MAY 1, 2020

HIKING COMPANY DATABASE DEVELOPMENT

DO MAN UYEN NGUYEN
FABIO TURAZZI

1. Project Introduction

A hiking event organizer requests our service to build a database from scratch to efficiently manage data and support the business operations. The team will deliver a database system that allows the company to store data securely, retrieve data swiftly, and generate business queries accurately, ensuring data integrity and availability.

The desired database will include information about its customers and their payment information, company's available hiking packages, information about hiking locations, types of hiking event, and the photos taken from those events.

2. Requirements Gathering

The company for this project organizes group hiking events for its customers. The following details have been accumulated from the client's requirements and broken down into entities and attributes.

2.1. Entities:

Customer

Properties: name (composed of first and last name); telephone number (multi-valued attribute); customer ID (Surrogate Key); date of birth; date of registration; address (composed of street address, city, province, zip code).

Derived attribute (not stored): payment balance.

- Calculated using the packages acquired, from the Customer-Packages relationship, and the customer payments, from the Customer-Payment relationship. Controls customer pending payments.

General assumptions: All customers must pay an entry fee to become members. We also consider that customers must register to acquire packages.

Package

Properties: package ID (Surrogate Key); package price.

General Assumptions: There are different options of hike packages. We consider that there are different event categories and each package is related to one event category, with their relationship being stored on a separate table. Also, the packages contain different amounts of hikes, for example, 10 hikes for $500, 15 hikes for $700 and so on.

Place:

Properties: place id (Surrogate Key); address (composed of street address, city, province, zip code); topography (defining attribute for specialization); challenge level; estimated time of completion.

General Assumptions: General Place will be a superclass with two subclasses below it (Mountain and Plain) in an attribute-defined specialization.

Place Subclasses – Mountain and Plain:

Properties: elevation (Mountain subclass)

General Assumptions: places are broken down into types (Mountain and Plain) in an attribute-defined specialization.

Event / Event Category:

Properties - Event: event date (partial key); time; duration.

Properties – Event Category: event category; required equipment.

General Assumptions: Event will be a weak entity, having a strong entity Event Category as its owner. We based this decision in the assumption that Event Category will have a relationship with Packages, which would not be applicable to specific event entities (packages will be related to event types, not single events). We also assume that events from different categories may take place in the same date, but events from the same category cannot, so the date can be used as a partial key for the weak entity.

Payment:

Properties: payment id; payment date; payment amount.

General Assumptions: We will store data about payments, including package payments and entry fee payments. Payment can be done full amount once or it can be paid in installments on different dates.

Photos:

Properties: photo directory.

General Assumptions: We will store information from photos taken in events. Photos will be shared only with customers that participated in that event.

2.2. Data Relationship Assumptions - Cardinality and Participations

Assumption 1 – Customer-Package relationship (Customers acquire Packages)

- **Cardinality:** Many-to-many – multiple customers acquire each package; customers may acquire more than one package.
- **Customer Participation:** Partial – some new customers will not have acquired any packages.
- **Package Participation:** Partial – some packages may not be acquired by anyone.

Assumption 2 – Event-Place relationship (Events are located in Places)

- **Cardinality:** Many-to-one – events happen at a single location, more than one event may happen in each location.
- **Event Participation:** Total – all events need a location.
- **Place Participation:** Partial – relevant data from locations in which no events took place yet will be stored.

Assumption 3 – Customer-Event relationship (Customers participate in Events)

- **Cardinality:** Many-to-many – several participants attend a single event, and every customer may attend multiple events.
- **Customer Participation:** Partial – some stored customers will not have attended an event.
- **Event Participation:** Total – every event is attended by a customer.

Assumption 4 – Customer-Payment relationship (Customers make Payments)

- **Cardinality:** One-to-many – customers may perform multiple payments, but every payment is performed by a single customer.
- **Customer Participation:** Total – all registered customers must pay the entry fee.

- **Payment Participation:** Total – all payments were done by a customer.

