Elysian Fly Fishing Company

Miranda Lambert

Miranda
Lambert

Davis Dunkleberger, Allyson Cusato, Grace Dennis,

Grace Fain, John Foster, and Miles McClanahan

MIS 3353 – Database Management

Clarke Daughterty

December 6th, 2020

Contents

Executive Summary

Miranda Lambert offers efficient and reliable database design for business entities. The Miranda Lambert team consists of college students attending the University of Oklahoma. Our team expects to provide our clients with service that exceeds our clients' needs and wants. We expect competition from other database design companies and are confident we will be able to build a strong market position with our team 's unique skillset and age that leaves us a with different perspective that can help our clients reach their business goals. The clients we seek are businesses ranging from corporations to small businesses. Miranda Lambert was founded in 2020 during the COVID-19 pandemic when the Elysian Fly Company reached out requesting our service. Although we are a new company, we have been able to proudly assist Elysian with their business database and reports. In this document, we will report on the journey of our project with Elysian from the initial meeting to the final product. As well as our findings, strengths, and weaknesses. After reading this report, one will gain a clear understanding of our project's journey.

Get to Know the Team: Miranda Lambert

Name	Major	Year in School	Internship Experience	Background	Photo
Davis Dunkleberger	MIS and Sports Business	Junior	None at the moment	I am from Edmond, Oklahoma and maintain a 3.84 GPA	
Grace Dennis	MIS & Supply Chain MGT	Junior	Interned with Valero Energy in 2020 and will return in 2021	From Blanchard, OK. I have studied abroad in Costa Rica & Panama	
John Foster	MIS	Junior	Tyler Technologies 2019 Texas Capital Bank 2020	I'm from Dallas TX.	
Grace Fain	MIS	Junior	None	From Overland Park, Kansas	

Allyson Cusato	**MIS**	**Senior**	**None**	**From Chickasha, OK.**	
Miles McClanahan	**Finance, Accounting**	**Junior**	**None**	**From Dallas, TX**	

Conceptual Design

Conceptual design is like drawing up a rough outline for a database. You are drawing up a sketch of what your database will look like. It lays the foundation for what will come next in making a database. You also have to make certain assumptions about how certain entities will interact and what needs to happen. These assumptions will help ease later processes in database construction. Good conceptual design eases future steps so it deserves a lot of focus. You will notice how asking questions, making assumptions, and drawing an Entity-Relationship Diagram (ERD) are what we did to conceptually design the database.

The Client Meeting

At the outset of our adventure, we set up a client meeting with Tom from Elysian Fly company through our professor. Here was our chance to ask questions about what Elysian is looking for and clarify what we needed to fulfill what they require of us. Below is the information about our interview. The interview went well, and Tom was impressed with our questions. As a result of the interview, our team was able to better understand Tom's requirements and what his business requirements involved. Time with our clients is exponentially valuable and our team had prepared questions in order to deepen our knowledge of Tom's needs. Each encounter with one of our clients is crucial to progressing any of our projects and ensuring the client is satisfied. He trusted our capable hands and wanted an update on October 26th.

- Meeting Time: October 13th, 2020 4:45PM
- Location: Microsoft Teams
- Interviewers: Davis Dunkleberger, Allyson Cusato, Grace Dennis, Grace Fain, John Foster, and Miles McClanahan
- Interviewee: Tom from Elysian Fly Company

Q&A During the Meeting & Information We Learned

This section contains the dialogue from our first client meeting. During this conversation, we learned more specifics of what the client wants, gained insight on multiple factors of the ERD, and solved all of our ambiguities about the project.

- Is there a special relationship between the items a DIY fly tier can buy?
    - DIY produced in house so think as a production line
- Guides are separate from employees, correct?
    - Both considered as employees
- The customer section mentions four types of customers, yet there are only three types listed. Is that a typo, or is there a fourth one to be aware of?
    - Only 3
- Should the fly bundles contribute to each individual item's sales or as its own product?
    - Bundles/DIY kits are their own products
- Do you want to collect the email address of just the purchaser or the guests as well?
    - Both would be ideal
- Do you want to label the discount by its promotion name or by percentage?
    - Use a reference table or associative entity.
- Can vendors also be customers?
    - Classified separately from customers
- One of the expectations is to control discounts, what exactly is expected of this?
    - Are there any certain limits to discounts? There can be multiple discounts, no more than 10% but can have multiple types
- What metrics do you want to use to measure the success of your employees?
    - Predicting the fly, we can have multiple criteria
- For the top 10% margin products, would you like that separated by month or weather during the time of those sales? For example, a certain fly may be able to sell for more when in high demand, increasing the margin.
    - Fine as is. No more separation needed
- What sort of data are you looking for in regard to the decision-making process for expansion or new business opportunities?
    - Have the location of where products are frequently purchased
- Only one subtype/super***
- Consider tables used for each query. Can model lots of tables with those queries.
- Unary M2M somewhere

Significant Assumptions

This section contains the significant assumptions that we made in order to complete the ERD. These assumptions were crucial to the functionality of the database.

One of the assumptions that we made is that a guided tour is related to only one body of water. So that means that each tour only visits one body of water and does not go to multiple lakes or rivers. This is reflected in the relationship between Body Water and Guided Trip. The second thing that we assumed was that each fly was in one and only one bundle. This means that the same fly would not appear in multiple bundle types. For example, the flies that are in the "weekend pack" will not be included in a different bundle. This is shown in the relationship between product and bundle. The third thing that we assumed is that the best way to track orders and the way that the orders were placed would be for each order channel to be tracked separately. This is shown in the sub-type super-type relationship. The fourth assumption that we made is that partial orders can be shipped. This is shown in the relationship between sales order line and delivery out. The last assumption that we made is that someone does not have to place an order in order to be considered a customer. This is shown in the relationship between sales order and customer.

What is an ERD? Why is it necessary?

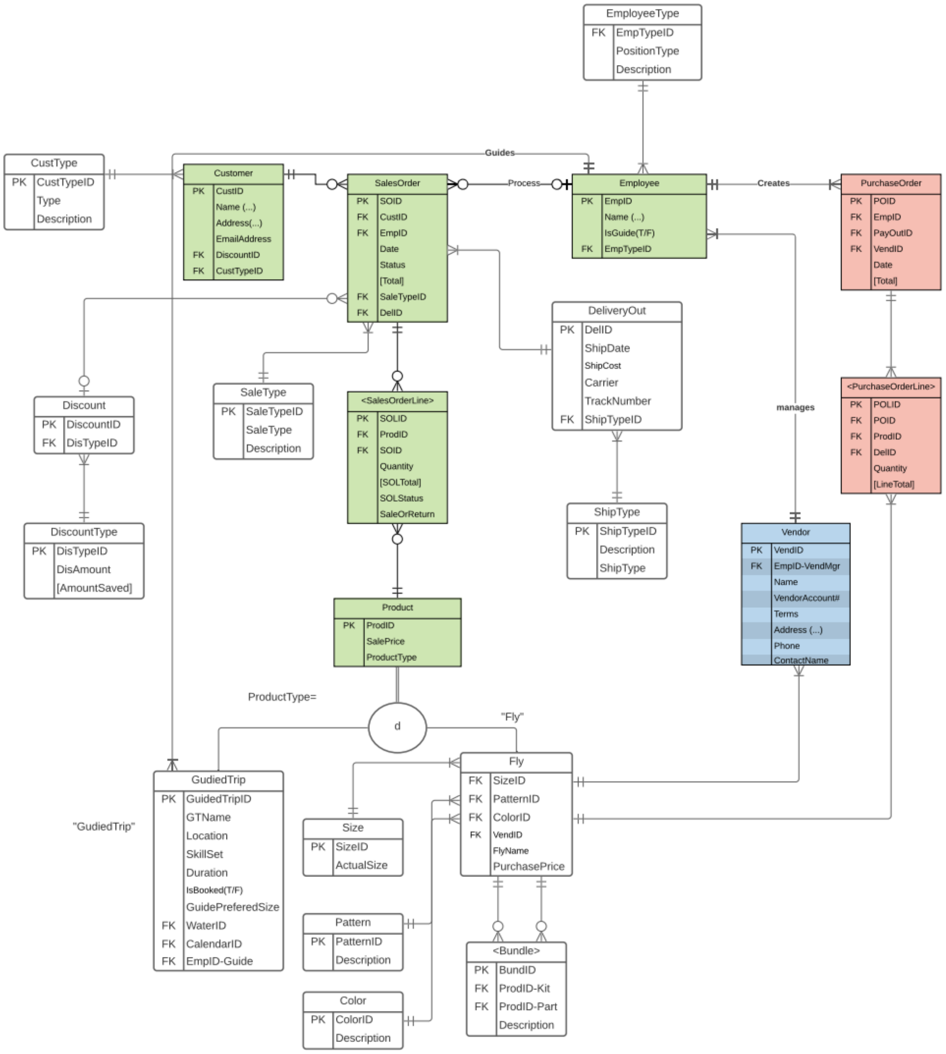
ERD's act as a blueprint for your company in regard to defining the relationships between entities and attributes. The ERD also helps with the overall design within a database. By defining these relationships, an ERD allows a database to store information and output reports that can help a company make decisions. For example, Elysian Fly Company will use an ERD to define the relationship between employees and sale orders. This way the company database can determine what employees are selling what items.

Business Cycles Used

For the relationships between customers and what they purchased, we need to use the revenue cycle. This ERD allows for businesses to create sales order invoices and track what is sold, where it goes, which employees were involved, and any payment methods. The Expenditure cycle is included since there is mention of receiving and purchasing materials. The process of buying products means we need the expenditure cycle. Finally, since Elysian produces their own ties and DIY kits, we need the production cycle.

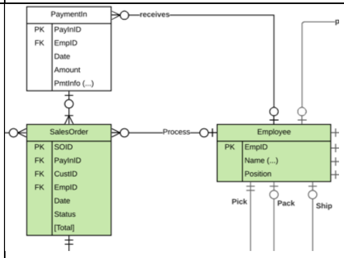
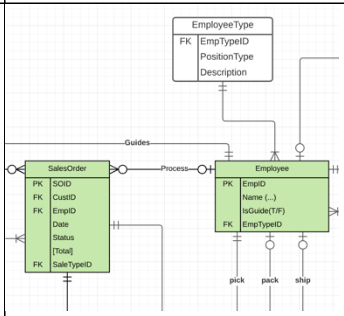
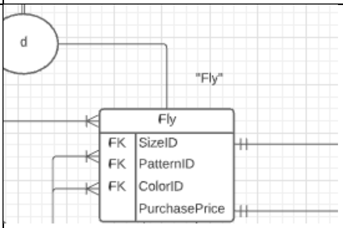
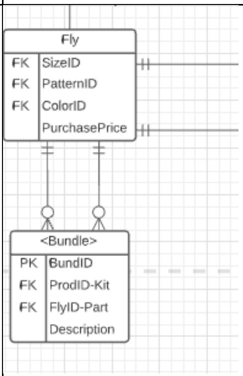
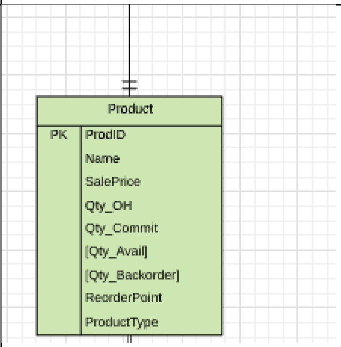
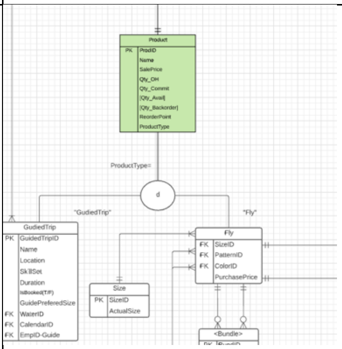
ERD Created

