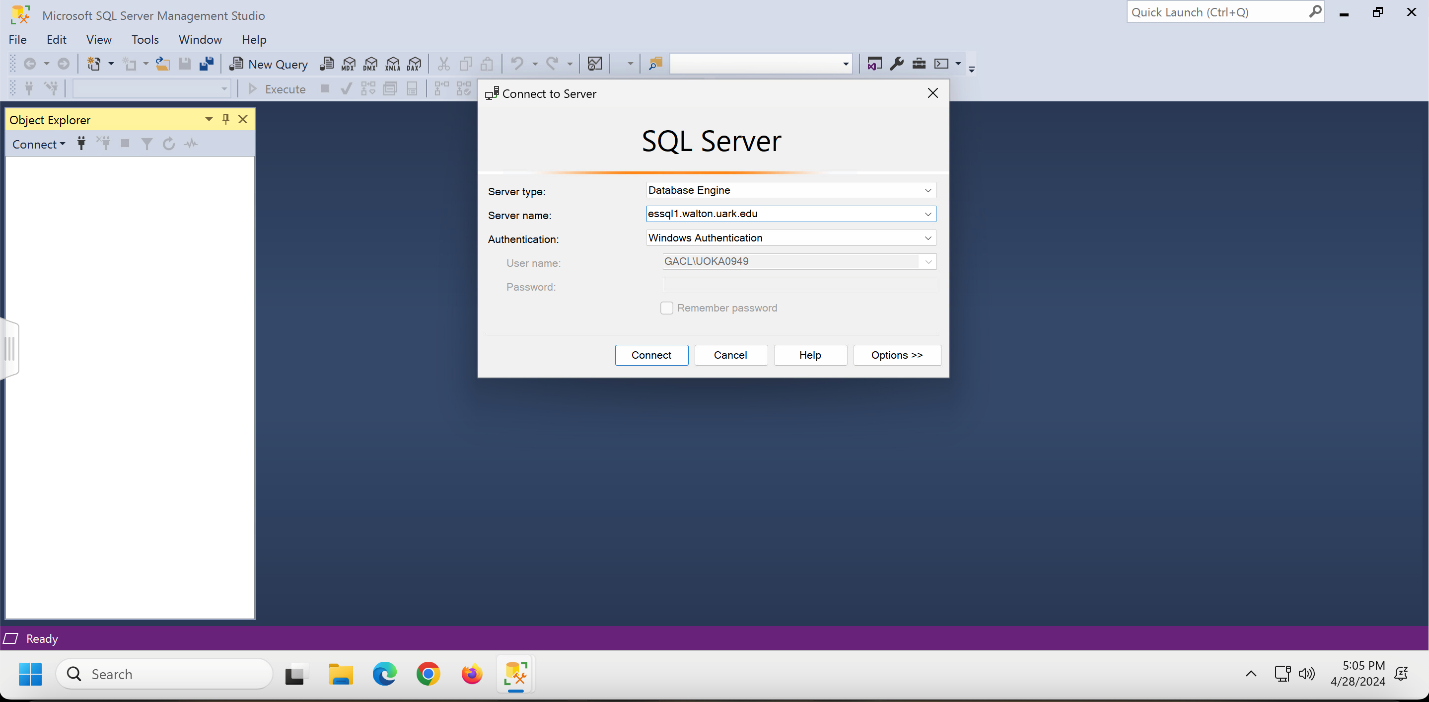


**6.** After that, you will be taken to the University of Arkansas’s virtual desktop. Please click the Windows icon in the lower left corner of the desktop screen, then choose the “Microsoft SQL Server Management Studio 19” program.



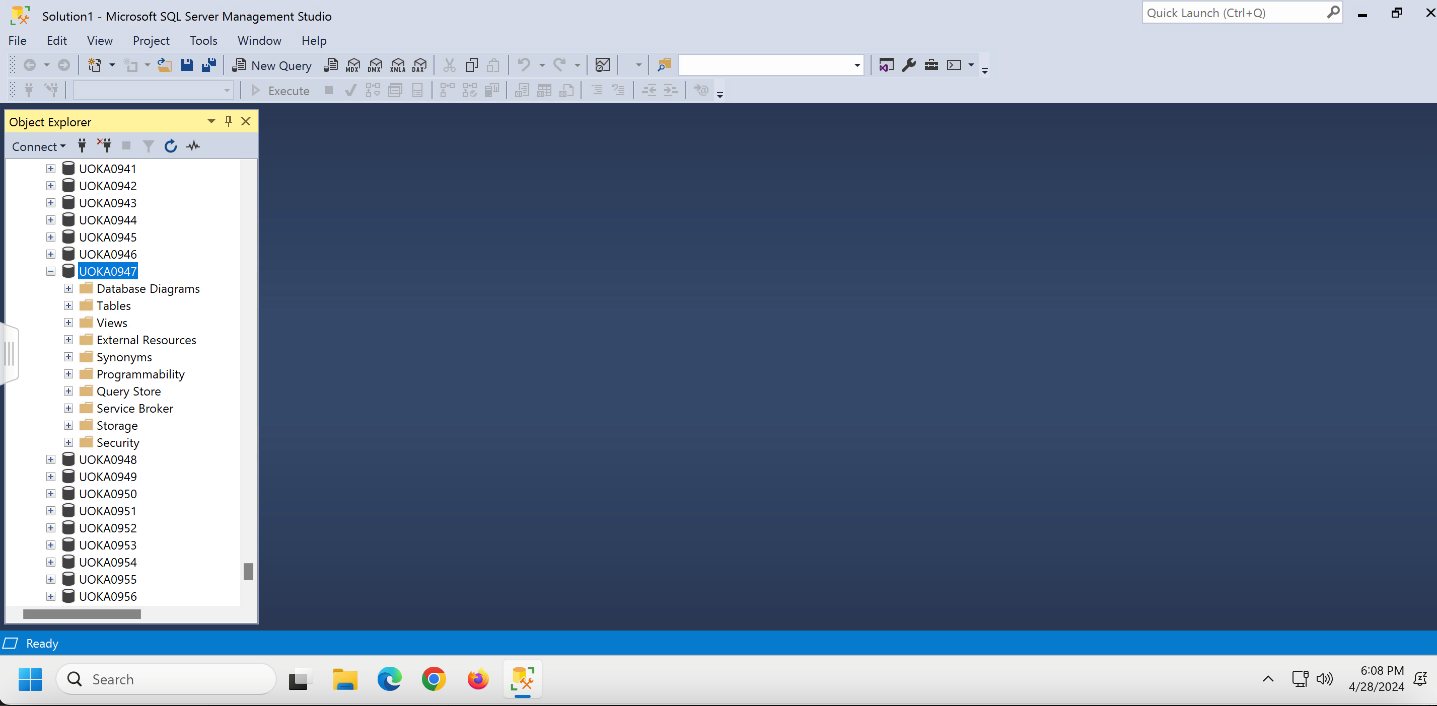


**7.** Once the correct server is selected, the application will load, displaying a window labeled “Connect to Server” and requesting some details to establish a connection. Verify that “Database Engine” is chosen in the “Server type” selection. Next, enter “essql1.walton.uark.edu” in the box labeled “Server name”. Choose “Windows Authentication” from the “Authentication” drop-down option. Once all the necessary data has been entered, please select “Connect.”