Assumption 5 – Payment-Package relationship (Payment is linked to Package)

- **Cardinality:** Many-to-many – Customer can make multiple payments for each purchased package, and one payment can be made for multiple packages.
- **Package Participation:** Partial – Some payments are related to entry fees, not packages.
- **Payment Participation:** Partial – Not all customers purchase a package.

Assumption 6 – Event Category-Package relationship (Event Category is included in Package)

- **Cardinality:** One-to-many – Each package contains 1 event category, but multiple packages may refer to the same category (with different number of events).
- **Package Participation:** Total – Every package will be related to an event category.
- **Event Category Participation:** Total – Every event category will be contained in a package.

Assumption 7 – Event-Event Category relationship (Event belongs to Event Category)

- **Cardinality:** Many-to-one – Multiple events belong to a single category, but each event has only one category.
- **Event Participation:** Total – All events have a category.
- **Event Category Participation:** Total – All categories have events from its type.

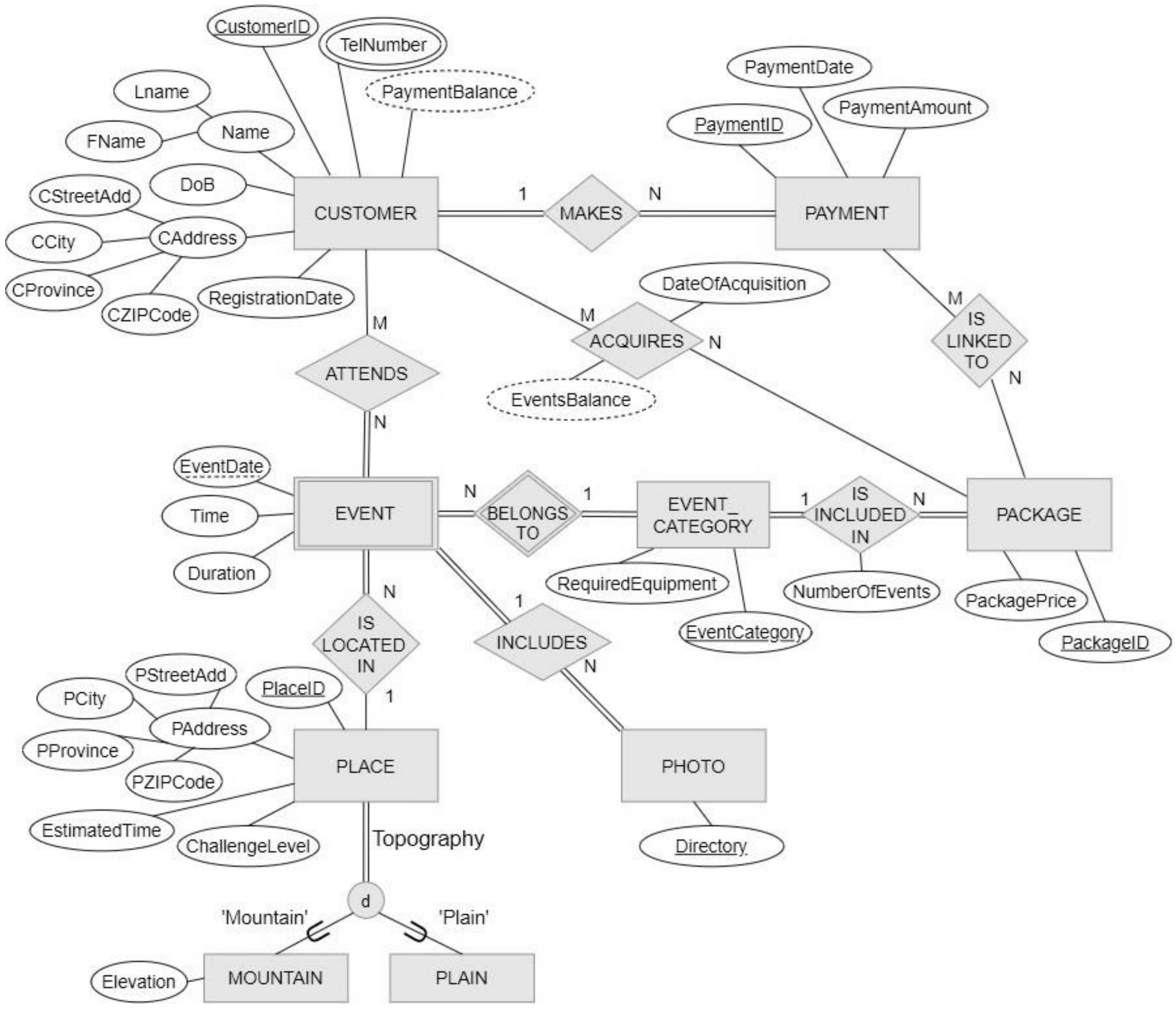
Assumption 8 – Event-Photo relationship (Event includes Photos)