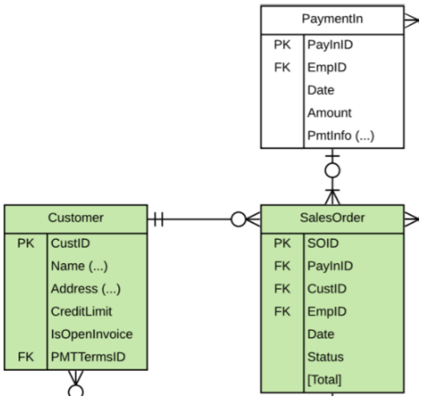
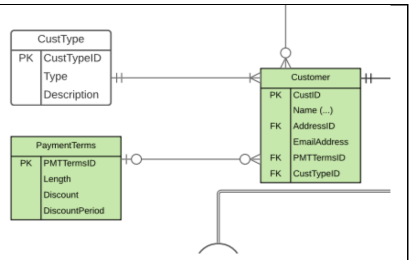
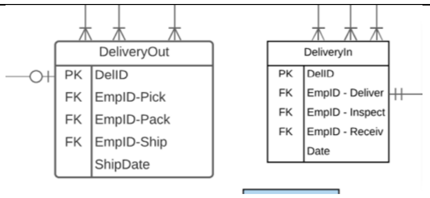
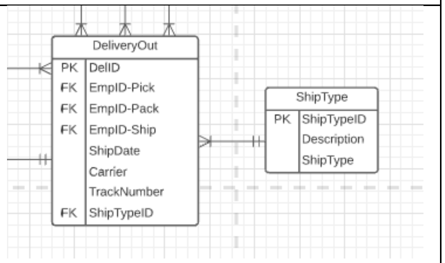
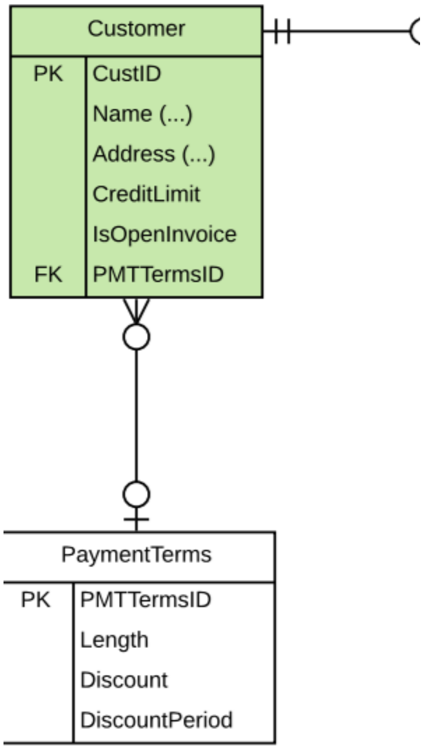
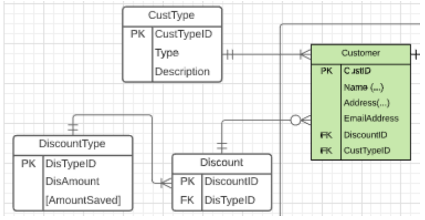
Below is a screenshot of the ERD created for Elysian Fly Company. As shown below the ERD includes the revenue, expenditure and production cycles. Added entities and attributes have been added to the generic ERD in order to store information and run reports as specified by the company.


EmployeeType
FK EmpTypeID
PositionType
Description
Guides
CustType
PK CustTypeID
Type
Description
Customer
PK CustID
Name (...)
Address(...)
EmailAddress
FK DiscountID
FK CustTypeID
SalesOrder
PK SOID
FK CustID
FK EmpID
Date
Status
[Total]
FK SaleTypeID
FK DelID
Process
Employee
PK EmpID
Name (...)
IsGuide(T/F)
FK EmpTypeID
Creates
PurchaseOrder
PK POID
FK EmpID
FK PayOutID
FK VendID
Date
[Total]
DeliveryOut
PK DelID
ShipDate
ShipCost
Carrier
TrackNumber
FK ShipTypeID
manages
<PurchaseOrderLine>
PK POLID
FK POID
FK ProdID
FK DelID
Quantity
[LineTotal]
Discount
PK DiscountID
FK DisTypeID
SaleType
PK SaleTypeID
SaleType
Description
<SalesOrderLine>
PK SOLID
FK ProdID
FK SOID
Quantity
[SOLTotal]
SOLStatus
SaleOrReturn
ShipType
PK ShipTypeID
Description
ShipType
Vendor
PK VendID
FK EmpID-VendMgr
Name
VendorAccount#
Terms
Address (...)
Phone
ContactName
DiscountType
PK DisTypeID
DisAmount
[AmountSaved]
Product
PK ProdID
SalePrice
ProductType
ProductType=
d
"Fly"
Fly
FK SizeID
FK PatternID
FK ColorID
FK VendID
FlyName
PurchasePrice
GudiedTrip
PK GuidedTripID
GTName
Location
SkillSet
Duration
IsBooked(T/F)
GuidePreferedSize
FK WaterID
FK CalendarID
FK EmpID-Guide
"GudiedTrip"
Size
PK SizeID
ActualSize
Pattern
PK PatternID
Description
Color
PK ColorID
Description
<Bundle>
PK BundID
FK ProdID-Kit
FK ProdID-Part
Description

Changes made to generic ERDs

Listed here are the changes made to our ERD as the project has evolved. Each change contains details such as the chronological number, original generic ERD, and the updated ERD. This section can be used to track different changes throughout the course of the project.

<table>
<tr><th>Change #</th><th>Original ERD</th><th>Updated ERD</th></tr>
<tr>
<td>1) We removed the payment in entity from the Revenue cycle. We did not notice anything about payment on account for customers. The table would sit empty and cause unnecessary clutter with the ERD and database.</td>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>2) We added an associative entity to Fly and Bundle in the revenue cycle. This entity is important because it allows us to track which fly parts are in a bundle, as well as which flies can be sold in a bundle. This is especially helpful for</td>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>3) The two types of products were used to create a subtype/supertype of product types. Since Elysian offers two products with distinct attributes to each, a subtype/supertype can accurately convey those differences.</td>
<td>
</td>
<td>
</td>
</tr>
</table>

• 4) With the new types of customers, adding a reference table was necessary. It allows Elysian to track how many customers they have of each type and the habits and attributes in the Customer table.	 PaymentIn PK PayInID FK EmpID Date Amount PmtInfo (...) Customer PK CustID Name (...) Address (...) CreditLimit IsOpenInvoice FK PMTTermsID SalesOrder PK SOID FK PayInID FK CustID FK EmpID Date Status [Total]	 CustType PK CustTypeID Type Description PaymentTerms PK PMTTermsID Length Discount DiscountPeriod Customer PK CustID Name (...) FK AddressID EmailAddress FK PMTTermsID FK CustTypeID
• 5) To keep track of shipping types, adding a reference table for the types prevents more data duplication. The reference table allows more specifics about the shipping type to be stored right there.	 DeliveryOut PK DelID FK EmpID-Pick FK EmpID-Pack FK EmpID-Ship ShipDate DeliveryIn PK DelID FK EmpID - Deliver FK EmpID - Inspect FK EmpID - Receiv Date	 DeliveryOut PK DelID FK EmpID-Pick FK EmpID-Pack FK EmpID-Ship ShipDate Carrier TrackNumber FK ShipTypeID ShipType PK ShipTypeID Description ShipType
• 6) In a generic cycle, there are no discounts available. Elysian would like to offer them to customers and track them. The best way to do this is by adding a discount table with specific attributes to track the discounts. Since they are applied on a customer basis, they should be attached to the customer table.	 Customer PK CustID Name (...) Address (...) CreditLimit IsOpenInvoice FK PMTTermsID PaymentTerms PK PMTTermsID Length Discount DiscountPeriod	 CustType PK CustTypeID Type Description Customer PK CustID Name (...) Address(...) EmailAddress FK DiscountID FK CustTypeID DiscountType PK DisTypeID DisAmount [AmountSaved] Discount PK DiscountID FK DisTypeID

Logical Design

Logical Design is the process of conceptualizing how entities and their attributes are arranged within a database. The result of logical design is a set of well-structured tables that detail each entity's data and foreign keys in a well-organized method. This will enable the process of normalization to be done more efficiently.

Normalization

Normalization is a process that is used to help improve a database by ensuring that it is reliable and efficient. A normalized database consists of two things which are atomicity and no data duplication. Atomicity means that values cannot be broken down any further than what they already are. Atomicity is important because it makes sure that queries can be run easily. The second thing that is not wanted within a database is data redundancy. This is a bad thing because if duplicate data is taking up space in the database when it does not need to be it can cause issues with the efficiency of the database. For example, it can cause queries to run slower than normal and cause user frustration. This is important for our project because we want our database to be efficient and user friendly.

Normalized Relations

TCustomer(CustID, CName, CAddress, CEmailAddress, CDiscountID, CCustTypeID)

Foreign Key CDiscountID references TDiscount

Null Allowed

On Delete Set Null

Foreign Key CCustTypeID references TCustType

Not Null

On Delete Restrict

TCustType(CustTypeID, CType, CDescription)

TDiscount(DiscountID, DDisTypeID)

Foreign Key DDisTypeID references TDiscountType

Not Null

On Delete Restrict

TDiscountType(DisTypeID, DDisAmount, DAmountSaved)

TProduct(ProductID, PSalePrice, PQty_OH, PQty_Commit, PReorderPoint, PProductType)

TGuidedTrip(GuidedTripProductID, GTName, GTLocation, GTSkillSet, GTDuration, GTIsBooked, GTGuidePreferedSize, GTEmpID-Guide)

TFly(FlyProductID, FSizeID, FPatternID, FColorID, FPurchasePrice)

Foreign Key FSizeID references TSize

Not Null

On Delete Restrict

Foreign Key FPatternID references TPattern

Not Null

On Delete Restrict

Foreign Key FColorID references TColor

Not Null

On Delete Restrict

TSize(SizeID, ZActualSize)

TPattern(PatternID, PDescription)

TColorID(ColorID, CDescription)

TVendor(VendID,VEmpID, VName, VTerms, VStreet, VZipCode, VPhone, VContactFirstName, VContactLastName)

Foreign Key VZipCode references TZipCode

Not Null

On Delete Restrict

TZipCode(ZipCode, ZCity, ZStateCode)

Foreign Key ZStateCode references TState

Not Null

On Delete Restrict

TState(StateCode, StateName)

TSaleOrder(SOID, SOCustID, SOEmpID, SOSaleTypeID, SODate, SOStatus, SOTotal, _SODelID_)

Foreign Key SOCustID references TCustomer

Not Null

On Delete Restrict

Foreign Key SOEmpID references TEmployee

Null Allowed

On Delete Set Null

Foreign Key SOSaleType references TSaleType

Not null

On Delete Restrict

Foreign Key SOLDellID references TDeliveryOut

Not Null

On Delete Restrict

TSaleOrderLine(SOLID, SOLProdID, SOLSOID, SOLGuideTripID, SOLQuantity, SOLTotal, SOlStatus, SOLSaleorReturn)

Foreign Key SOLProdID references TProduct

Not Null

On Delete Restrict

Foreign Key SOLSOID references TSaleOrder

Not Null

On Delete Restrict

Foreign Key SOLGuideTripID references TGuideTrip

Not Null

On Delete Restrict

TSaleType(SaleTypeID, STSaleType, STDescription)

TEmployee(EmpID, EmpType, EFirstName, ELastName, EisGuideT, EisGuideF)

Foreign Key EmpType references TEmployeeType

Not Null

On Delete Restrict

TPurchaseOrder(POID, EmpID, PayOutID, VendID, PODate, POTotal)

Foreign Key EmpID references TEmployee

Not Null

On Delete Restrict

Foreign Key PayOutID references TPaymentOut

Not Null

On Delete Restrict

Foreign Key VendID references TVendor

Not Null

On Delete Restrict

TPurchaseOrderLine(POLID, POID, ProdID, DelID, Quantity, LineTotal)

Foreign Key POID references TPurchaseOrder

Not Null

On Delete Restrict

Foreign Key ProdID references TProduct

Not Null

On Delete Restrict

Foreign Key DelID references TDeliveryOut

Not Null

On Delete Restrict

TShipType(ShipTypeID, Description, ShipType)