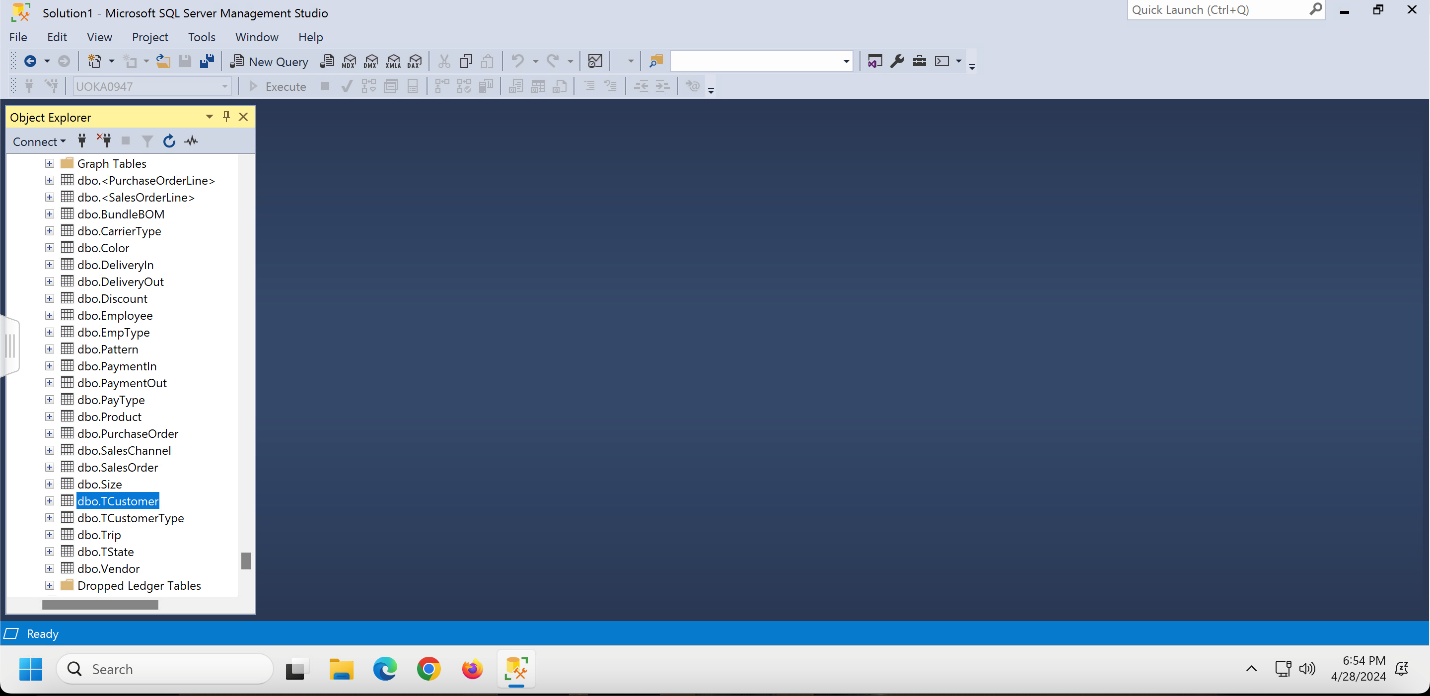


8. It’s time to locate the database that our team worked with now that you have access to the server! Click the “+” button next to the “Databases” folder in the Object Explorer to accomplish this. Please scroll all the way down to locate the database titled “UOKA0947” after the folder has expanded, then click the “+” to the left of it.

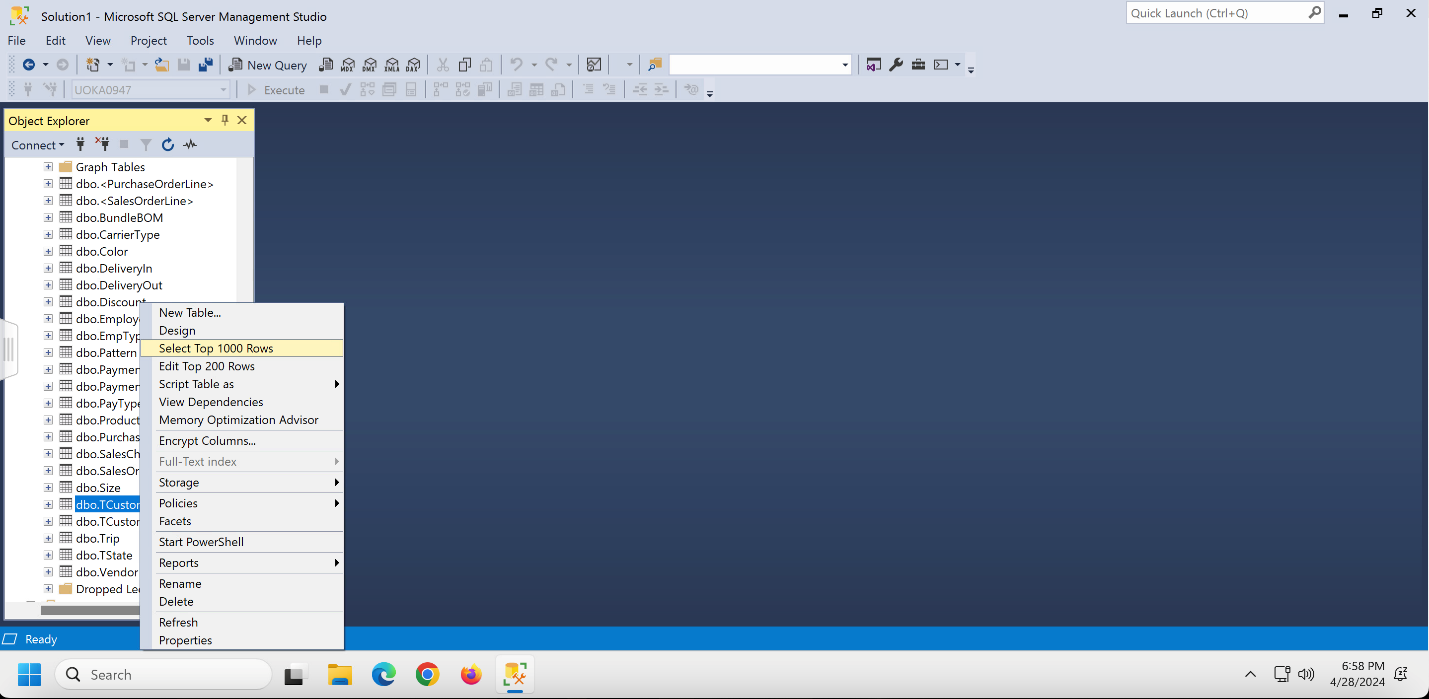


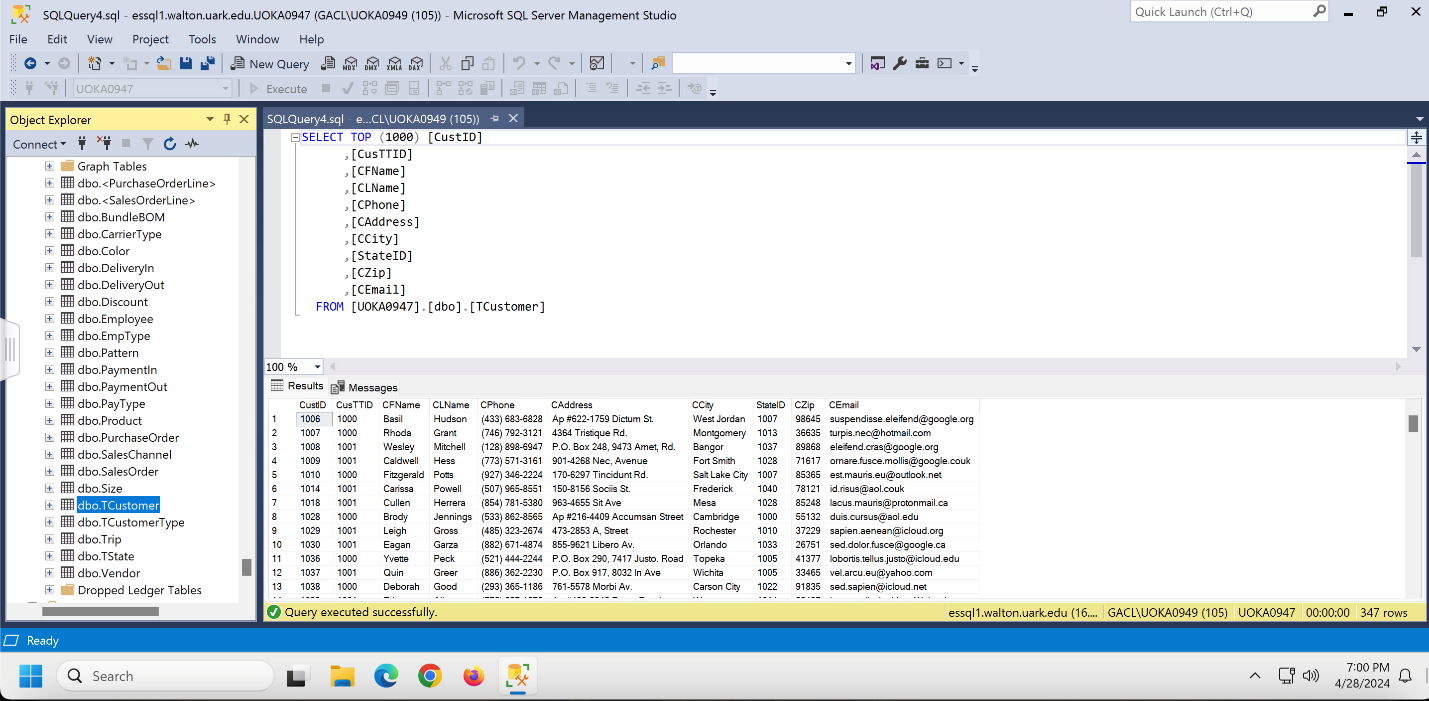


**9.** To see the tables we’ve generated for the database, simply click the “+” button next to the “Tables” folder. Here’s a list of the tables we have created.

