# Ultimate Adventures Management System

(*MSSQL + MongoDB Implementation*)

Created by: Andrew Lane

04/04/16

CS3550

Visit the Repository

## Part 1: Project Overview

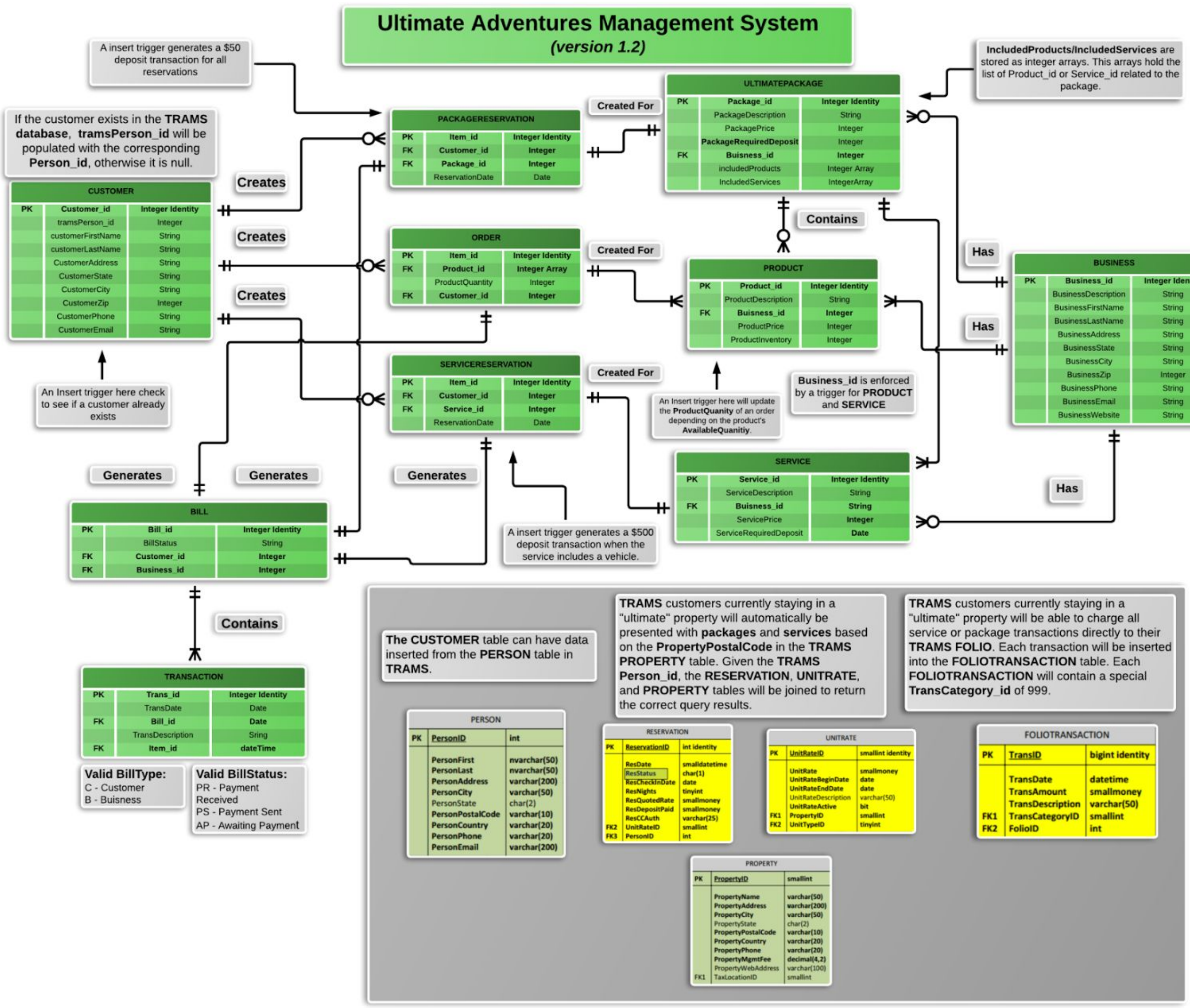
Ultimate Adventures (UA) is becoming every vacationer's go-to platform for crafting their dream get away. UA contains a team of expert adventurers that have scavenged the planet to find the best activities and tours in the industry. We have developed thousands of partnerships from all over the globe in order to bring our customers the best adventure booking experience. We are rapidly growing, and adding new businesses to our arsenal every week. We utilize these "ultimate" partnerships by bundling all kinds of goods/services together to make for the ultimate vacation. All of our packages come with our low price/no hassle "ultimate" guarantee. Our customers have the option to pick their adventurous goods/services al-la-carte as well.