TBundle(BundID, ProdID-Kit, FlyID-Part, Description)

TFly(SizeID, PatternID, ColorID, PurchasePrice)

Foreign Key ProdID-Kit references Vendor

Not Null

On Delete Restrict

Foreign Key FlyID-Part references Vendor

Not Null

On Delete Restrict

TDeliveryOut(DellID, EmpID-Pick, EmpID-Pack, EmpID-Ship, ShipDate, Carrier, TrackNumber, ShipTypeID)

TEmployee(EmpID, EmpFirstName, EmpLastName, IsGuide, EmpTypeID)

Foreign Key EmpID-Pick

Not Null

On Delete Restrict

Foreign Key EmpID-Pack references

Not Null

On Delete Restrict

Foreign Key EmpID-Ship references Employee

Not Null

On Delete Restrict

Foreign Key ShipTypeID references ShipType

Not Null

On Delete Restrict

Foreign Key EmpTypeID references SalesOrder

Not Null

On Delete Restrict

TEmployeeType(EmpTypeID, PositionType, Description)

TEmployee(EmpID, EmpFirstName, EmpLastName, IsGuide, EmpTypeID)

Foreign Key EmpType references SalesOrder

Not Null

On Delete Restrict

Foreign Key EmpTypeID references SalesOrder

Not Null

On Delete Restrict

Differences between ERD and Normalized Relations

An essential part of the design process is ERDs. To provide a macro view of your company's data requirements and operations, ERDs are very useful for accomplishing just that. An ER Diagram displays major groups of information and certain attributes. ERDs help with the process of compiling large levels of information that will eventually be implemented into the database. Overall, ERDs highlight the relationship amongst attributes and entities. With ERDs, the data that is being addressed is only in the conceptual stage. The conceptual aspect enhances the database design and implementation. Working with ERDs enables our team to analyze your company's business needs in a manner that enhances data quality.

Normalized relations can give insight into data redundancy and help ensure accuracy when updating or removing data. The higher normal forms have less redundancy and assist with ensuring improved database performance. For example, we have created a Customer entity, otherwise, each time there was a new Customer there would need to be a new line item created. By creating the Customer entity, we have intended to save hard drive space for the company. Another beneficial aspect of normalized relations is that normalizing can help ensure the database is trustworthy and efficient. Trustworthy in the sense that your company can generate reliable reports and gather information swiftly.

The three major differences between ERDs and Normalized Relations are that normalization requires atomicity, no data redundancy, and has no anomalies. Anomalies to avoid include deletion, insertion, and modification.

Referential Integrity

When doing normalization by hand, you can draw an arrow between the foreign key in the new relation to the primary key in the original relation. Computers do not like these arrows at all; they do not understand them one bit. In order to maintain that relationship of primary key donating to become a foreign key, we need what is called a Referential Integrity Constraint, or RI constraint for short. RI constraints tell the computer what foreign key is referencing which relation, if that key can be null or not, and the restriction for the database system. RI constraints ensure that foreign keys contain only valid values, such as values that exist in a primary key in the table that the foreign key is referencing. RI constraints help the computer know what our ERD looks like and keep the relationship integrity of that diagram.

The other integrity constraints are Entity Integrity and Domain Integrity. The Entity Integrity Constraint requires every entity to have a primary key that is not null, does not change over time, and exists for all records. The Domain Integrity Constraint requires that all values from a column must be from a set of values or value types specific to that column. Composite attributes must be split into atomic attributes.

Physical Design and Implementation

Physical design and implementation are the processes of implementing the database design into a relational database management system or RDBMS. It is in this step of database development that we are create a database that works and effectively outputs the information being asked for. It is important that a physical design ensures efficient performance and database integrity, security, and recoverability.

Data Dictionary

A Data Dictionary is a set of information describing the physical design process and acts as a guide for how the database is to be implemented. The data dictionary lists the contents, formats and structure of the database. It is important because it sets the framework when making decisions about the physical design of the database and allows those working with the database to understand the limits and intent behind each field within the database. Below are examples of the data dictionary for tables used in our database. Our data dictionary includes the field name, primary key or foreign key, data type, constraints, and table referenced.

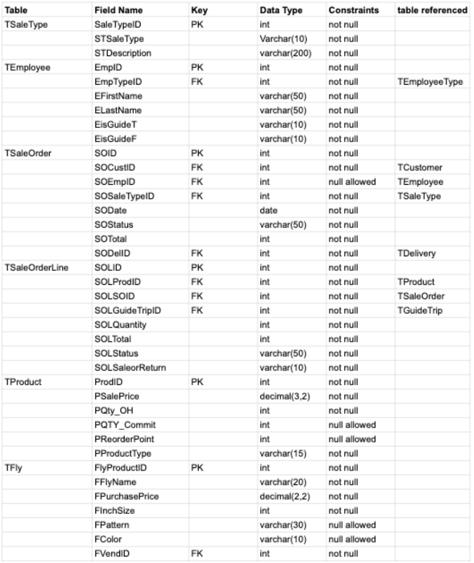
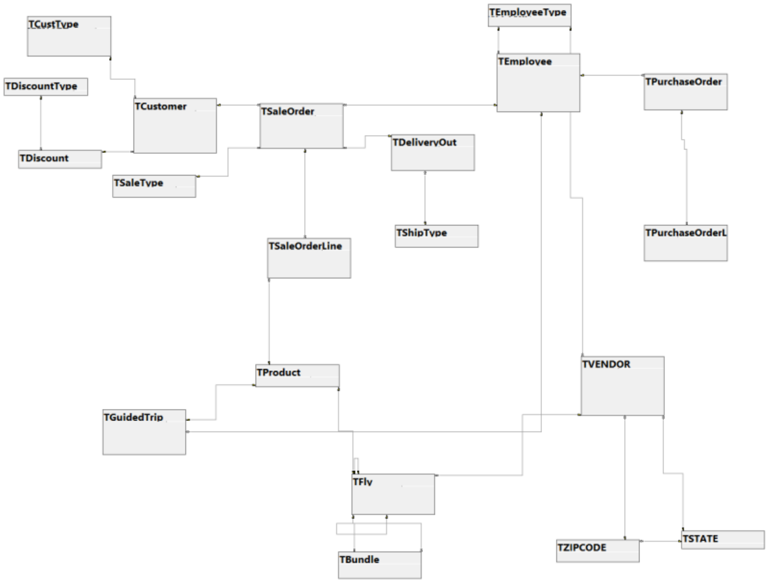


Table	Field Name	Key	Data Type	Constraints	table referenced
TSaleType	SaleTypeID	PK	int	not null	
	STSaleType		Varchar(10)	not null	
	STDescription		varchar(200)	not null	
TEmployee	EmpID	PK	int	not null	
	EmpTypeID	FK	int	not null	TEmployeeType
	EFirstName		varchar(50)	not null	
	ELastName		varchar(50)	not null	
	EisGuideT		varchar(10)	not null	
	EisGuideF		varchar(10)	not null	
TSaleOrder	SOID	PK	int	not null	
	SOCustID	FK	int	not null	TCustomer
	SOEmpID	FK	int	null allowed	TEmployee
	SOSaleTypeID	FK	int	not null	TSaleType
	SODate		date	not null	
	SOStatus		varchar(50)	not null	
	SOTotal		int	not null	
	SODelID	FK	int	not null	TDelivery
TSaleOrderLine	SOLID	PK	int	not null	
	SOLProdID	FK	int	not null	TProduct
	SOLSOID	FK	int	not null	TSaleOrder
	SOLGuideTripID	FK	int	not null	TGuideTrip
	SOLQuantity		int	not null	
	SOLTotal		int	not null	
	SOLStatus		varchar(50)	not null	
	SOLSaleorReturn		varchar(10)	not null	
TProduct	ProdID	PK	int	not null	
	PSalePrice		decimal(3,2)	not null	
	PQty_OH		int	not null	
	PQTY_Commit		int	null allowed	
	PReorderPoint		int	null allowed	
	PProductType		varchar(15)	not null	
TFly	FlyProductID	PK	int	not null	
	FFlyName		varchar(20)	not null	
	FPurchasePrice		decimal(2,2)	not null	
	FInchSize		int	not null	
	FPattern		varchar(30)	null allowed	
	FColor		varchar(10)	null allowed	
	FVendID	FK	int	not null	

Denormalization

Regarding denormalization, our team decided to not use this strategy. For example, with TZip and TState we did not denormalize. The thinking behind this strategy was that we wanted to reduce data redundancy and maintain data integrity. We wanted to enhance the database by maintaining the normalized state, which also helps when the database is updated. Often, database designers choose to denormalize to speed up data retrieval. Our team believed that if we normalized TZip and TState that this would ensure the data is accurate, as opposed to denormalizing just to speed up data retrieval.

Implemented Physical Design



Challenges Faced/Addressed During Implementation

When implementing the database, we found some challenges we did not expect to encounter. Coming up with the queries for Elysian Fly Company was harder than we thought. While we should know the intricacies of a database we created, some of the vaguer goals set forth can be hard to translate into SQL. While putting our heads together, we found that each of us should attack the queries related to our area of the database. Splitting up work earlier in the project helped us reach our goal; however, we specialized in areas. Certain queries will focus on certain areas. Helping focus the queries on those areas made writing them a lot easier.

Another challenge was inserting the sample data into our tables. While still a new concept, inserting data tripped a lot of us up. The exactness of the “insert into” format tripped us up. None of us are real pros at adding data and it showed. There were lots of error messages in the statements. Referencing text files from class helped us piece our way through. The explanations there made these statements a lot easier to digest and understand. All the data there should now be in there free from harm.

A specific challenge we faced was when creating the queries that required the user to input data. Queries 11 and 13 specifically required this. We had to use a parameter to do so and had difficulty creating a parameter that worked and reported what was being asked.

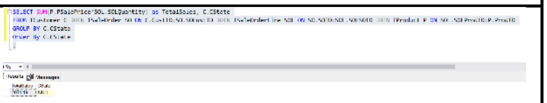
Strengths and Weaknesses Encountered During Implementation

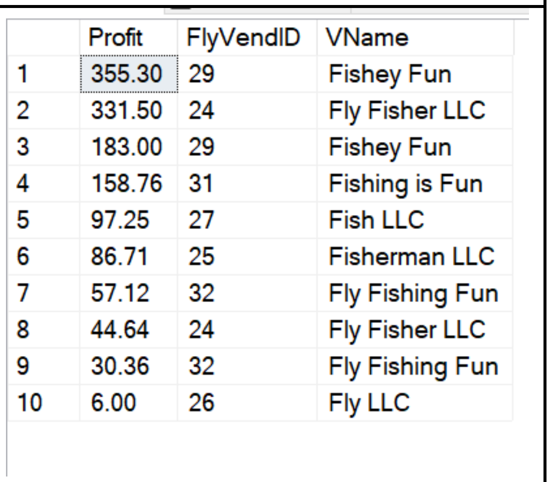
Our team discovered both strengths and weaknesses with building ERDs and writing queries when implementing data. One of our team's strengths was creating tables. Our team effectively and efficiently created the tables necessary for the database. Our team's weaknesses lie in inputting values or data into the tables and creating the necessary query to run specific reports. Inputting values in the correct syntax and creating queries that reported properly were aspects the team struggled with when implementing this database. It became evident that a weakness for the entire team was the Data Query Language and that our lack of knowledge and experience with the relational database management system was causing challenges to occur.