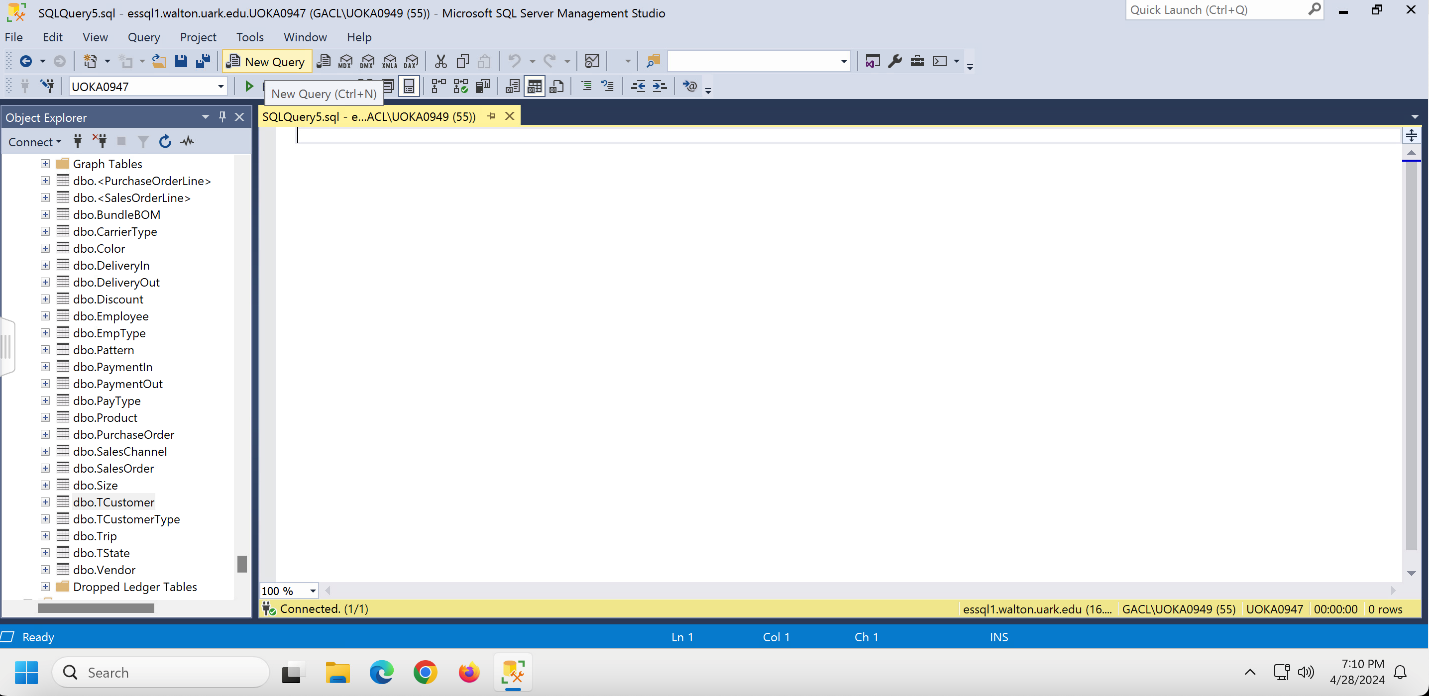


**10.** To view the columns and records in our tables, right-click on the appropriate table and choose “Select Top 1000 Rows”. For example, we’re using TCustomerTable.





**11.** For example, suppose you want to conduct the following query: Total sales (in dollars) by customer state per year. You just need to enter your SQL statement by selecting the “New Query” option at the top.



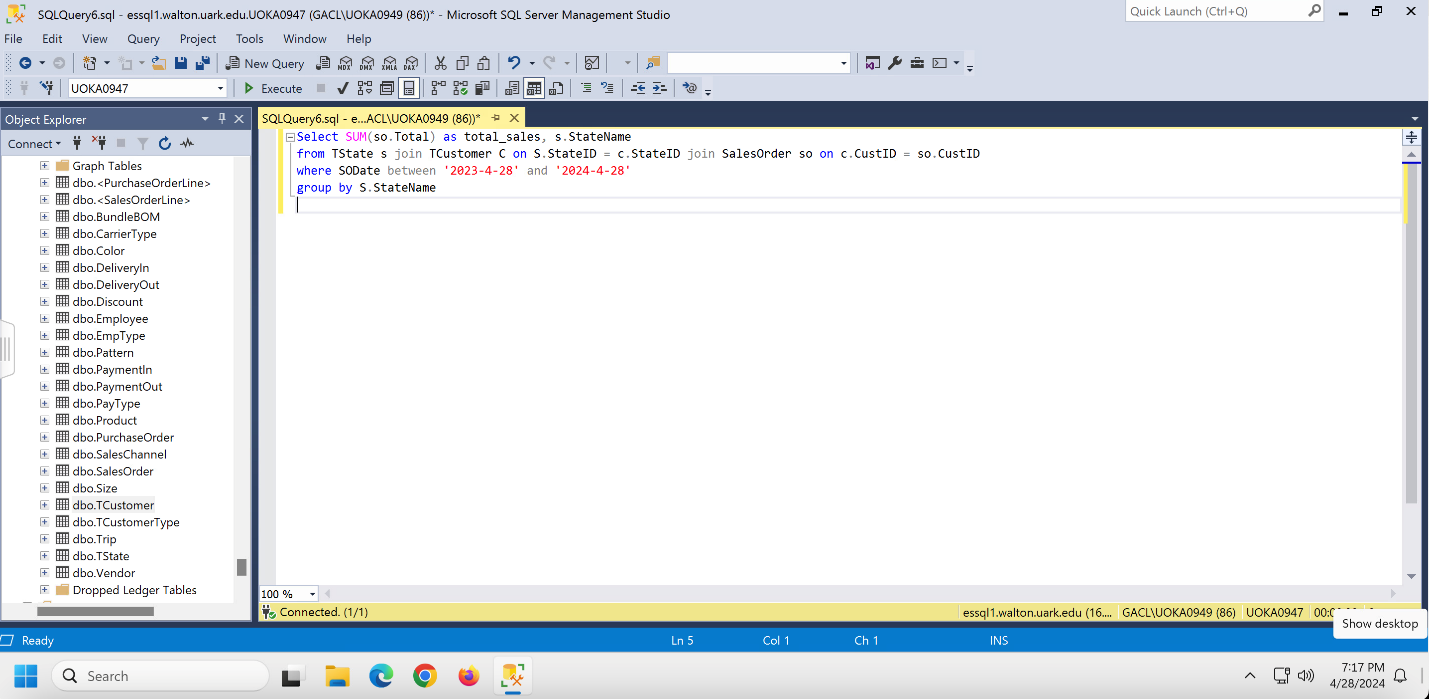
12. From that please input,

Select SUM(so.Total) as total\_sales, s.StateName

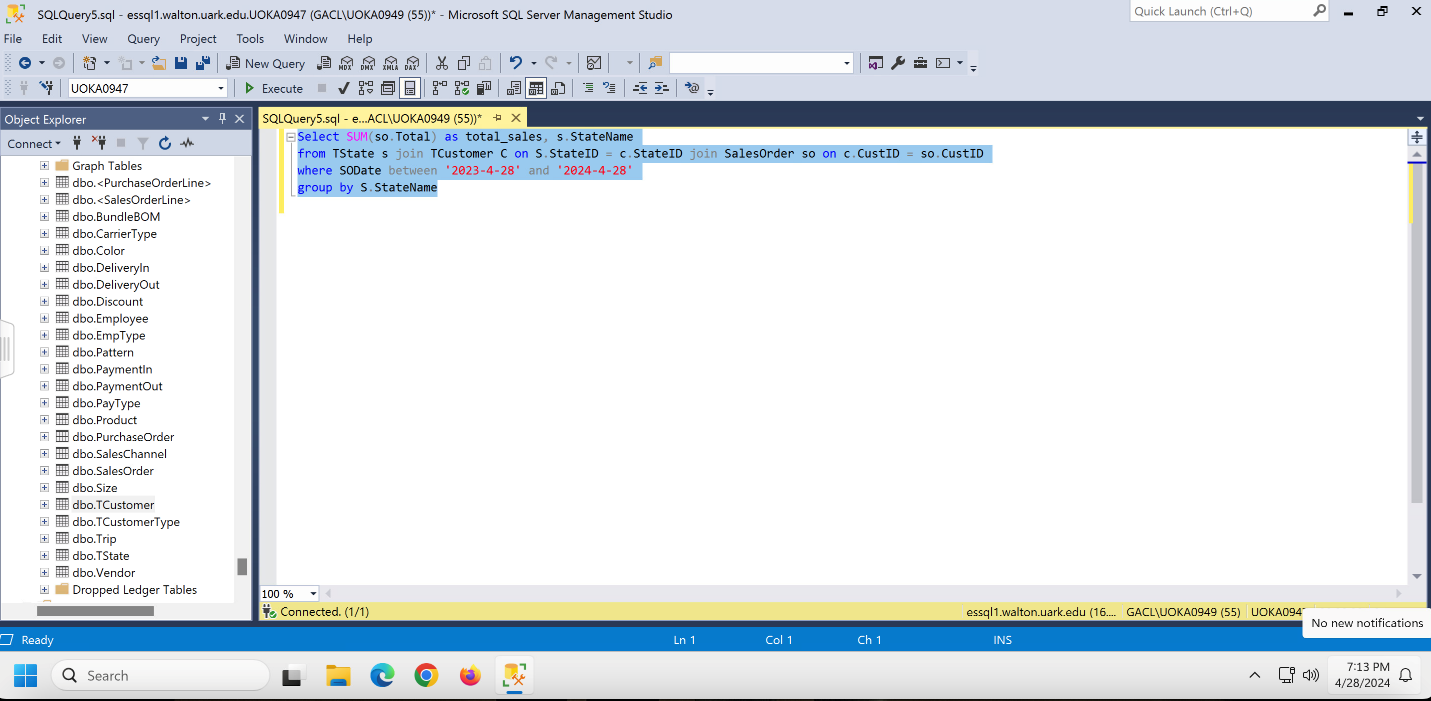
From TState s join TCustomer c on s.StateID = c.StateID join SalesOrder so on c.CustID = so.CustID