Customers choose UA because they know all of our goods/services are "ultimate approved" with no hidden fees applied. We take the headache out of adventure booking by streamlining the entire process and delivering the best adventures straight to you. We bring together a variety of goods/services, so that the customer does not have to book through numerous outlets. There is only one payment, and we take care of the hassle that typically happens upon arrival at the activity site. The activities you want require you to sign disclosures/agreements? No problem. We have digitized everything and allow you to sign all documents electronically. If you happen to be vacationing at one of our "ultimate" vacation properties, it is an even faster and easier process. Each property location is associated with a variety of "ultimate" packages and because you are staying at an "ultimate" property, we give you an “ultimate” discount. This simplifies the booking process and gives the customer that relaxing vacation feeling even during the booking process.

**Overview of the UAMS Data Tier:**

The core functionality of UAMS is to provide customers the ability to search/book various “ultimate” goods and services (from a variety of partnered businesses). UAMS can store customer information, produce detailed financial statements, and distribute customer payments across all involved “ultimate” businesses. UAMS has partnered with the Timeshare Reservation Accounting Management System (TRAMS), to simplify the booking process for customers staying at an “ultimate” property. Leveraging TRAMS, UAMS can autofill customer information; automatically show goods/services affiliated with the “ultimate” property, and also bill the property directly. If a customer stays at “ultimate” properties frequently, UAMS will give the customer a bigger discount.