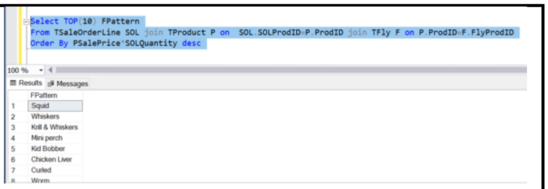
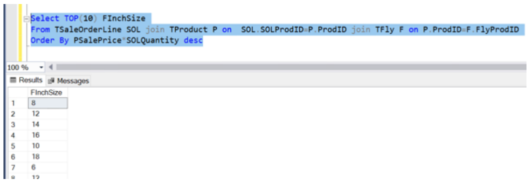
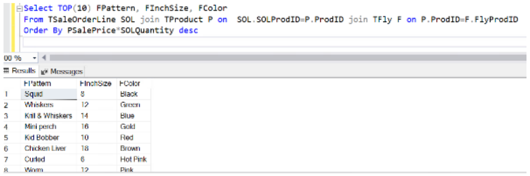
Specific SQL Statements Requested

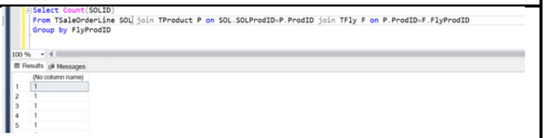
Elysian Fly Company asked us to make sure the database could execute certain functions to test their business' health. These queries can help them learn important trends about products and customers and managers can turn that information into action. These queries will be procedures in the database and can be run at any time and allow for reuse. That way, Elysian Fly can use these at any point if they decide to monitor their business.

Query #	Question	SQL	Partial Output

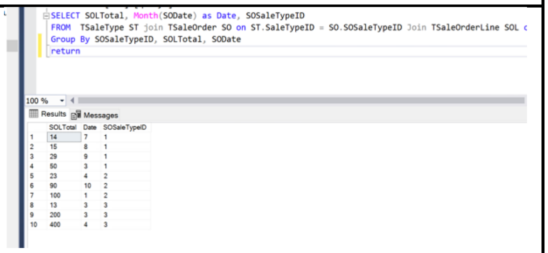
1	Total sales (in dollars) by customer state per year (e.g., total sales for all customers from Montana, Wyoming, Colorado, etc.).	SELECT SUM(P.PSalePrice*SOL.SOLQuantity) as TotalSales, C.CAddress
		FROM TCustomer C JOIN TSaleOrder SO ON C.CustID=SO.SOCustID JOIN TSaleOrderLine SOL ON SO.SOID=SOL.SOLSOID JOIN TProduct P ON SOL.SOLProdID=P.ProdID
		WHERE year(SO.SODate) =
			
			SELECT SUM(P.PSalePrice*SOL.SOLQuantity) as TotalSales, C.CState
			FROM TCustomer C JOIN TSaleOrder SO ON C.CustID=SO.SOCustID JOIN TSaleOrderLine SOL ON SO.SOID=SOL.SOLSOID JOIN TProduct P ON SOL.SOLProdID=P.ProdID
			GROUP BY C.CState
			Order By C.CState
			TotalSales CState
			[illegible] [illegible]

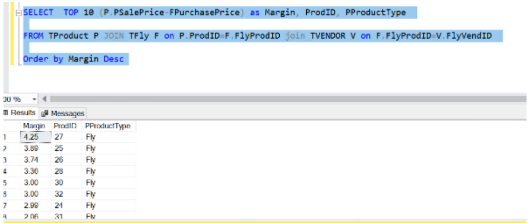
<table>
<tr>
<td></td>
<td></td>
<td>2020 AND year(SO.SODate) = 2021
GROUP BY C.CAddress
Order By C.CAddress
;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Total sales (in dollars) by vendor per year. We must be able to calculate profit (sale price – purchase price).</td>
<td>SELECT ((P.PSalePrice-F.FPurchasePrice)*SOL.SOLQuantity) AS Profit, V.FlyVendID, V.VName
FROM TProduct P JOIN TSaleOrderLine SOL ON P.ProdID=SOL.SOLProdID JOIN TSaleOrder SO ON SOL.SOLSOID=SO.SOID JOIN TFly F ON P.ProdID=F.FlyProdID JOIN TVENDOR V ON F.FFlyVendID=V.FlyVendID
GROUP BY V.FlyVendID, (P.PSalePrice-F.FPurchasePrice)*SOL.SOLQuantity, V.VName
ORDER BY Profit DESC, V.FlyVendID
;</td>
<td>
<table>
<tr><th></th><th>Profit</th><th>FlyVendID</th><th>VName</th></tr>
<tr><td>1</td><td>355.30</td><td>29</td><td>Fishey Fun</td></tr>
<tr><td>2</td><td>331.50</td><td>24</td><td>Fly Fisher LLC</td></tr>
<tr><td>3</td><td>183.00</td><td>29</td><td>Fishey Fun</td></tr>
<tr><td>4</td><td>158.76</td><td>31</td><td>Fishing is Fun</td></tr>
<tr><td>5</td><td>97.25</td><td>27</td><td>Fish LLC</td></tr>
<tr><td>6</td><td>86.71</td><td>25</td><td>Fisherman LLC</td></tr>
<tr><td>7</td><td>57.12</td><td>32</td><td>Fly Fishing Fun</td></tr>
<tr><td>8</td><td>44.64</td><td>24</td><td>Fly Fisher LLC</td></tr>
<tr><td>9</td><td>30.36</td><td>32</td><td>Fly Fishing Fun</td></tr>
<tr><td>10</td><td>6.00</td><td>26</td><td>Fly LLC</td></tr>
</table>
</td>
</tr>
</table>

3	The ten highest selling (in dollars) (a) patterns, (b) sizes, (c) pattern-size-color combinations in a given year.	(a) SELECT TOP (10) FPattern FROM TSaleOrderLine SOL join TProduct P on SOL.SOLProdID=P.ProdID join TFly F on P.ProdID=F.FlyProdID ORDER BY PSalePrice * SOLQuantity desc (b) SELECT TOP (10) FInchSize FROM TSaleOrderLine SOL join TProduct P on SOL.SOLProdID=P.ProdID join TFly F on P.ProdID=F.FlyProdID ORDER BY PSalePrice* SOLQuantity desc (c) SELECT TOP (10) FPattern, FInchSize, FColor FROM TSaleOrderLine SOL join TProduct P on SOL.SOLProdID=P.ProdID join TFly F on P.ProdID=F.FlyProdID ORDER BY PSalePrice* SOLQuantity desc	 Select TOP(10) FPattern From TSaleOrderLine SOL join TProduct P on SOL.SOLProdID=P.ProdID join TFly F on P.ProdID=F.FlyProdID Order By PSalePrice*SOLQuantity desc 100 % Results Messages FPattern 1 Squid 2 Whiskers 3 Krill & Whiskers 4 Mini perch 5 Kid Bobber 6 Chicken Liver 7 Curled 8 Worm  Select TOP(10) FInchSize From TSaleOrderLine SOL join TProduct P on SOL.SOLProdID=P.ProdID join TFly F on P.ProdID=F.FlyProdID Order By PSalePrice*SOLQuantity desc 100 % Results Messages FInchSize 1 8 2 12 3 14 4 16 5 10 6 18 7 6 8 12  Select TOP(10) FPattern, FInchSize, FColor From TSaleOrderLine SOL join TProduct P on SOL.SOLProdID=P.ProdID join TFly F on P.ProdID=F.FlyProdID Order By PSalePrice*SOLQuantity desc 00 % Results Messages FPattern FInchSize FColor 1 Squid 8 Black 2 Whiskers 12 Green 3 Krill & Whiskers 14 Blue 4 Mini perch 16 Gold 5 Kid Bobber 10 Red 6 Chicken Liver 18 Brown 7 Curled 6 Hot Pink 8 Worm 12 Pink

4	The number of times each product (fly) was sold. We want to see also those flies that have never been	SELECT COUNT (SOLID)
		FROM TSaleOrderLine SOL join TProduct P on SOL.SOLProdID=P.ProdID join TFly F on P.ProdID=F.FlyProdID
			
			Select Count(SOLID)
			From TSaleOrderLine SOL join TProduct P on SOL.SOLProdID=P.ProdID join TFly F on P.ProdID=F.FlyProdID
			Group by FlyProdID
			100 %
			Results Messages
			(No column name)
			1 1
			2 1
			3 1
			4 1
			5 1

	sold so that we can discontinue them	GROUP BY FlyProdID	

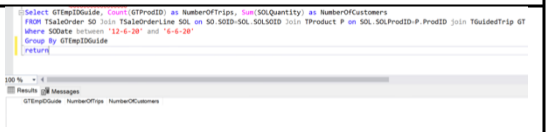
5	Total sales (in dollars) for each channel per month.	CREATE PROC query5 AS
		SELECT SOLTotal, Month(SODate) as Date, SOSaleTypeID
		FROM TSaleType ST join TSaleOrder SO on ST.SaleTypeID = SO.SOSaleTypeID Join TSaleOrderLine SOL on SO.SOID=SOL.SOLSOID join TProduct P on SOL.SOLProdID= P.ProdID
		Group By SOSaleTypeID, SOLTotal, SODate
			
			SELECT SOLTotal, Month(SODate) as Date, SOSaleTypeID
			FROM TSaleType ST join TSaleOrder SO on ST.SaleTypeID = SO.SOSaleTypeID Join TSaleOrderLine SOL c
			Group By SOSaleTypeID, SOLTotal, SODate
			return
			100 %
			Results Messages
			SOLTotal Date SOSaleTypeID
			1 14 7 1
			2 15 8 1
			3 29 9 1
			4 50 3 1
			5 23 4 2
			6 90 10 2
			7 100 1 2
			8 13 3 3
			9 200 3 3
			10 400 4 3

6	The 10% of products that have the highest margin.	CREATE PROC query6 AS
		SELECT TOP 10 (P.PSalePrice-FPurchasePrice) as Margin, ProdID, PProductType
			
			SELECT TOP 10 (P.PSalePrice-FPurchasePrice) as Margin, ProdID, PProductType
			FROM TProduct P JOIN TFly F on P.ProdID=F.FlyProdID join TVENDOR V on F.FlyProdID=V.FlyVendID
			Order by Margin Desc
			00 %
			Results Messages
			Margin ProdID PProductType
			1 4.25 27 Fly
			2 3.89 25 Fly
			3 3.74 26 Fly
			4 3.36 28 Fly
			5 3.00 30 Fly
			6 3.00 32 Fly
			7 2.99 24 Fly
			8 2.06 31 Fly

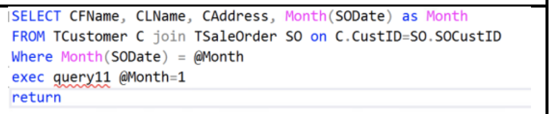
		FROM TProduct P JOIN TFly F on P.ProdID=F.FlyProdID join TVENDOR V on F.FlyProdID=V.FlyVendID Order by Margin Desc	

7	The ten most popular (units sold) DIY fly-tying materials.	CREATE PROC
		query7
		SELECT FLY, COUNT(SOLID) AS ‘count’, ProductName
		FROM TSalesOrder SO
		JOIN TSalesORderLine SL on TSalesOrder.SOID=TSalesOrderLine.SOLID
		WHERE ProducType=’bundle’	

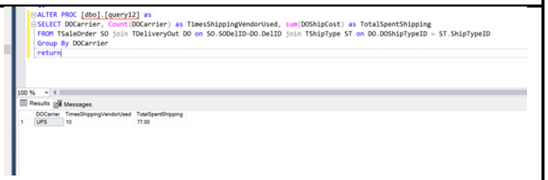
8	The number of distinct products managed by each vendor manager.	CREATE PROC
		query8
		SELECT ManagerID, ManagerName, COUNT(ProductID)	

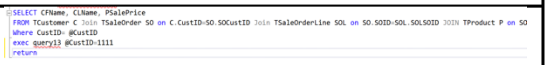
		FROM Employee E JOIN Vendor V On E.EmplyeeID=V.VendorID Join Fly F On V.VendorID=F.FLyID GROUP BY ManagerID, ManagerName	
9	The upcoming, scheduled guided trips (i.e., the guided trips that have already been sold) for each guide, including the guide's name, the trip destination, the customer name, and the number in the customer's party.	CREATE PROC Query9 SELECT EmpID-Guide, C.FirstName, Guide-PreferredSize, Location FROM GuidedTrip GT JOIN Employee E ON E.EmpID=GT.GuidedTrip JOIN SalesOrder SO On SO.SOID = E.EmpID GROUP BY	
10	Number of trips and number of customers taken on fishing trips by	CREATE PROC query10 as Select GTEmpIDGuide,	 Select GTEmpIDGuide, Count(GTProdID) as NumberOfTrips, Sum(SOLQuantity) as NumberOfCustomers FROM TSaleOrder SO Join TSaleOrderLine SOL on SO.SOID=SOL.SOLSOID Join TProduct P on SOL.SOLProdID=P.ProdID join TGuidedTrip GT Where SODate between '12-6-20' and '6-6-20' Group By GTEmpIDGuide return 100 % Results Messages GTEmpIDGuide NumberOfTrips NumberOfCustomers