- **Cardinality:** one-to-many – Events will have multiple photos and each photo will be related to a single event.
- **Event Participation:** Total – All events will have photos.
- **Photo Participation:** Total – All stored photos will be from a given event.

3. Database Design

3.1. Conceptual EER Modeling

Using the assumptions established in items 5 and 4, the following EER was developed for the Hiking Events Company database:

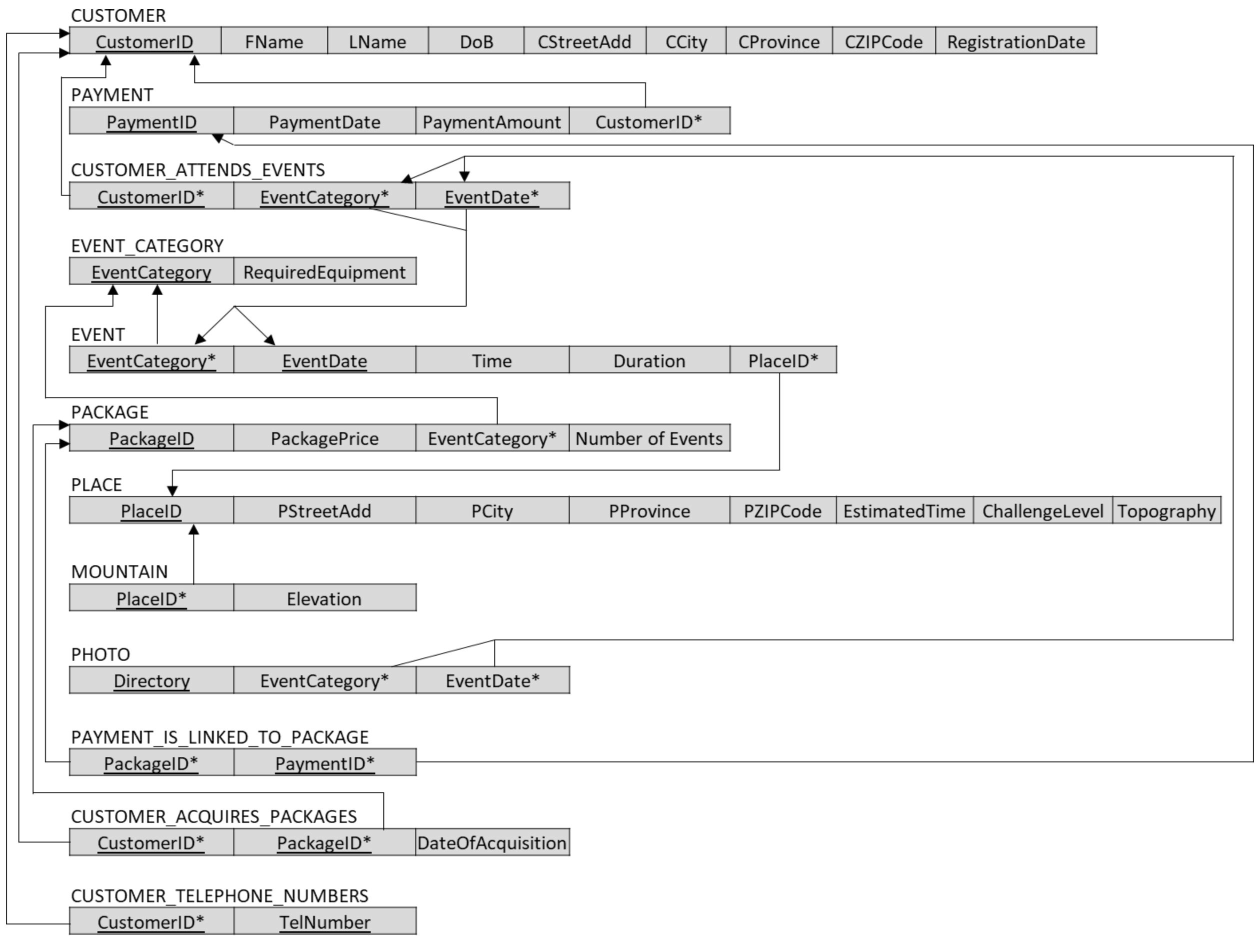


3.2. ER-Model Mapping to Database Relational Schema

Considering the ERD shown in item 6, we established the following relations for the database schema:

- CUSTOMER (CustomerID, FName, LName, DoB, CStreetAdd, CCity, CProvince, CZIPCode, RegistrationDate)
- PAYMENT (PaymentID, PaymentDate, PaymentAmount, CustomerID*)
- EVENT_CATEGORY (EventCategory, RequiredEquipment)
- PACKAGE (PackageID, PackagePrice, EventCategory*, Number of Events)
- PHOTO (Directory, EventCategory*, EventDate*)
- EVENT (EventCategory*, EventDate, Time, Duration, PlaceID*)
- CUSTOMER_ATTENDS_EVENTS (CustomerID*, EventCategory*, EventDate*)
- CUSTOMER_ACQUIRES_PACKAGES (CustomerID*, PackageID*, DateOfAcquisition)
- PAYMENT_IS_LINKED_TO_PACKAGE (PaymentID*, PackageID*)
- CUSTOMER_TELEPHONE_NUMBERS (CustomerID*, TelNumber)
- PLACE (PlaceID, PStreetAdd, PCity, PProvince, PZIPCode, EstimatedTime, ChallengeLevel, Topography)
- MOUNTAIN (PlaceID*, Elevation)

The schema below describes the relationships, indicating the references of each foreign key. The relationships were reorganized to provide better visualization of each connection, not reflecting the order in which the mapping was made as displayed on the list above.



4. Normalization

The following demonstrations intend to prove that all relations contained in the database are normalized up to BCNF.

First Normal Form (1NF)

The entities that contain potentially non-atomic values (multivalued or composite attributes) are CUSTOMER (Address and Telephone Number) and PLACE (Address). Both Address attributes were broken down into Street Address, City, Province and ZIP Code. For Telephone Number, the attribute was placed in a separate relation (CUSTOMER_TELEPHONE_NUMBERS). Additionally, no nested relations were considered in this model (every tuple is independent).

Second Normal Form (2NF)

The keys chosen for each relation are determining attributes for all other components of the respective relation. The functional dependencies are shown below, with a brief explanation included for the most complex cases:

Functional Dependencies

Customer

- CustomerID -> { FName, LName, DoB, CStreetAdd, CCity, CProvince, CZIPCode, RegistrationDate }

Payment

- PaymentID -> { PaymentDate, PaymentAmount, CustomerID }

Event_Category

- EventCategory -> RequiredEquipment
    - Each event category requires a specific equipment, therefore, it may be used to define equipment.