Where SODate between '2023-4-28' and '2024-4-28'

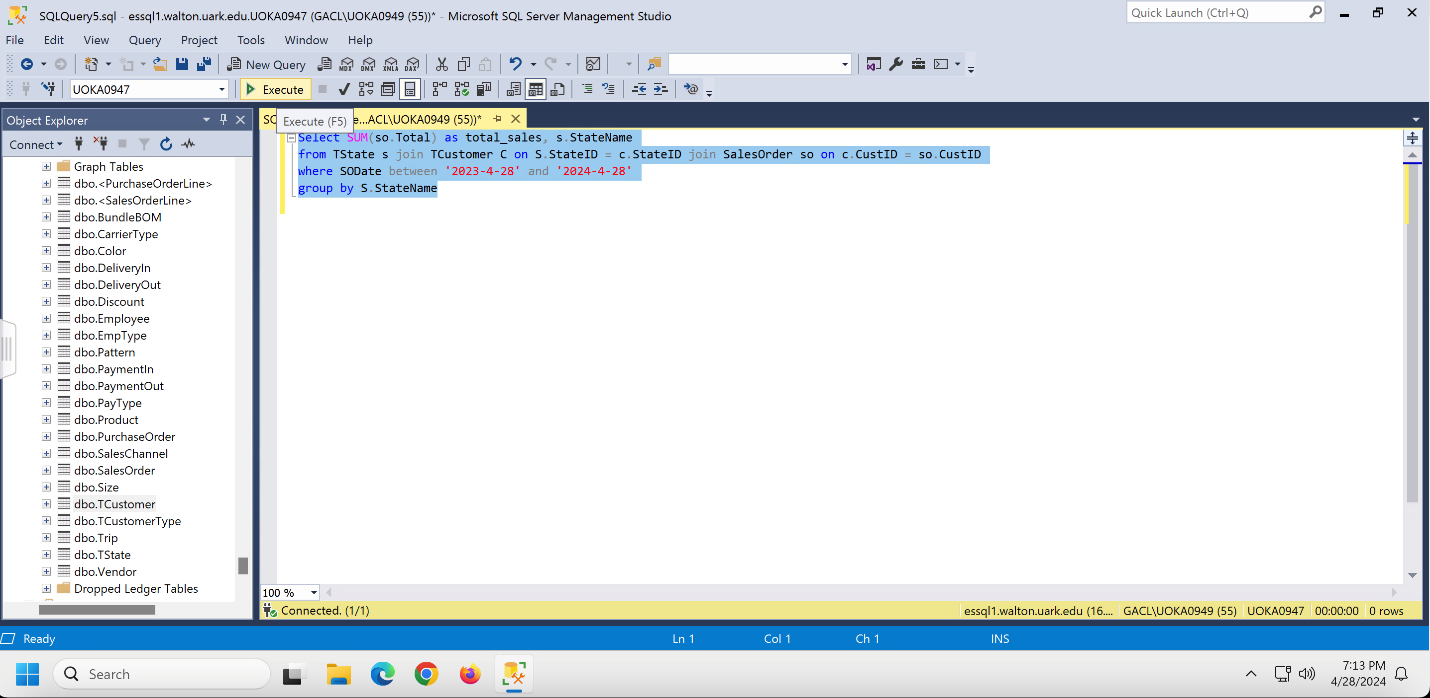
Group by s.StateName



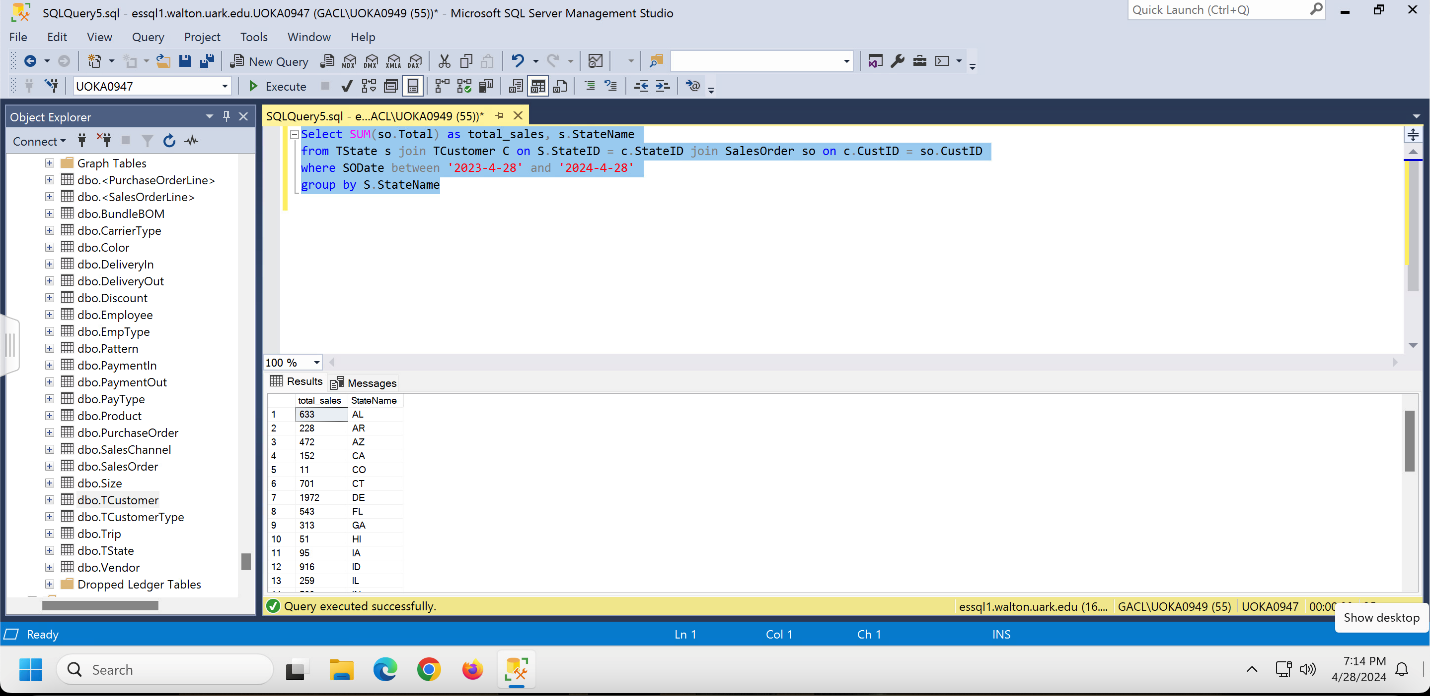
13. After that, kindly use Command A on a Mac or Control A on a PC to highlight everything.



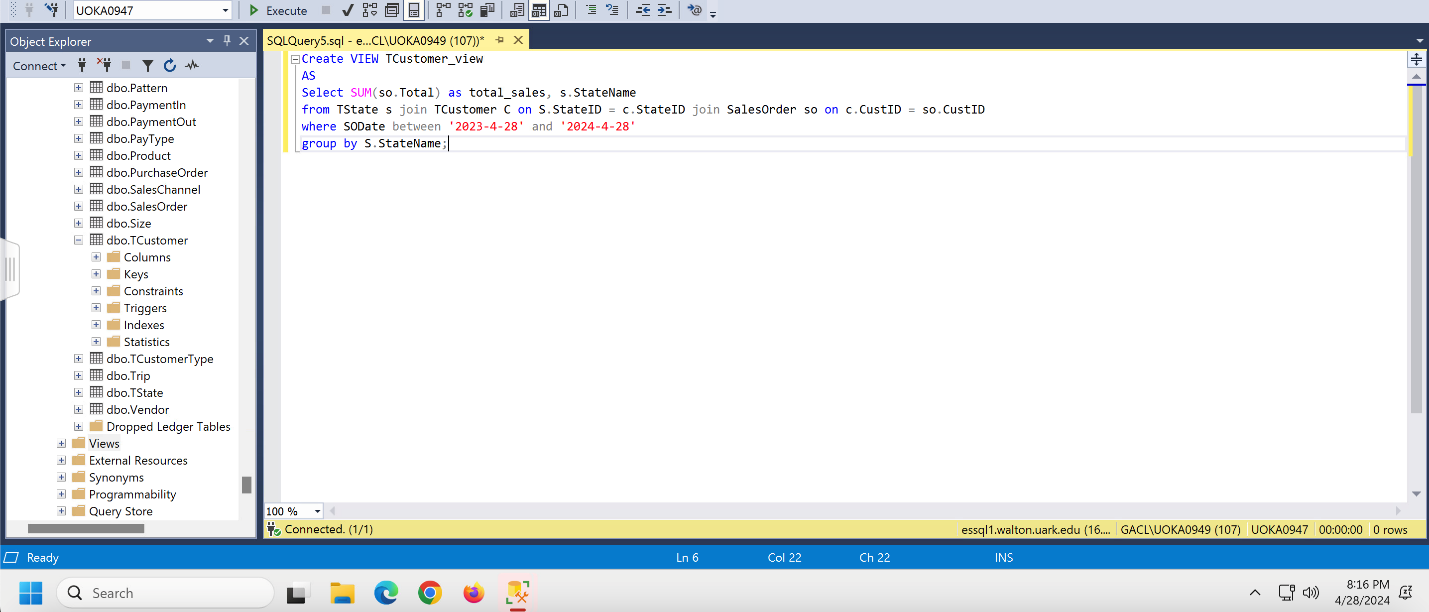
14. Once everything has been highlighted, please press Execute.