## Part 2: Data Specifications Overview


Ultimate Adventures Management System
(version 1.2)
A insert trigger generates a $50 deposit transaction for all reservations
If the customer exists in the TRAMS database, tramsPerson_id will be populated with the corresponding Person_id, otherwise it is null.
IncludedProducts/IncludedServices are stored as integer arrays. This arrays hold the list of Product_id or Service_id related to the package.
PACKAGERESERVATION
PK Item_id Integer Identity
FK Customer_id Integer
FK Package_id Integer
ReservationDate Date
ULTIMATEPACKAGE
PK Package_id Integer Identity
PackageDescription String
PackagePrice Integer
PackageRequiredDeposit Integer
FK Buisness_id Integer
includedProducts Integer Array
IncludedServices IntegerArray
Created For
Creates
Creates
Creates
CUSTOMER
PK Customer_id Integer Identity
tramsPerson_id Integer
customerFirstName String
customerLastName String
CustomerAddress String
CustomerState String
CustomerCity String
CustomerZip Integer
CustomerPhone String
CustomerEmail String
ORDER
PK Item_id Integer Identity
FK Product_id Integer Array
ProductQuantity Integer
FK Customer_id Integer
Created For
Contains
PRODUCT
PK Product_id Integer Identity
ProductDescription String
FK Buisness_id Integer
ProductPrice Integer
ProductInventory Integer
Has
Has
BUSINESS
PK Business_id Integer Identity
BusinessDescription String
BusinessFirstName String
BusinessLastName String
BusinessAddress String
BusinessState String
BusinessCity String
BusinessZip Integer
BusinessPhone String
BusinessEmail String
BusinessWebsite String
SERVICERESERVATION
PK Item_id Integer Identity
FK Customer_id Integer
FK Service_id Integer
ReservationDate Date
Created For
An Insert trigger here check to see if a customer already exists
An Insert trigger here will update the ProductQuanity of an order depending on the product's AvailableQuanitiy.
Business_id is enforced by a trigger for PRODUCT and SERVICE
SERVICE
PK Service_id Integer Identity
ServiceDescription String
FK Buisness_id String
ServicePrice Integer
ServiceRequiredDeposit Date
Has
Generates
Generates
Generates
BILL
PK Bill_id Integer Identity
BillStatus String
FK Customer_id Integer
FK Business_id Integer
A insert trigger generates a $500 deposit transaction when the service includes a vehicle.
Contains
TRANSACTION
PK Trans_id Integer Identity
TransDate Date
FK Bill_id Date
TransDescription Sring
FK Item_id dateTime
Valid BillType:
C - Customer
B - Buisness
Valid BillStatus:
PR - Payment Received
PS - Payment Sent
AP - Awaiting Payment
The CUSTOMER table can have data inserted from the PERSON table in TRAMS.
TRAMS customers currently staying in a "ultimate" property will automatically be presented with packages and services based on the PropertyPostalCode in the TRAMS PROPERTY table. Given the TRAMS Person_id, the RESERVATION, UNITRATE, and PROPERTY tables will be joined to return the correct query results.
TRAMS customers currently staying in a "ultimate" property will be able to charge all service or package transactions directly to their TRAMS FOLIO. Each transaction will be inserted into the FOLIOTRANSACTION table. Each FOLIOTRANSACTION will contain a special TransCategory_id of 999.
PERSON
PK PersonID int
PersonFirst nvarchar(50)
PersonLast nvarchar(50)
PersonAddress varchar(200)
PersonCity varchar(50)
PersonState char(2)
PersonPostalCode varchar(10)
PersonCountry varchar(20)
PersonPhone varchar(20)
PersonEmail varchar(200)
RESERVATION
PK ReservationID int identity
ResDate smalldatetime
ResStatus char(1)
ResCheckInDate date
ResNights tinyint
ResQuotedRate smallmoney
ResDepositPaid smallmoney
ResCCAuth varchar(25)
FK2 UnitRateID smallint
FK3 PersonID int
UNITRATE
PK UnitRateID smallint identity
UnitRate smallmoney
UnitRateBeginDate date
UnitRateEndDate date
UnitRateDescription varchar(50)
UnitRateActive bit
FK1 PropertyID smallint
FK2 UnitTypeID tinyint
FOLIOTRANSACTION
PK TransID bigint identity
TransDate datetime
TransAmount smallmoney
TransDescription varchar(50)
FK1 TransCategoryID smallint
FK2 FolioID int
PROPERTY
PK PropertyID smallint
PropertyName varchar(50)
PropertyAddress varchar(200)
PropertyCity varchar(50)
PropertyState char(2)
PropertyPostalCode varchar(10)
PropertyCountry varchar(20)
PropertyPhone varchar(20)
PropertyMgmtFee decimal(4,2)
PropertyWebAddress varchar(100)
FK1 TaxLocationID smallint

## Part 2: Data Specifications Overview (*continued*)

**Business Rules Satisfied by ERD and Normal Constraints:**

- All primary keys have identity values
- Tax is built in to every service, ultimate package, and product.

**Business Rules Satisfied by Triggers (all defined in MongoDB):**

1. A $50 deposit is required for all reservations.

- **Trigger Name: tr _addDepositForReservation**
- **Trigger Type:** AFTER trigger
- **Trigger Description:** A $50 deposit is required at checkout for **each different** vehicle (quad, jet ski, snow mobile, etc.). If a TRANSACTION is created from a UTLIMATEPACKAGE entry, this trigger will call the **addVehicleDeposit** stored procedure for each vehicle rental.

2. A $500 deposit is required at checkout for all vehicles (quad, jet ski, snow mobile, etc.)

- **Trigger Name: tr_addDepositForVehicle**
- **Trigger Type:** AFTER trigger
- **Trigger Description:** A $500 deposit is required at checkout for all vehicles (quad, jet ski, snow mobile, etc.). If a TRANSACTION is created from a SERVICERESERVATION, this trigger will call the addVehicleDeposit stored procedure which will add a new deposit transaction to the BILL.

3. The product quantity of an order cannot be greater than the product inventory.

- **Trigger Name: tr _checkStock**
- **Trigger Type:** INSERT trigger
- **Trigger Description:** When an order is created, the ***ProductQuantity*** is checked against PRODUCT's ***ProductInventory***. If ***ProductQuantity*** is less than ***AvailableQuantity****,* we call the **upd**
- **ateOrderQuantity** stored procedure which inserts the ***ProductQuantity*** value as the ***ProductInventory*** value.

4. A customer cannot have multiple CustomerID

- **Trigger Name: tr _checkExistingCustomer**
- **Trigger Type:** INSERT trigger
- **Trigger Description:** When a new customer is added, we double check to see if a customer with that **CustomerEmail** already exists, if so throw an error.