Package

- PackageID -> { PackagePrice, EventCategory, Number of Events }

Photo

- Directory -> { EventCategory, EventDate }
    - Each photo will have a unique directory in the company's LAN, therefore, the related event can be identified using the directory.

Event

- { EventCategory, EventDate } -> { Time, Duration, PlaceID }
    - Events may occur simultaneously, but only a single event from each category will occur at a given date. Event category and date together, then, determine the remaining attributes.

Customer_Attends_Events

- { CustomerID, EventCategory, EventDate } -> { CustomerID, EventCategory, EventDate }
    - Since all attributes in the relation compose the primary key, the only functional dependency required (and possible) is from the key with itself.

Customers_Acquire_Packages

- { CustomerID, PackageID } -> DateOfAcquisition
    - Both customer and package ID are required to determine the date of acquisition in this relation.

Payment_Linked_to_Package:

- { PaymentID, PackageID } -> { PaymentID, PackageID }
  - Since all attributes in the relation compose the primary key, the only functional dependency required (and possible) is from the key with itself.

Customer_Telephone_Number

- { CustomerID, TelNumber } -> { CustomerID, TelNumber }
  - Since all attributes in the relation compose the primary key, the only functional dependency required (and possible) is from the key with itself.

Place

- PlaceID -> { PStreetAdd, PCity, PProvince, PZIPCode, EstimatedTime, ChallengeLevel, Topography }

Mountain

- PlaceID -> Elevation

Third Normal Form (3NF)

All the relations in the database are in 3NF. None of them present Transitive Functional Dependencies, since no attributes in the right side of the aforementioned Functional Dependencies can determine each other without help of the primary key.

The only potential TFDs identified were between ZIP Codes and Street Addresses, so we made the following analysis of their potential functional dependencies:

- Similar Street Addresses in different cities possess two different ZIP Codes. Additionally, some Street Addresses posses more than one ZIP Code:
  - StreetAdd -> ZIPCode is not a valid FD.
- Some ZIP Codes point to two different Street Addresses, since the ZIP Code is a logical address used by Post Offices:
  - ZIPCode -> StreetAdd is not a valid FD.

Boyce-Codd Normal Form (BCNF)

Since no Transitive Functional Dependencies were found in our analysis for 3NF in any of our relation attributes, we can infer that no candidate-key TFD is possible. We can then conclude that our whole database is in accordance with Boyce-Codd Normal Form.

5. Determining Data Types (Domain) and Constraints

CUSTOMER

Column Name	Data Type	Nullable	Constraint	Key
CUSTOMERID	NUMBER	No	-	PK
FNAME	VARCHAR2(35)	No	-	-
LNAME	VARCHAR2(35)	No	-	-
DOB	DATE	No	-	-
CSTREETADD	VARCHAR2(100)	Yes	-	-
CCITY	VARCHAR2(20)	Yes	-	-
CPROVINCE	VARCHAR2(20)	Yes	-	-
CZIPCODE	VARCHAR2(9)	Yes	-	-
REGISTRATIONDATE	DATE	No	-	-

PAYMENT

Column Name	Data Type	Nullable	Constraint	Key
PAYMENTID	NUMBER	No	-	PK
PAYMENTDATE	DATE	No	-	-
PAYMENTAMOUNT	NUMBER	No	-	-
CUSTOMERID	NUMBER	Yes	ON DELETE SET NULL	FK

EVENT_CATEGORY

Column Name	Data Type	Nullable	Constraint	Key
EVENTCATEGORY	VARCHAR2(20)	No	-	PK
REQUIREDEQUIPMENT	VARCHAR2(50)	Yes	-	-

PACKAGES

Column Name	Data Type	Nullable	Constraint	Key
PACKAGEID	NUMBER	No	-	PK
PACKAGEPRICE	NUMBER	No	-	-
EVENTCATEGORY	VARCHAR2(20)	No	ON DELETE CASCADE	FK
NUMBEROFEVENTS	NUMBER	No	-	-