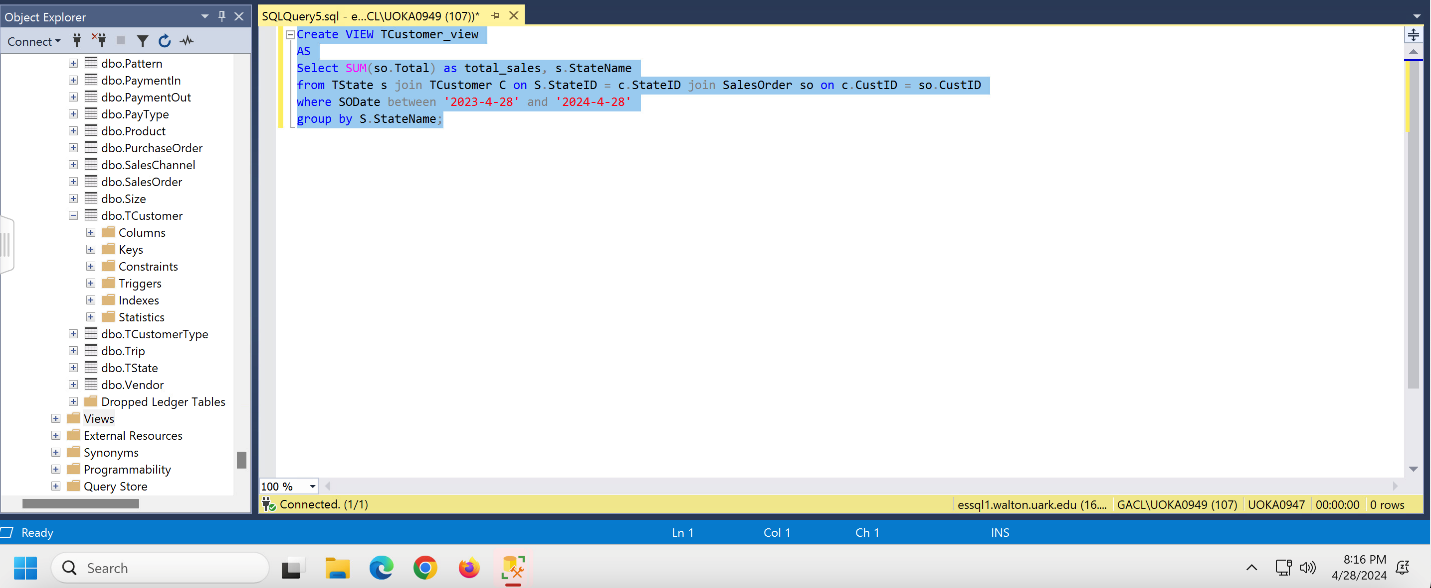


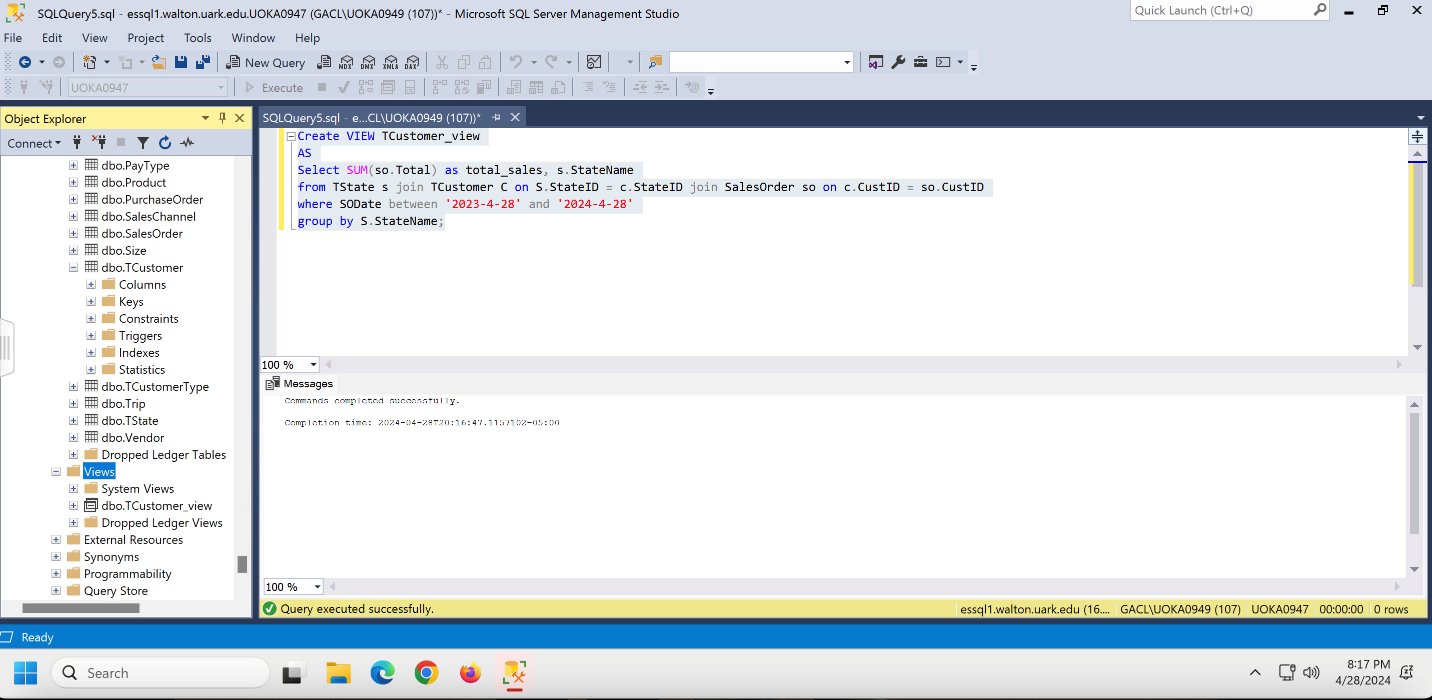
15. Ultimately, the data extracted from your query will appear at the bottom of the screen.



16. Write “Create View (TCustomer) As” before your Select statement, as seen in the highlighted area of the image below, to save your query. You will get the message “Commands completed successfully” after clicking Execute. The resultant data table may then be found stored as dbo.(TCustomer) in the Views folder.





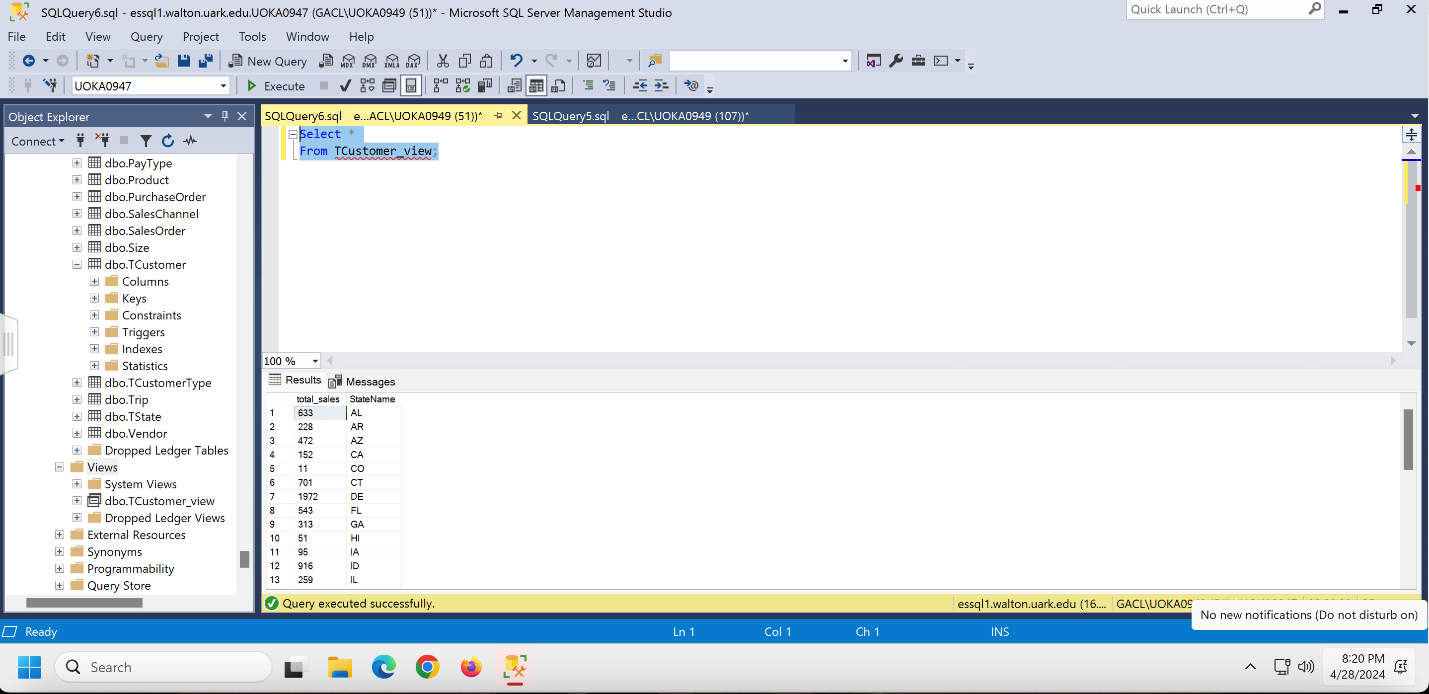


17. To execute your view, please open a new query and type the following to find the query again:

Select \*

From TCustomer\_view

Here you will find the exact same data.