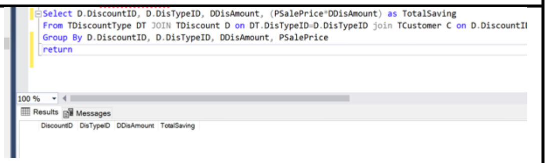
	each guide in the past 6 months.	Count(GTProdID) as NumberOfTrips, Sum(SOLQuantity) as NumberOfCustomers FROM TSaleOrder SO Join TSaleOrderLine SOL on SO.SOID=SOL.SOLSOID Join TProduct P on SOL.SOLProdID=P.ProdID join TGuidedTrip GT on P.ProdID=GT.GTProdID Where SODate between '12-6-20' and '6-6-20' Group By GTEmpIDGuide	

11	Names and email addresses of all customers who made purchases in a given month. We need to be able to enter the month.	CREATE PROC query11 (@month int) as
		SELECT CFName, CLName, CAddress, Month(SODate) as Month
		FROM TCustomer C join TSaleOrder SO on C.CustID=SO.SOCustID
			
			SELECT CFName, CLName, CAddress, Month(SODate) as Month
			FROM TCustomer C join TSaleOrder SO on C.CustID=SO.SOCustID
			Where Month(SODate) = @Month
			exec query11 @Month=1
			return

		Where Month(SODate) = @Month	

12	Number of times used and dollars spent on each shipping vendor and shipping type by vendor.	CREATE PROC query12 as
		SELECT DOCarrier, Count(DOCarrier) as TimesShippingVendorUsed, sum(DOShipCost) as TotalSpentShipping
		FROM TSaleOrder SO join TDeliverOut DO on SO.SODelID=DO.DelID join TShipType ST on DO.DOShipTypeID = ST.ShipTypeID
		Group By DOCarrier
			
			ALTER PROC [dbo].[query12] as
			SELECT DOCarrier, Count(DOCarrier) as TimesShippingVendorUsed, sum(DOShipCost) as TotalSpentShipping
			FROM TSaleOrder SO join TDeliveryOut DO on SO.SODelID=DO.DelID join TShipType ST on DO.DOShipTypeID = ST.ShipTypeID
			Group By DOCarrier
			return
			DOCarrier TimesShippingVendorUsed TotalSpentShipping
			1 UPS 10 77.00

13	Invoice lines for a given sales invoice number and given customer name.	CREATE PROC query13 (@CustID int) as
		SELECT CFName, CLName, PSalePrice
		FROM TCustomer C Join TSaleOrder SO on C.CustID=SO.SOCustID Join
			
			SELECT CFName, CLName, PSalePrice
			FROM TCustomer C Join TSaleOrder SO on C.CustID=SO.SOCustID Join TSaleOrderLine SOL on SO.SOID=SOL.SOLSOID JOIN TProduct P on SO
			Where CustID= @CustID
			exec query13 @CustID=1111
			return

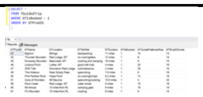
		TSaleOrderLine SOL on SO.SOID=SOL.SOLSOID JOIN TProduct P on SOL.SOLProdID=P.ProdID Where CustID= @CustID	
14	Number of times a discount was applied to a sales order. List all information about the discount, total amount saved by customers that used the discount.	CREATE PROD query14 as Select D.DiscountID, D.DisTypeID, DDisAmount, (PSalePrice*DDisAmount) as TotalSaving From TDiscountType DT JOIN TDiscount D on DT.DisTypeID=D.DisTypeID join TCustomer C on D.DiscountID = C.DiscountID join TSaleOrder SO on C.CustID=SO.SOCustID Join TSaleOrderLine SOL on SO.SOID=SOL.SOLSOID join TProduct P on SOL.SOLProdID=P.ProdID	 Select D.DiscountID, D.DisTypeID, DDisAmount, (PSalePrice*DDisAmount) as TotalSaving From TDiscountType DT JOIN TDiscount D on DT.DisTypeID=D.DisTypeID join TCustomer C on D.DiscountI Group By D.DiscountID, D.DisTypeID, DDisAmount, PSalePrice return DiscountID DisTypeID DDisAmount TotalSaving

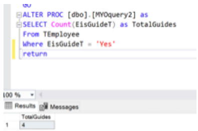
		Group By D.DiscountID, D.DisTypeID, DDisAmount, PSalePrice	

Three Additional Queries

Here we have decided to add some of our own queries for the database. We believe these queries can provide finer research on sectors of your business and analysis of your different areas within the business. These can help find new trends in data, locate the most booked trips, and so much more. We think you will enjoy these.

Query #	Question	Why is this important	SQL	Partial Output	Recap of Findings

1	Which Trips have been booked the most	Knowing which trips are being booked the most can help Elysian know what trips deserve more attention. You can notice which ones are often booked with the boolean value for this reason.	SELECT *
			FROM TGuidedTrip
			WHERE IsBooked = 1
			ORDER BY GTProdID
			;
				 SELECT *
				FROM TGuidedTrip
				WHERE GTIsBooked = 1
				ORDER BY GTProdID
				;
					We have found that over half of the trips that Elysian offers are booked often. The trip section of the business is strong but other trips can help flesh out that sector of your market.

2	How many Employees are also Guides	Knowing how many employees are guides allows the company to be aware of how many trips can be out at once and what they need to look for when hiring	CREATE PROC MYOquery2 as
			SELECT Count(EisGuideT) as TotalGuides
			From TEmployee
			Where EisGuideT = 'Yes'
				 GO
				ALTER PROC [dbo].[MYOquery2] as
				SELECT Count(EisGuideT) as TotalGuides
				From TEmployee
				Where EisGuideT = 'Yes'
				return
				100 %
				Results Messages
				TotalGuides
				1 4
					This query finds that of the employees in the database 4 of them are guides.

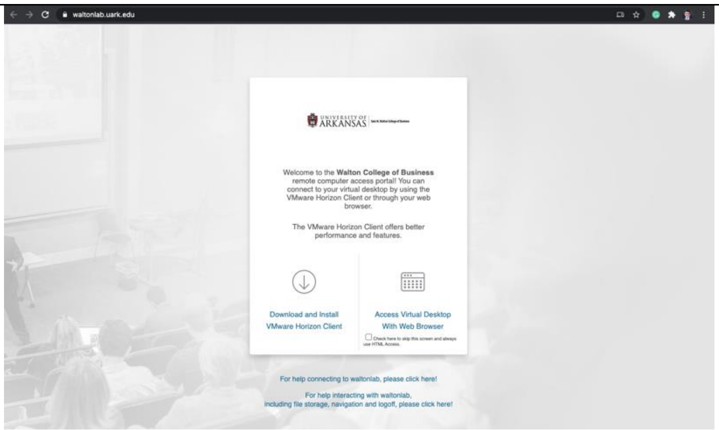
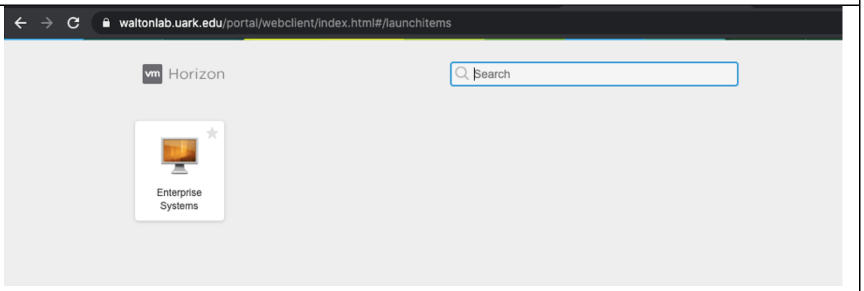
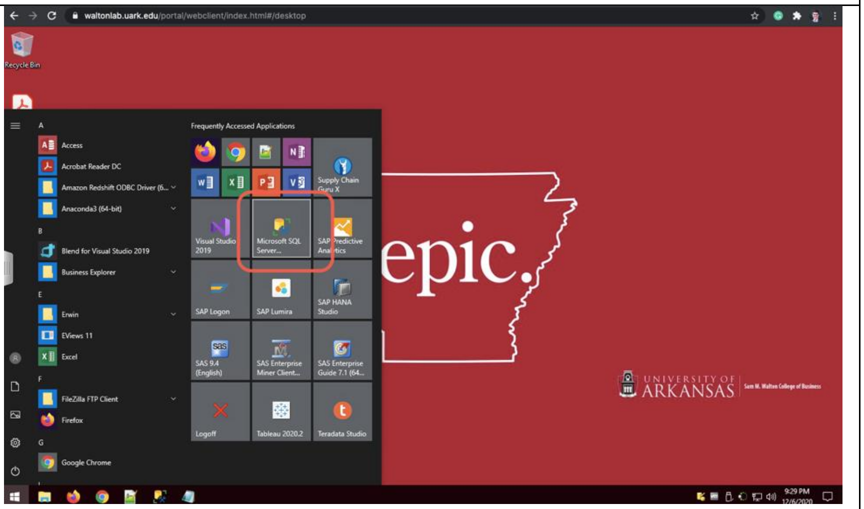
3	Which products have the	Understanding the products with the lowest margin provides	SELECT (P.SalePrice-F.PurchasePrice)		The third query outputs the products with the

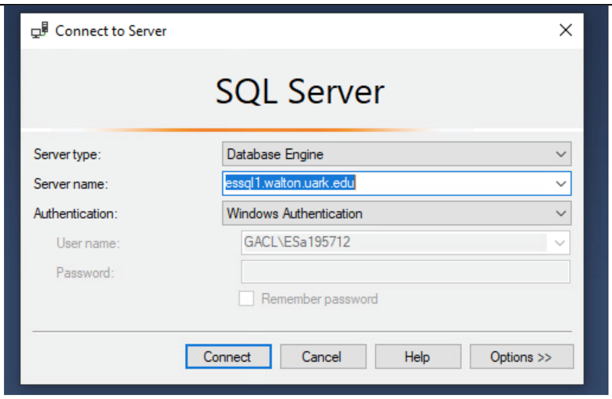
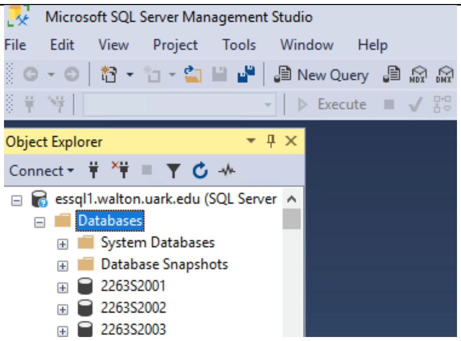
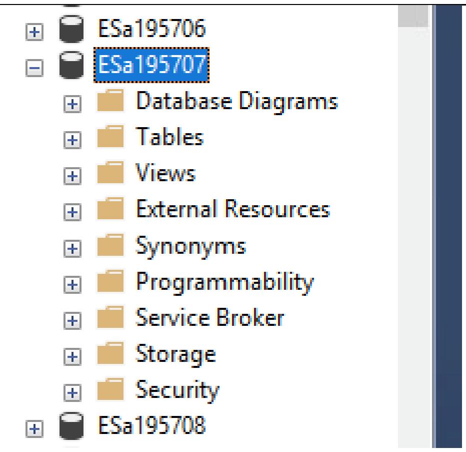
	lowest margin	insight into products that may need to be eliminated or improved	as Margin, FlyName FROM TProduct Order by Margin DESC		lowest margins.