## Business Rules Satisfied by Stored Procedures/User Defined Functions (FUNCTIONS/UDFS are the same in MongoDB):

1. Businesses can update product inventory, and placed orders can update product inventory.

- **Function Name: *updateInventory()***

- **Function Description:** This function is designed to be called after an order has placed, or when a business has restocked – It will update a product's inventory amount accordingly.
- **Input Parameter 1:** ***ProductId*** (integer) – This is the corresponding **ProductID**, which will be updated.
- **Input Parameter 2:** ***Inventory*** (integer) – This is a positive or negative integer. The value of this parameter will be added/subtracted from ***productInventory***.

~~2. Customers vacationing at an "ultimate" property, receive a 5% discount. Customers vacationing at an "ultimate" property who have booked with UA more than twice receive a 10% discount.~~

- ~~**Function Name:** ***addUltimateDiscount()***~~
- ~~**Function Description:** This function is designed to be called if ***getTramsVacations()*** returns an integer greater than zero. If the return value of ***getTramsVacations()*** is less than 2, reduce bill amount by 5%. If the return value is greater than 2, reduce the bill amount by 10% *(called via trigger when transaction is added to bill).*~~
- ~~**Input Parameter: Vacations** (integer) – The amount of TRAMS vacations~~
- ~~**Returns:** Boolean on successful bill update.~~

**3.** Customers vacationing at an "ultimate" property, receive a 5% discount. Customers vacationing at an "ultimate" property who have booked with UA more than twice receive a 10% discount.

- **Function Name:** ***getTramsVacations()***
- **Function Description:** This function queries **TRAMS** – Given the ***PersonID***, this function will count the occurrences in the reservation table. This function also relies on the helper function ***customerExistsInTrams()***. If this function doesn't return a ***PersonID***, ***getTramsVacations()*** will log an error.
- **Input Parameter:** ***PersonID*** *(integer)* – The TRAMS ***PersonID***
- **Returns:** An Integer representation which reflects the Customer's total amount of TRAMS vacations.

**4.** Customers can view bills related to them.

- **Function Name:** ***produceCustomerBill()***
- **Function Description:** Given a customer's name *<first last>*, produce a bill (*stdout*) listing the customer details, transaction count, transaction details, and the total amount due.
- **Input Parameter:** ***customerName*** *(String)* – The customer's name *<first last>*
- **Returns:** The customer's bill, via console output.

**5.** Customers that exist in the TRAMS database can opt to have their customer information copied over to the Customer Table.

- **Function Name:** ***InsertTRAMSPersonData()***
- **Function Description**: Given a ***PersonId*** from TRAMS, add all related data to **Customer**.
- **Input Parameter: TRAMSPersonID** (*integer*)
- **Returns:** Inserts a new customer

**6.** If the customer wants to have the charge sent to their **TRAMS** Folio.

- **Function Name:** ***sendTransactionToTRAMS()***
- **Function Description:** If a customer exists in TRAMS and has a FOLIO, create a new FOLIOTransaction for that customer containing all the details from the transaction. Remove the transaction from the UAMS database.
- **Input Parameter:** ***transID*** (*integer*), ***customerID*** (integer)
- **Returns:** The **TRAMS *TransID*** and the **TRAMS *FolioID***, along with a success message *(stdout)*.

**7.** The system can create transactions when necessary

- **Function Name:** ***createBillTransaction()***
- **Function Description:** Given a BillID, create a transaction. The timestamp for this transaction will be generated when it is created.
- **Input Parameter:** ***BillID*** (integer), ***transactionDescription*** (string), ***transactionAmount*** *(integer)*,
- **Optional Parameter:** ***BillDescription*** (*string*)
- **Returns:** Adds a new transaction to the specified bill

**8. Check whether a customer exists in TRAMS (this is a helper function)**

- **Function Name:** ***customerExistsInTrams()***
- **Function Description:** Given a customer's name <first last> check whether or not they exist in the **TRAMS Person** table.
- **Input Parameter:** ***customerName*** (String) – The customer's name *<first last>*

- **Returns:** The TRAMS PersonID if the customer exists in the database, otherwise this will return null;