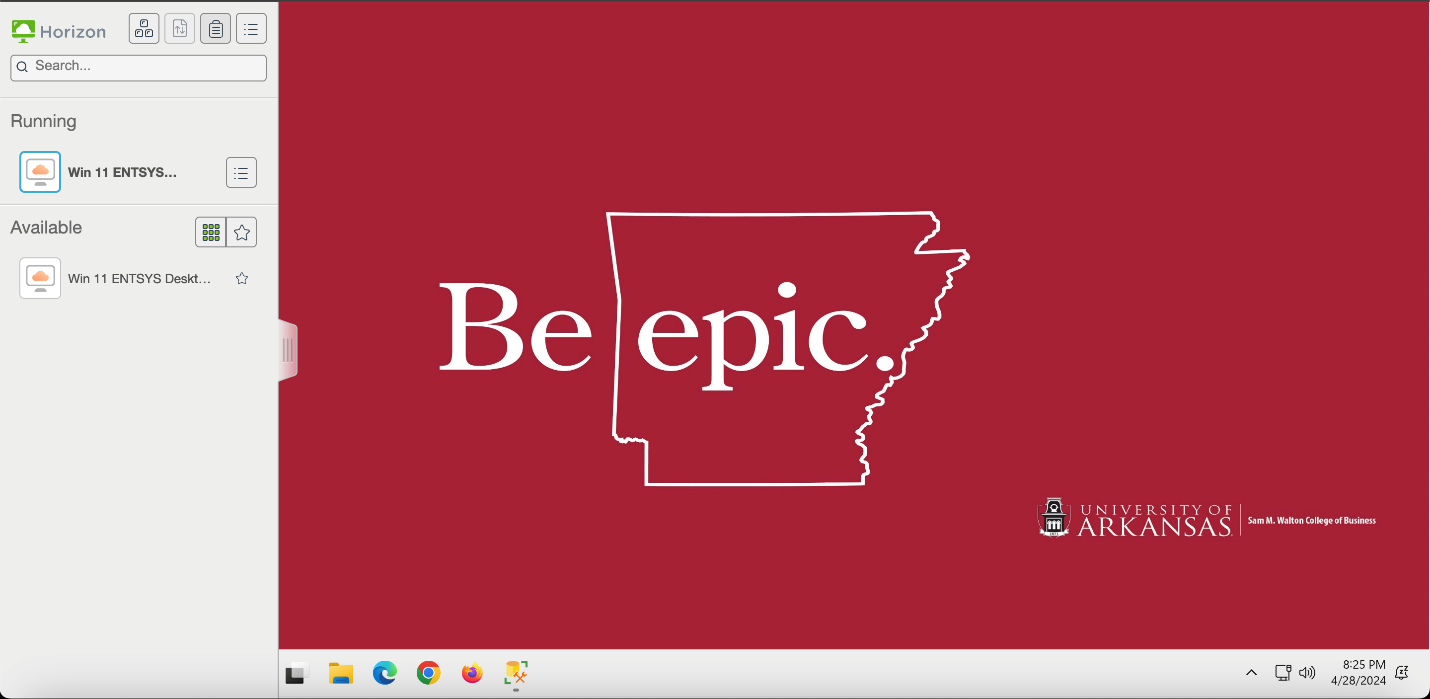


18. Well done!

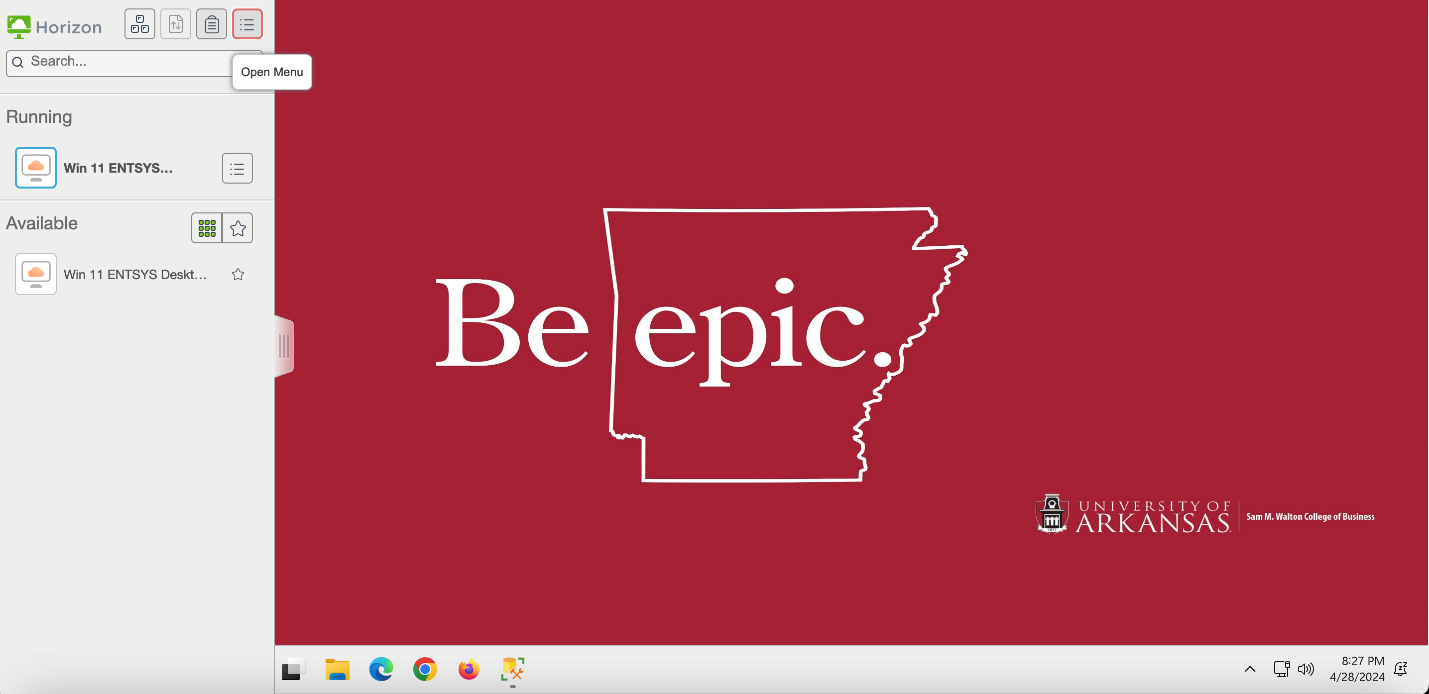
19. To Log Off, exit out of the application and please go back to the desktop:



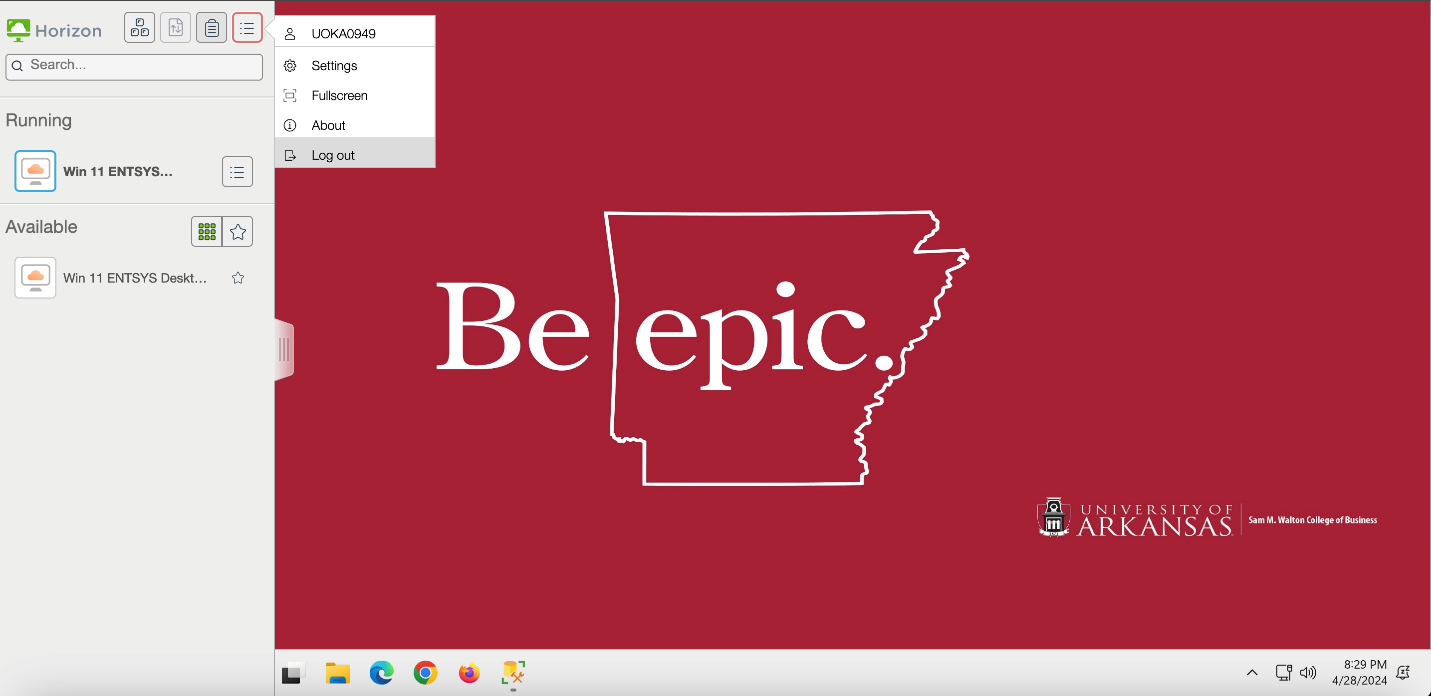
20. Please click the left-hand side it will prompt a window:



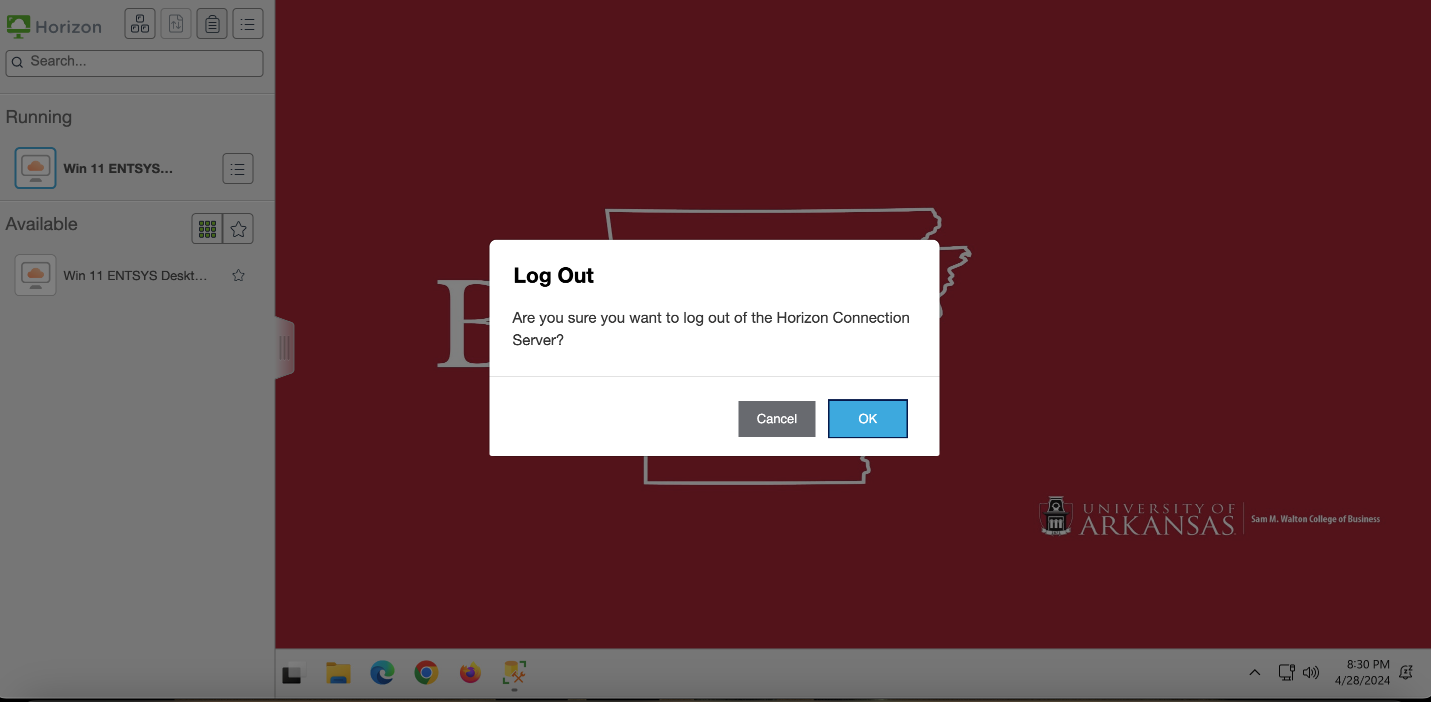
21. Simply select the “Open Menu” button located at the top of the display.



22. Please click “Log out”



23. Please click “OK”



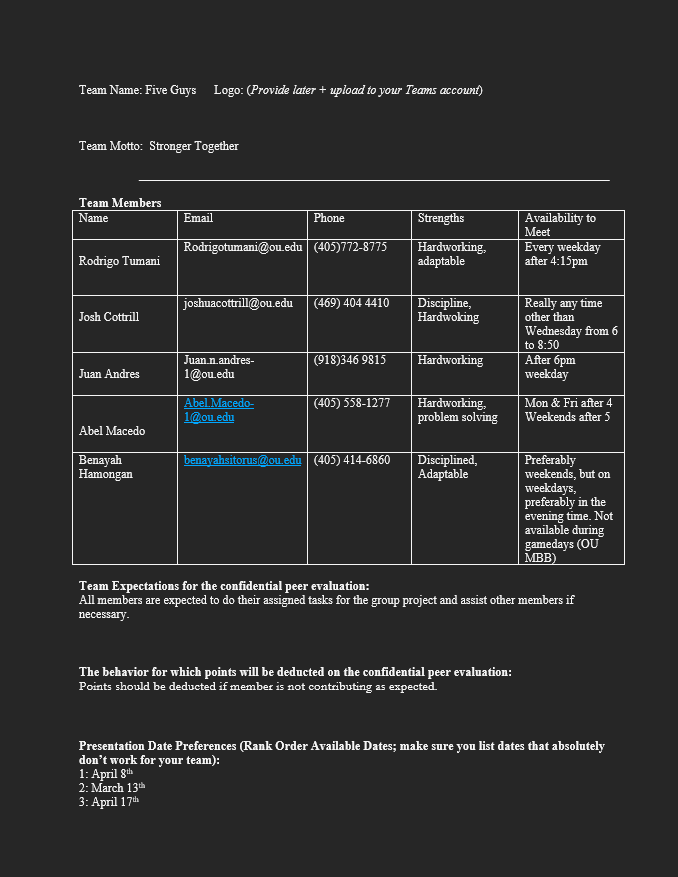
# **What We Learned Throughout This Process**

Throughout our project journey, we’ve gathered a wealth of information that has expanded our understanding of collaborative dynamics. We quickly realized the value of teamwork, realizing that combining different perspectives and contributions results in inventive ideas that surpass each person's abilities. Furthermore, we confronted problems with tenacity and dedication, viewing setbacks and criticism as opportunities for personal development and growth. Furthermore, our commitment to open communication was critical in aligning our goals, handling conflicts, and keeping a positive attitude throughout the whole process. The use of MSSQL for project management demonstrated the importance of technology in streamlining operations, unifying data, and increasing team cohesiveness.

|  |  |
| --- | --- |
| **Member Name:** | **What you learned:** |
| Abel Macedo | Overall, this project was a great way for our team to test our ability to collaborate and aim for success. It was difficult to begin with, but as we progressed through each milestone, we gained more knowledge and developed. Seeing that was eye-opening. For example, I recall initially struggling with a few of the queries and getting them wrong, but I learned from that and made the necessary corrections to ensure the queries functioned. Seeing every member of our team ask questions during our Zoom call and have someone able to respond was incredibly satisfying. |
| Benayah Hamonangan | Throughout this project, I’ve seen how important time management and effective utilization of time are. When combined with the invaluable asset of teamwork, these factors have proven essential. Balancing multiple commitments in my life has presented obstacles, but I’ve persevered, effectively working around them. There were times when, after completing other responsibilities, I quickly returned to my team at the library, ready to resume our work. Despite the occasional frustration of manually inputting data, this procedure gave me a renewed feeling of focus, ensuring that each entry was carefully exact and complete. |
| Joshua Cottrill | I realized teamwork is the most important thing, and getting everyone together to help is crucial. Throughout this project, I learned that creating a database is complex and time-consuming, and that if problems develop right away, it can be tough to address them while working on other tasks. Also, I learned how to work through challenges and maintain perseverance throughout the project; there were times when I was overwhelmed with the amount of work, but I settled down and asked questions to understand. I discovered that using MSSQL may be frustrating at times, especially with the refresh button and some of the queries, but I was satisfied I could persevere despite feeling burdened. |
| Rodrigo Tumani Soubhia Giesteira | I learned the invaluable importance of feedback—both positive and critical criticism—during this entire effort. At first, it was difficult to put in a lot of work and then hear criticism pointing out shortcomings. But I quickly saw how important this input was to the success of our team. Every criticism functioned as an opportunity for development. Motivated by constructive criticism, I reviewed my mistakes to ensure I didn't make the same ones again. For instance, there were times when the cardinalities in our Entity-Relationship Diagram (ERD) were not appropriately displayed. Still, supplied with feedback I quickly fixed these mistakes to make sure the cardinalities were accurately represented. I’m proud of my ability to accept criticism well and keep going against difficulties. |