User Documentation

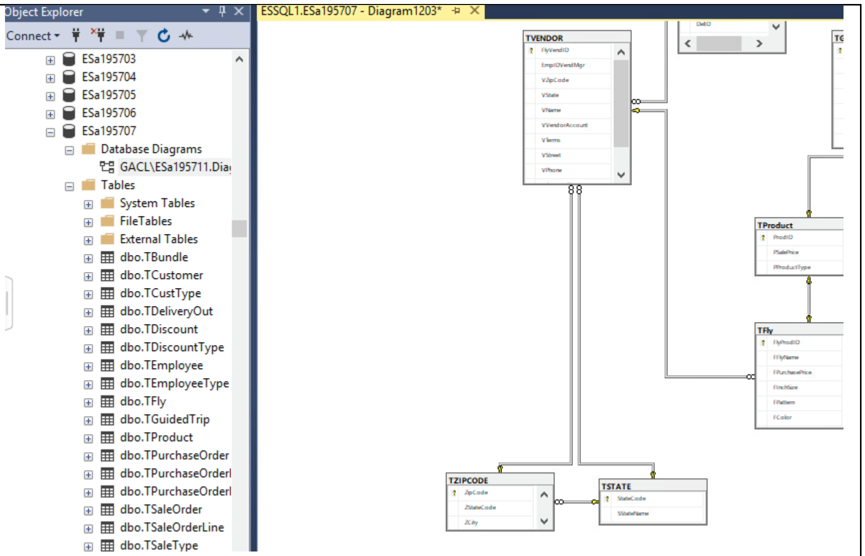
Regarding access to the database, we will cover the process step-by-step in detail. This section will be useful for learning how to access the database and will be beneficial for anyone else at the company that will need to enter the database. The process to access the database does not change, meaning this user documentation is useful to repeatedly refer to. In this section there will be visuals for each step. There will be screenshots to show you where to click and what to look for as you navigate towards the database. After reading the detailed instructions on how to access this database, Elysian's personnel will feel confident regarding how to utilize the database.

To access database:

1) Access Walton Remote by searching: "https://waltonlab.uark.edu/"	 waltonlab.uark.edu UNIVERSITY OF ARKANSAS Welcome to the **Walton College of Business** remote computer access portal! You can connect to your virtual desktop by using the VMware Horizon Client or through your web browser. The VMware Horizon Client offers better performance and features. Download and Install VMware Horizon Client Access Virtual Desktop With Web Browser □ Check here to skip this screen and always use HTML Access. For help connecting to waltonlab, please click here! For help interacting with waltonlab, including file storage, navigation and logoff, please click here!
2) Click "Access Virtual Desktop With Web Browser"	
3) Select "Enterprise Systems"	 waltonlab.uark.edu/portal/webclient/index.html#/launchitems vm Horizon Search Enterprise Systems
4) From the Walton desktop interface, Click the Windows Start in the bottom-left corner	 waltonlab.uark.edu/portal/webclient/index.html#/desktop Recycle Bin A Access Acrobat Reader DC Amazon Redshift ODBC Driver (6... Anaconda3 (64-bit) B Blend for Visual Studio 2019 Business Explorer E Erwin EViews 11 Excel F FileZilla FTP Client Firefox G Google Chrome Frequently Accessed Applications Supply Chain Guru X Visual Studio 2019 Microsoft SQL Server... SAP Predictive Analytics SAP Logon SAP Lumira SAP HANA Studio SAS 9.4 (English) SAS Enterprise Miner Client... SAS Enterprise Guide 7.1 (64... Logoff Tableau 2020.2 Teradata Studio epic. UNIVERSITY OF ARKANSAS 9:29 PM 12/6/2020
5) Select "Microsoft SQL Server" in the applications	

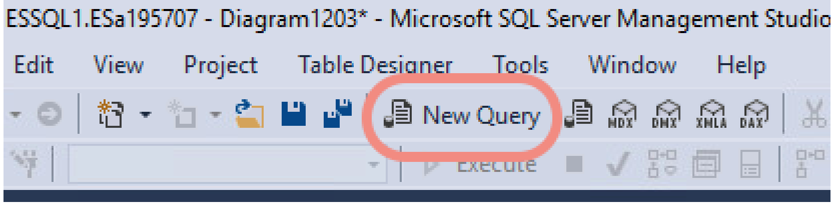
6) Enter Server Name "essql1.walton.uark.edu" into the Server Name box and Click Connect	 Connect to Server SQL Server Server type: Database Engine Server name: essql1.walton.uark.edu Authentication: Windows Authentication User name: GACL\ESa195712 Password: □ Remember password
7) Double-Click to expand on "Databases"	 Microsoft SQL Server Management Studio File Edit View Project Tools Window Help New Query Execute Object Explorer Connect essql1.walton.uark.edu (SQL Server Databases System Databases Database Snapshots 2263S2001 2263S2002 2263S2003
8) Scroll down to find the database titled "ESA195707", double-click to expand	 ESa195706 ESa195707 Database Diagrams Tables Views External Resources Synonyms Programmability Service Broker Storage Security ESa195708

8a) To view the ERD, double-click “Database Diagrams” and double-click the diagram listed

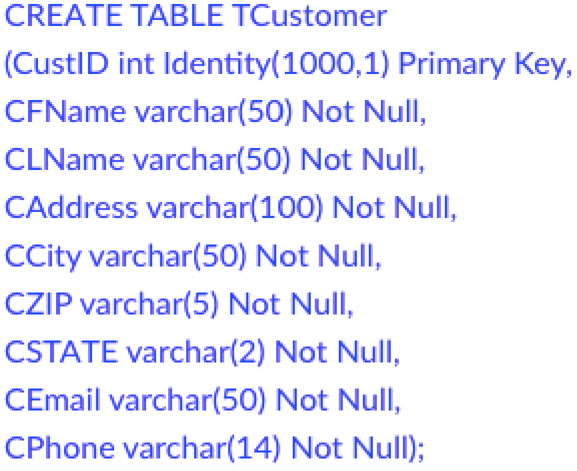


To enter data into tables:

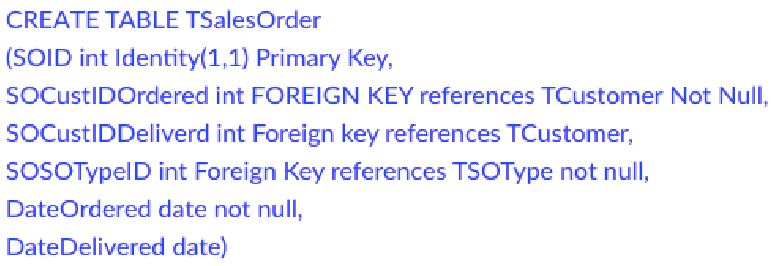
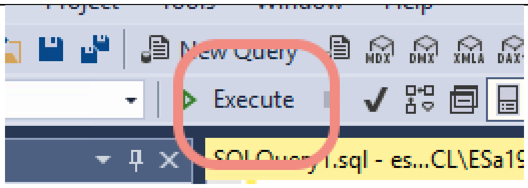
1) Create the table to input data into by selecting “New Query”

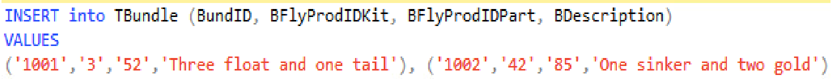


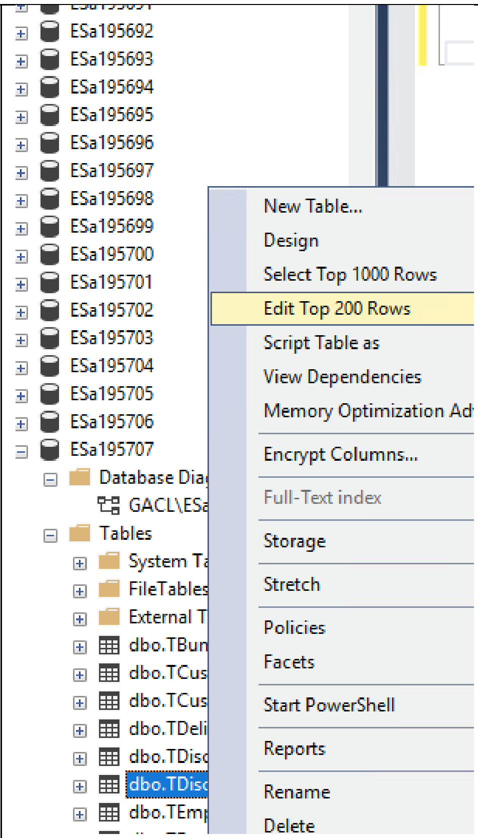
2) Create the table using SQL as follows. Each Table creation begins with “CREATE TABLE”. After this comes the letter T, indicating ‘Table’, followed by the Table Name, in this case, “Customer”. Next, open parenthesis and begin with the Primary Key Name, then the data type and denotation of it as the PK. The following entries begin with the Column Name, and also include data type and constraints. Follow each line with a comma to indicate the



```
CREATE TABLE TCustomer
(CustID int Identity(1000,1) Primary Key,
CFName varchar(50) Not Null,
CLName varchar(50) Not Null,
CAddress varchar(100) Not Null,
CCity varchar(50) Not Null,
CZIP varchar(5) Not Null,
CSTATE varchar(2) Not Null,
CEmail varchar(50) Not Null,
CPhone varchar(14) Not Null);
```

end of the line and return to the next line. If there is a Foreign Key present, it should be written as such in the 2nd picture – Column Name, data type, note what the FK references, and constraints. At the end of all columns, close the query with); to signify the end of the query.	 CREATE TABLE TSalesOrder (SOID int Identity(1,1) Primary Key, SOCustIDOrdered int FOREIGN KEY references TCustomer Not Null, SOCustIDDeliverd int Foreign key references TCustomer, SOSOTypeID int Foreign Key references TSOType not null, DateOrdered date not null, DateDelivered date)
3) Highlight all of the query you are wanting to run and hit Execute.	 New Query MDX DMX XMLA DAX Execute SQLQuery1.sql - es...CL\ESa19

4) Now that the table is created, we can input data into it. To do so, we will open a New Query (step 1) and type as follows in the picture. Begin with "INSERT into" to signify the entry of data. After, name the table you are inserting into in a similar style as creating the table. Open parenthesis after and indicate the column names you are inserting data into. Close the parenthesis and return to the next line and type "Values" to indicate values are about to fill those columns. Next, open parenthesis and input the data to fill the columns, separating them like this: ('###','###'). Once you've inserted all data desired, repeat Step 3.	
	INSERT into TBundle (BundID, BFlyProdIDKit, BFlyProdIDPart, BDescription)
	VALUES
	('1001','3','52','Three float and one tail'), ('1002','42','85','One sinker and two gold')

4b) To access/view the data inserted, right-click the table desired and select “Edit Top 200 Rows”	

What We Learned Throughout This Process

As a team we learned how to work together in order to efficiently generate a database for our client. This is a great accomplishment for our team considering that while we were furthering Elysian Fly Company's operational efficiency, we were also students. With COO Sam Ferreira's guidance and clarifications at the beginning of the project, we gained a deeper understanding of what the company was requesting, which helped us to move forward. Throughout the process, our team maintained respect and we often communicated to help manage the project. Respect and communication were our goal from the beginning and we overall kept this team-environment as we worked on the project. We were able to communicate tasks and accomplishments through the project management tool. This tool allowed us to track what was getting done and by who. Respect and strong communication throughout this project enabled us to maneuver difficult situations that we faced. Particularly when our team was creating queries and implementing the database. Below is what each team member has learnt as a result of the project:

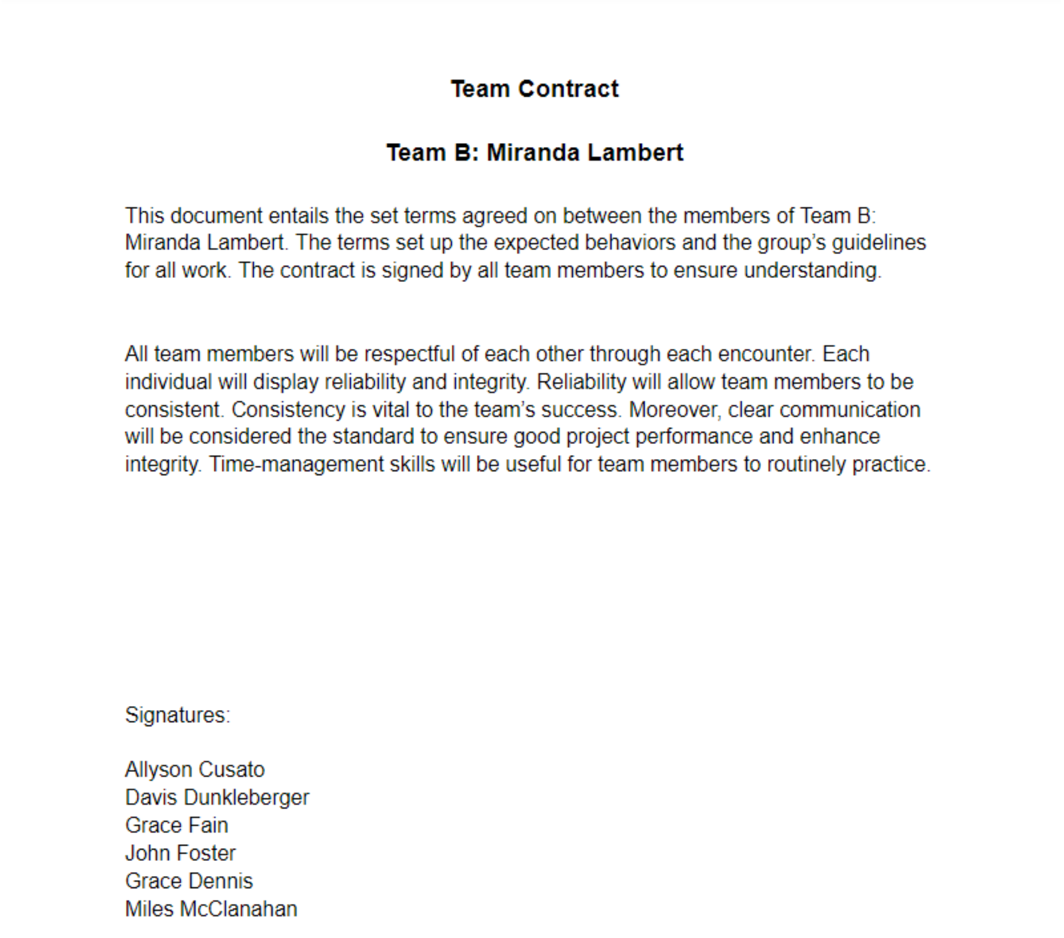
Member Name:	**What you learned:**
Grace Dennis	Learning how to use Data Query Language was one of the biggest challenges. This aspect really challenged me and after experiencing the project, I feel like I have grown in my capabilities regarding the design and implementation of a database. Another challenging factor was learning how to maneuver the tables within SSMS. After finishing the project, I am still new to maneuvering, however, I feel like I am standing on solid ground compared to the beginning of the Elysian Fly Company project.
Grace Fain	At the beginning of this project, I had little knowledge on how to create a database or the steps required to do so. Throughout working on the Elysian Fly Company project, I have learned how to create complex ERDs, how to normalize relations, and create and implement a physical design of a database. I specifically learned the importance of a data dictionary when implementing a database and the clarity that a data dictionary can bring. I also learned how challenging working within a SQL server can be and the importance of being able to troubleshoot while implementing the database.
John Foster	I have worked in SQL databases in the past, but I had never learned how to create one from scratch. This project taught me project management skills and how to create a database in SSMS after thorough planning and revisioning through diagrams. The biggest challenges I faced in this project was connecting all my tables with foreign keys and creating the queries.

Miles McClanahan	Working with databases and SQL was an entirely new concept to me, so learning the concepts needed to apply it to the project has been difficult. However, I have learned how to craft SQL statements to search a database, the process for creating tables in a database and how to view your database inside of SQL. While they are relatively small achievements, I feel as though I could reasonably walk through the process from start to finish, or at least be able to understand the thought process behind it.
Davis Dunkleberger	The most interesting thing I learned was how to normalize relations from an ERD. With earlier classes in MIS, I was familiar enough with SQL to run queries and implementation made sense. I had no idea about the relations and what is needed to make sure the database does not collapse on itself. Normalization was simple enough to learn but the tendencies to look out for were so cool for me to learn and practice in this project.
Allyson Cusato	When we first started this project I had very little knowledge of how to create a database. I would say the most important thing that I learned is about the amount of work that goes into creating a database and how labor intensive it can be. I also found that it can be quite challenging at times to get the database to work but once it is complete and it is fully functioning then it is very rewarding.

Appendix

Team Contract & Logo

The contents within this section include the team contract. The team contract entails behavioral expectations and the standard from which the group members have intended to establish. The behavioral aspect is a foundation for group members to act by. The standards are intended to enhance execution of the project's tasks in order to successfully complete the project.



Team Contract

Team B: Miranda Lambert

This document entails the set terms agreed on between the members of Team B: Miranda Lambert. The terms set up the expected behaviors and the group's guidelines for all work. The contract is signed by all team members to ensure understanding.

All team members will be respectful of each other through each encounter. Each individual will display reliability and integrity. Reliability will allow team members to be consistent. Consistency is vital to the team's success. Moreover, clear communication will be considered the standard to ensure good project performance and enhance integrity. Time-management skills will be useful for team members to routinely practice.

Signatures:

Allyson Cusato
Davis Dunkleberger
Grace Fain
John Foster
Grace Dennis
Miles McClanahan

Miranda Lambert

Data Dictionary Model

This is our team's complete data dictionary. This was used to complete the physical design and acted as a guide while implementing the database. Our data dictionary lists the field name, key, data type, constraints and tables referenced.

Table	Field Name	Key	Data Type	Constraints	table referenced
TSaleType	SaleTypeID	PK	int	not null	
	STSaleType		Varchar(10)	not null	
	STDescription		varchar(200)	not null	
TEmployee	EmpID	PK	int	not null	
	EmpTypeID	FK	int	not null	TEmployeeType
	EFirstName		varchar(50)	not null	
	ELastName		varchar(50)	not null	
	EisGuideT		varchar(10)	not null	
	EisGuideF		varchar(10)	not null	
TSaleOrder	SOID	PK	int	not null	
	SOCustID	FK	int	not null	TCustomer
	SOEmpID	FK	int	null allowed	TEmployee
	SOSaleTypeID	FK	int	not null	TSaleType

	SODate		date	not null	
	SOStatus		varchar(50)	not null	
	SOTotal		int	not null	
	SODelID	FK	int	not null	TDelivery
TSaleOrderLine	SOLID	PK	int	not null	
	SOLProdID	FK	int	not null	TProduct
	SOLSOID	FK	int	not null	TSaleOrder
	SOLGuideTripID	FK	int	not null	TGuideTrip
	SOLQuantity		int	not null	
	SOLTotal		int	not null	
	SOLStatus		varchar(50)	not null	
	SOLSaleorReturn		varchar(10)	not null	
TProduct	ProdID	PK	int	not null	
	PSalePrice		decimal(3,2)	not null	
	PQty_OH		int	not null	
	PQTY_Commit		int	null allowed	
	PReorderPoint		int	null allowed	
	PProductType		varchar(15)	not null	
TFly	FlyProductID	PK	int	not null	
	FFlyName		varchar(20)	not null	
	FPurchasePrice		decimal(2,2)	not null	
	FInchSize		int	not null	
	FPattern		varchar(30)	null allowed	

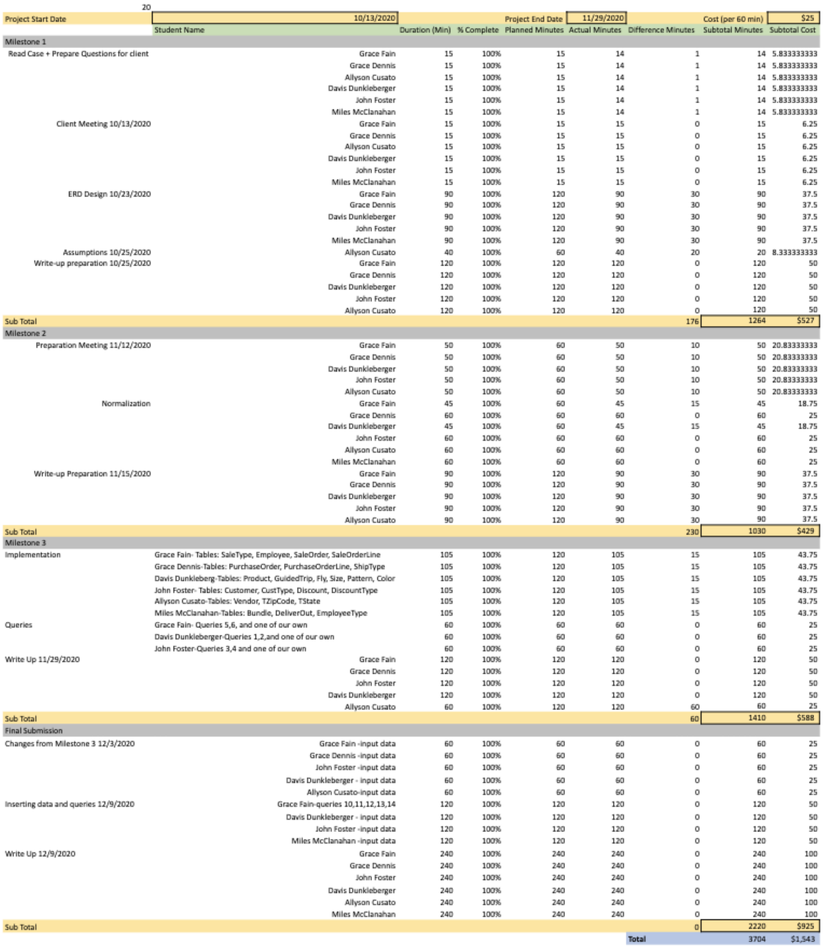
	FColor		varchar(10)	null allowed	
	FVendID	FK	int	not null	
TGuidedTrip	GuidedTripProductID	PK	int	not null	
	GTName		varchar(20)	not null	
	GTLocation		varchar(20)	not null	
	GTSkillSet		varchar(20)	not null	
	GTDuration		varchar(10)	not null	
	GTIsBooked		tinyint(0,1)	null allowed	
	GTGuidePreferredSize		tinyint(1,15)	not null	
	GTEmpID-Guide	FK	int	not null	TEmployee
TPurchaseOrder	POID	PK	int	not null	
	PODate		date	not null	
	POTotal		int	not null	
	EmpID	FK	int	not null	TEmployee
	PayOutID	FK	int	not null	TPaymentOut
	VendID	FK	int	not null	TVendor
TShipType	ShipTypeID	PK	int	not null	
	Description		varchar(200)	not null	
	ShipType		varchar(50)	not null	
TPurchaseOrderLine	POLID	PK	int	not null	
	POID	FK	int	not null	TPurchaseOrder

	ProdID	FK	int	not null	TProduct
	DelID	FK	int	not null	TDeliveryOut
	Quantity		int	not null	
	Line Total		int	not null	
TBundle	BundID	PK	int	not null	
	BProdID-Kit	FK	int	not null	TProduct
	BProdID-Part	FK	int	not null	TProduct
	BDescription		varchar(50)	not null	
TEmployeeType	EmpTypeID	PK	int	not null	
	ETPositionType		varchar(15)	not null	
	ETDescription		varchar(50)	not null	
TDeliveryOut	DelID	PK	int	not null	
	DOShipDate		date	not null	
	DOShipCost		decimal(3,2)	not null	
	DOCarrier		varchar(15)	not null	
	DOTrackingNumber		varchar(50)	not null	
	DOShipTypeID	FK	int	not null	TShipType
TVendor	VendID	PK	int	not null	
	EmpIDVendMgr	FK	int	not null	TEmployee
	VName		varchar(50)	not null	
	VVendorAccount		varchar(20)	not null	
	VTerms		varchar(50)	not null	