# **Appendix**

## **Team Contract**



## **Data Dictionary Model**

The data dictionary is a comprehensive reference table for the entire database. It helps to explain how fields are utilized and maintains the database's credibility. The data dictionary includes the following: Table, Field Name, Key, Data Type, Size, Null, References, and Sample.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Table** | **Field Name** | **Key** | **Data Type** | **Size** | **Null** | **References** | **Sample** |
| TCustomer | CustID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 1006-1483 |
|  | CusTTID | FK | int |  | Not Null | TCustomerType(CusTTID) | 1001 |
|  | CFName |  | varchar | 50 | Not Null |  | Dean |
|  | CLName |  | varchar | 50 | Not Null |  | Weaver |
|  | CPhone |  | varchar | 100 | Not Null |  | (730) 773-4327 |
|  | CAddress |  | varchar | 100 | Not Null |  | Ap #678-689 Eu, Road |
|  | CCity |  | varchar | 50 | Not Null |  | Independence |
|  | StateID | FK | int |  |  | TState(StateID) | 1004 |
|  | CZip |  | varchar | 5 | Not Null |  | 77827 |
|  | CEmail |  | varchar | 50 |  |  | [enim@google.couk](mailto:enim@google.couk) |
| TCustomerType | CusTTID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 1000-1002 |
|  | CusTTName |  | varchar | 50 | Not Null |  | The Local |
| Trip | TripID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 819-1131 |
|  | SOID | FK | int |  | Null | SalesOrder(SOID) | 2382 |
|  | EmpTypeID | FK | int |  | Null | Employee(EmpID) | 912 |
|  | DepartureDate |  | date |  | Null |  | 2023-11-12 |
|  | NoOfTravelers |  | int |  | Null |  | 24 |
| TState | StateID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 1000-1050 |
|  | StateName |  | varchar | 50 | Not Null |  | NM |
| Vendor | VenID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 1001-1020 |
|  | EmpID | FK | int |  | Null | Employee(EmpID) | 1009 |
|  | ProdID | FK | int |  | Null | Product(ProdID) | 1910 |
|  | Name |  | varchar | 50 | Null |  | Pellentesque Ultricies Inc. |
|  | Address |  | varchar | 50 | Null |  | 11400 Casman Ct |
|  | City |  | varchar | 50 | Null |  | Missoula |
|  | StateID | FK | int |  | Null | TState(StateID) | 1021 |
|  | Zip |  | varchar | 15 | Null |  | 59715 |
|  | Phone |  | varchar | 15 | Null |  | (406) 544-1346 |
|  | VendReliabilityRank |  | int |  | Null |  | 8 |
| Size | SizeID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 401-420 |
|  | SizeName |  | varchar | 50 | Null |  | 8 |
|  | PayInID | FK | int |  | Not Null | PaymentIn(PayInID) | 742 |
|  | CustID | FK | int |  | Null | TCustomer(CustID) | 1076 |
|  | EmpID | FK | int |  | Null | Employee(EmpID) | 1083 |
|  | SODate |  | date |  | Null |  | 2023-12-17 |
|  | Total |  | varchar | 50 | Null |  | $81.78 |
|  | DiscountID | FK | int |  | Null | Discount(DiscountID) | 205 |
|  | SCID | FK | int |  | Null | SalesChannel(SCID) | 401 |
| SalesChannel | SCID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 400-402 |
|  | SCName |  | varchar | 50 | Null |  | Phone |
| PurchaseOrder | POID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 801-1150 |
|  | EmpID | FK | int |  | Null | Employee(EmpID) | 915 |
|  | PayOutID | FK | int |  | Null | PaymentOut(PayOutID) | 604 |
|  | PODate |  | date |  | Null |  | 2023-05-15 |
|  | Total |  | varchar | 50 | Null |  | $87.15 |
| Product | ProdID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 917-2287 |
|  | PName |  | varchar | 50 | Null |  | fly fishing gear |
|  | SizeID | FK | int |  | Null | Size(SizeID) | 415 |
|  | ColorID | FK | int |  | Null | Color(ColorID) | 317 |
|  | PatternID | FK | int |  | Null | Pattern(PatternID) | 394 |
|  | Total |  | varchar | 50 | Null |  | 11 |
| PayType | PTID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 200-206 |
|  | PTName |  | varchar | 50 | Null |  | Debit Card |
| PaymentOut | PayOutID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 600-899 |
|  | Invoice# |  | varchar | 50 | Null |  | 11450 |
|  | Amount |  | int |  | Null |  | $113.96 |
|  | PODate |  | date |  | Null |  | 2023-11-16 |
|  | PTID | FK | int |  | Null | PayType(PTID) | 203 |
| PaymentIn | PayInID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 500-759 |
|  | EmpID | FK | int |  | Null | Employee(EmpID) | 958 |
|  | PIDate |  | date |  | Null |  | 2024-06-26 |
|  | Amount |  | varchar | 50 | Null |  | $35 |
|  | PTID | FK | int |  | Null | PayType(PTID) | 200 |
| Pattern | PatternID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 350-414 |
|  | PatternName |  | varchar | 50 | Null |  | Haze |
| EmpType | EmpTypeID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 201-208 |
|  | EmpRole |  | varchar | 50 | Null |  | Guide |
| Employee | EmpID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 900-1200 |
|  | EFirstName |  | varchar | 50 | Null |  | Ursa |
|  | ELastName |  | varchar | 50 | Null |  | Spears |
|  | EAddress |  | varchar | 50 | Null |  | 9667 Volutpat Road |
|  | ECity |  | varchar | 50 | Null |  | College |
|  | EZip |  | varchar | 15 | Null |  | 93738 |
|  | EPhone |  | varchar | 50 | Null |  | (356) 845-8713 |
|  | EEmail |  | varchar | 15 | Null |  | [turpis.non@outlook.ca](mailto:turpis.non@outlook.ca) |
|  | Skill |  | varchar | 50 | Null |  | Good with Children |
|  | EmpTypeID | FK | int |  | Null | EmpType(EmpTypeID) | 208 |
| Discount | DiscountID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 200-208 |
|  | DiscountType |  | varchar | 50 | Null |  | Loyalty |
| DeliveryOut | DelID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 901-1200 |
|  | EmpID-Pick | FK | int |  | Null | Employee(EmpID) | 1191 |
|  | EmpID-Pack | FK | int |  | Null | Employee(EmpID) | 955 |
|  | EmpID-Ship | FK | int |  | Null | Employee(EmpID) | 1032 |
|  | CTID | FK | int |  | Null | CarrierType(CTID) | 505 |
|  | ShipDate |  | date |  | Null |  | 2023-12-28 |
| DeliveryIn | DelID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 701-1000 |
|  | EmpID - Deliver | FK | int |  | Null | Employee(EmpID) | 939 |
|  | EmpID - Inspect | FK | int |  | Null | Employee(EmpID) | 1090 |
|  | EmpID - Receiv | FK | int |  | Null | Employee(EmpID) | 1027 |
|  | DDate |  | date |  | Null |  | 2023-11-26 |
| Color | ColorID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 301-324 |
|  | ColorName |  | varchar | 50 | Null |  | Red |
| CarrierType | CTID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 501-511 |
|  | DeliveryType |  | varchar | 50 | Null |  | express |
|  | TrackingNumber |  | int |  | Null |  | 4714 |
| BundleBOM | BundleID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 901-1300 |
|  | ProdID - Kit | FK | int |  | Null | Product(ProdID) | 1151 |
|  | ProdID - Part | FK | int |  | Null | Product(ProdID) | 1157 |
|  | Quantity |  | int |  | Null |  | 78 |
| SalesOrderLine | SOLID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 933-1319 |
|  | SOID | FK | int |  | Null | SalesOrder(SOID) | 2436 |
|  | DellID | FK | int |  | Null | DeliveryOut(DelID) | 1028 |
|  | Quantity |  | varchar | 50 | Null |  | 37 |
|  | SalePrice |  | varchar | 50 | Null |  | $14.46 |
|  | ProdID | FK | int |  | Null |  | 1865 |
| PurchaseOrderLine | POLID | PK | int(Auto-Increment) Primary Key |  | Not Null |  | 962-1540 |
|  | POID | FK | int |  | Null | PurchaseOrder(POID) | 1534 |
|  | ProdID | FK | int |  | Null | Product(ProdID) | 1417 |
|  | Quantity |  | int |  | Null |  | 39 |
|  | PurchaseCost |  | varchar | 50 | Null |  | $221.62 |

## **Project Management**