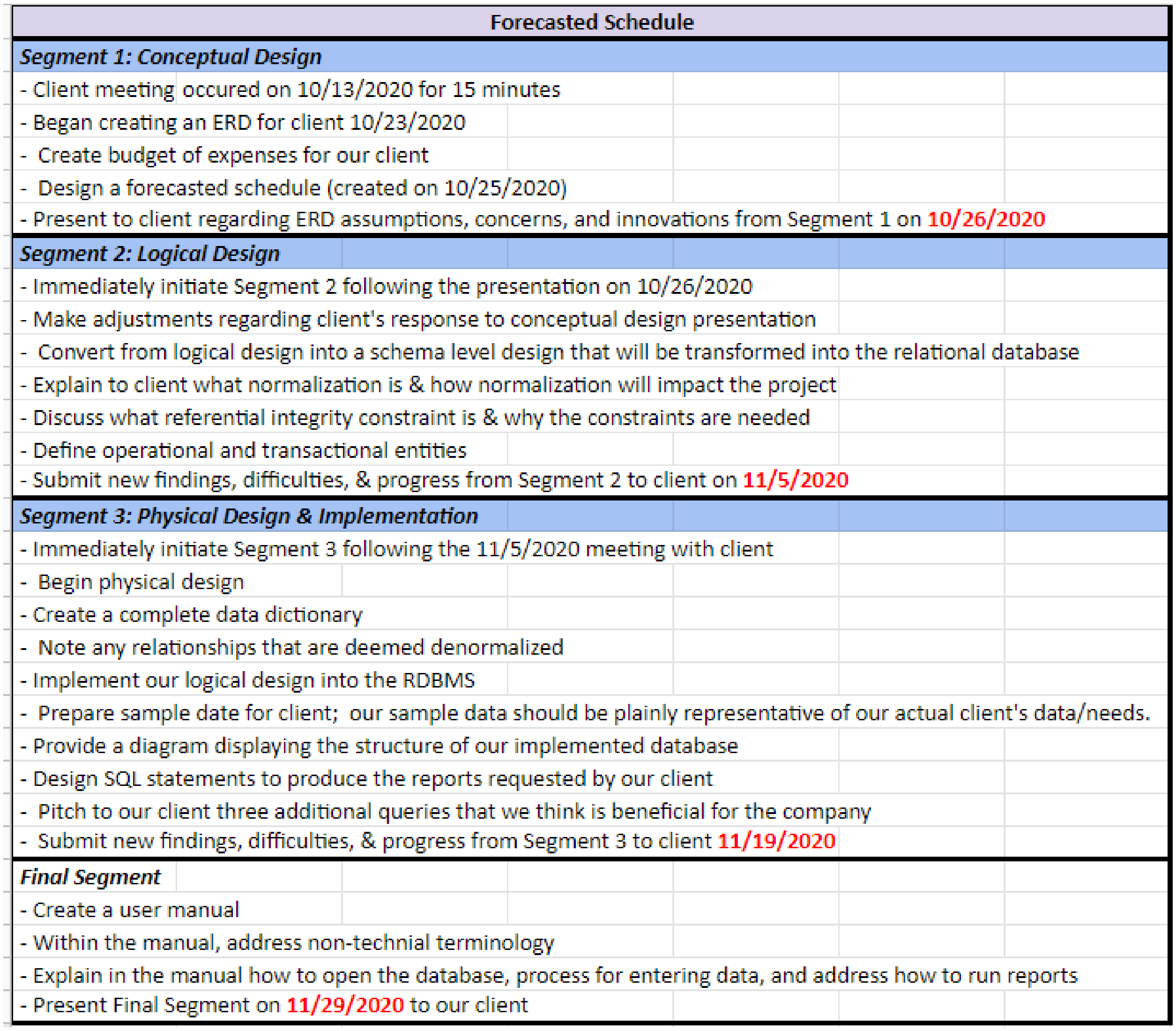
	VStreet		varchar(50)	not null	
	VPhone		varchar(14)	not null	
	VContactFirstName		varchar(50)	not null	
	VContactLastName		Varchar(50)	not null	
TCustomer	CustID	PK	int	Not null	
	CFName		Varchar(50)	Not null	
	CLName		Varchar(50)	Not null	
	CAddress		Varchar(50)	Not null	
	CState		Varchar(50)	Not null	
	CCity		Varchar(50)	Not null	
	CZip		Varchar(50)	Not null	
	CEmailAddress		Varchar(50)	Not null	
TCustType	CustTypeID	PK	int	Not null	
	CType		varchar(50)	Not null	
	CDescription		varchar(50)	Not null	
TDiscount	DiscountID	PK	int	Not null	
	DisTypeID	FK	int	Not null	
TDiscountType	DisTypeID	PK	int	Not null	
	DDisAmount		varchar(50)	Not null	

Project Management

This section contains the management details and processes we have used to complete this project efficiently and effectively, the forecasted schedule that details our milestone segments.



20								
Project Start Date	10/13/2020			Project End Date	11/29/2020		Cost (per 60 min)	$25
	Student Name	Duration (Min)	% Complete	Planned Minutes	Actual Minutes	Difference Minutes	Subtotal Minutes	Subtotal Cost
Milestone 1								
Read Case + Prepare Questions for client	Grace Fain	15	100%	15	14	1	14	5.833333333
	Grace Dennis	15	100%	15	14	1	14	5.833333333
	Allyson Cusato	15	100%	15	14	1	14	5.833333333
	Davis Dunkleberger	15	100%	15	14	1	14	5.833333333
	John Foster	15	100%	15	14	1	14	5.833333333
	Miles McClanahan	15	100%	15	14	1	14	5.833333333
Client Meeting 10/13/2020	Grace Fain	15	100%	15	15	0	15	6.25
	Grace Dennis	15	100%	15	15	0	15	6.25
	Allyson Cusato	15	100%	15	15	0	15	6.25
	Davis Dunkleberger	15	100%	15	15	0	15	6.25
	John Foster	15	100%	15	15	0	15	6.25
	Miles McClanahan	15	100%	15	15	0	15	6.25
ERD Design 10/23/2020	Grace Fain	90	100%	120	90	30	90	37.5
	Grace Dennis	90	100%	120	90	30	90	37.5
	Davis Dunkleberger	90	100%	120	90	30	90	37.5
	John Foster	90	100%	120	90	30	90	37.5
	Miles McClanahan	90	100%	120	90	30	90	37.5
Assumptions 10/25/2020	Allyson Cusato	40	100%	60	40	20	20	8.333333333
Write-up preparation 10/25/2020	Grace Fain	120	100%	120	120	0	120	50
	Grace Dennis	120	100%	120	120	0	120	50
	Davis Dunkleberger	120	100%	120	120	0	120	50
	John Foster	120	100%	120	120	0	120	50
	Allyson Cusato	120	100%	120	120	0	120	50
Sub Total						176	1264	$527
Milestone 2								
Preparation Meeting 11/12/2020	Grace Fain	50	100%	60	50	10	50	20.83333333
	Grace Dennis	50	100%	60	50	10	50	20.83333333
	Davis Dunkleberger	50	100%	60	50	10	50	20.83333333
	John Foster	50	100%	60	50	10	50	20.83333333
	Allyson Cusato	50	100%	60	50	10	50	20.83333333
Normalization	Grace Fain	45	100%	60	45	15	45	18.75
	Grace Dennis	60	100%	60	60	0	60	25
	Davis Dunkleberger	45	100%	60	45	15	45	18.75
	John Foster	60	100%	60	60	0	60	25
	Allyson Cusato	60	100%	60	60	0	60	25
	Miles McClanahan	60	100%	60	60	0	60	25
Write-up Preparation 11/15/2020	Grace Fain	90	100%	120	90	30	90	37.5
	Grace Dennis	90	100%	120	90	30	90	37.5
	Davis Dunkleberger	90	100%	120	90	30	90	37.5
	John Foster	90	100%	120	90	30	90	37.5
	Allyson Cusato	90	100%	120	90	30	90	37.5
Sub Total						230	1030	$429
Milestone 3								
Implementation	Grace Fain- Tables: SaleType, Employee, SaleOrder, SaleOrderLine	105	100%	120	105	15	105	43.75
	Grace Dennis-Tables: PurchaseOrder, PurchaseOrderLine, ShipType	105	100%	120	105	15	105	43.75
	Davis Dunkleberg-Tables: Product, GuidedTrip, Fly, Size, Pattern, Color	105	100%	120	105	15	105	43.75
	John Foster- Tables: Customer, CustType, Discount, DiscountType	105	100%	120	105	15	105	43.75
	Allyson Cusato-Tables: Vendor, TZipCode, TState	105	100%	120	105	15	105	43.75
	Miles McClanahan-Tables: Bundle, DeliverOut, EmployeeType	105	100%	120	105	15	105	43.75
Queries	Grace Fain- Queries 5,6, and one of our own	60	100%	60	60	0	60	25
	Davis Dunkleberger-Queries 1,2,and one of our own	60	100%	60	60	0	60	25
	John Foster-Queries 3,4 and one of our own	60	100%	60	60	0	60	25
Write Up 11/29/2020	Grace Fain	120	100%	120	120	0	120	50
	Grace Dennis	120	100%	120	120	0	120	50
	John Foster	120	100%	120	120	0	120	50
	Davis Dunkleberger	120	100%	120	120	0	120	50
	Allyson Cusato	60	100%	120	120	60	60	25
Sub Total						60	1410	$588
Final Submission								
Changes from Milestone 3 12/3/2020	Grace Fain -input data	60	100%	60	60	0	60	25
	Grace Dennis -input data	60	100%	60	60	0	60	25
	John Foster -input data	60	100%	60	60	0	60	25
	Davis Dunkleberger - input data	60	100%	60	60	0	60	25
	Allyson Cusato-input data	60	100%	60	60	0	60	25
Inserting data and queries 12/9/2020	Grace Fain-queries 10,11,12,13,14	120	100%	120	120	0	120	50
	Davis Dunkleberger - input data	120	100%	120	120	0	120	50
	John Foster -input data	120	100%	120	120	0	120	50
	Miles McClanahan -input data	120	100%	120	120	0	120	50
Write Up 12/9/2020	Grace Fain	240	100%	240	240	0	240	100
	Grace Dennis	240	100%	240	240	0	240	100
	John Foster	240	100%	240	240	0	240	100
	Davis Dunkleberger	240	100%	240	240	0	240	100
	Allyson Cusato	240	100%	240	240	0	240	100
	Miles McClanahan	240	100%	240	240	0	240	100
Sub Total						0	2220	$925
					Total		3704	$1,543



Forecasted Schedule
Segment 1: Conceptual Design
- Client meeting occured on 10/13/2020 for 15 minutes
- Began creating an ERD for client 10/23/2020
- Create budget of expenses for our client
- Design a forecasted schedule (created on 10/25/2020)
- Present to client regarding ERD assumptions, concerns, and innovations from Segment 1 on **10/26/2020**
Segment 2: Logical Design
- Immediately initiate Segment 2 following the presentation on 10/26/2020
- Make adjustments regarding client's response to conceptual design presentation
- Convert from logical design into a schema level design that will be transformed into the relational database
- Explain to client what normalization is & how normalization will impact the project
- Discuss what referential integrity constraint is & why the constraints are needed
- Define operational and transactional entities
- Submit new findings, difficulties, & progress from Segment 2 to client on **11/5/2020**
Segment 3: Physical Design & Implementation
- Immediately initiate Segment 3 following the 11/5/2020 meeting with client
- Begin physical design
- Create a complete data dictionary
- Note any relationships that are deemed denormalized
- Implement our logical design into the RDBMS
- Prepare sample date for client; our sample data should be plainly representative of our actual client's data/needs.
- Provide a diagram displaying the structure of our implemented database
- Design SQL statements to produce the reports requested by our client
- Pitch to our client three additional queries that we think is beneficial for the company
- Submit new findings, difficulties, & progress from Segment 3 to client **11/19/2020**
Final Segment
- Create a user manual
- Within the manual, address non-technial terminology
- Explain in the manual how to open the database, process for entering data, and address how to run reports
- Present Final Segment on **11/29/2020** to our client