PHOTO

Column Name	Data Type	Nullable	Constraint	Key
DIRECTORY	VARCHAR2(60)	No	-	PK
EVENTCATEGORY	VARCHAR2(20)	Yes	ON DELETE SET NULL	FK
EVENTDATE	DATE	Yes	ON DELETE SET NULL	FK

EVENT

Column Name	Data Type	Nullable	Constraint	Key
EVENTCATEGORY	VARCHAR2(20)	No	ON DELETE CASCADE	PK, FK
EVENTDATE	DATE	No	-	PK
STARTINGTIME	VARCHAR2(8)	No	-	-
DURATION	NUMBER	No	-	-
PLACEID	NUMBER	Yes	ON DELETE SET NULL	FK

CUSTOMER_ATTENDS_EVENTS

Column Name	Data Type	Nullable	Constraint	Key
CUSTOMERID	NUMBER	No	ON DELETE CASCADE	PK, FK
EVENTCATEGORY	VARCHAR2(20)	No	ON DELETE CASCADE	PK, FK
EVENTDATE	DATE	No	ON DELETE CASCADE	PK, FK

CUSTOMER_ACQUIRES_PACKAGES

Column Name	Data Type	Nullable	Constraint	Key
CUSTOMERID	NUMBER	No	ON DELETE CASCADE	PK, FK
PACKAGEID	NUMBER	No	ON DELETE CASCADE	PK, FK
DATEOFACQUISITION	DATE	No	-	-

PAYMENT_IS_LINKED_TO_PACKAGE

Column Name	Data Type	Nullable	Constraint	Key
PAYMENTID	NUMBER	No	ON DELETE CASCADE	PK, FK
PACKAGEID	NUMBER	No	ON DELETE CASCADE	PK, FK

CUSTOMER_TELEPHONE_NUMBERS

Column Name	Data Type	Nullable	Constraint	Key
CUSTOMERID	NUMBER	No	ON DELETE CASCADE	PK, FK
TELNUMBER	NUMBER(10,0)	No	-	PK

PLACE

Column Name	Data Type	Nullable	Constraint	Key
PLACEID	NUMBER	No	-	PK
PSTREETADD	VARCHAR2(100)	No	-	-
PCITY	VARCHAR2(20)	No	-	-
PPROVINCE	VARCHAR2(20)	No	-	-
PZIPCODE	VARCHAR2(9)	No	-	-
ESTIMATEDHOURS	NUMBER	Yes	-	-
CHALLENGELEVEL	NUMBER	Yes	-	-
TOPOGRAPHY	VARCHAR2(10)	No	-	-

MOUNTAIN

Column Name	Data Type	Nullable	Constraint	Key
PLACEID	NUMBER	No	ON DELETE CASCADE	PK, FK
ELEVATION	VARCHAR2(15)	Yes	-	-

6. Creating Database and Tables - SQL DDL

```sql
CREATE TABLE CUSTOMER(
    CustomerID INT,
    FName VARCHAR(35) NOT NULL,
    LName VARCHAR(35) NOT NULL,
    DoB DATE NOT NULL,
    CStreetAdd VARCHAR(100),
    CCity VARCHAR(20),
    CProvince VARCHAR(20),
    CZIPCode VARCHAR(9),
    RegistrationDate DATE NOT NULL,
    PRIMARY KEY (CustomerID)
);
CREATE TABLE PAYMENT(
    PaymentID INT,
    PaymentDate DATE NOT NULL,
    PaymentAmount DECIMAL NOT NULL CHECK(PaymentAmount > 0),
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE SET NULL,
    PRIMARY KEY (PaymentID)
);
CREATE TABLE EVENT_CATEGORY(
    EventCategory VARCHAR(20),
    RequiredEquipment VARCHAR(50),
    PRIMARY KEY (EventCategory)
);
CREATE TABLE PACKAGES(
    PackageID INT,
    PackagePrice DECIMAL NOT NULL CHECK(PackagePrice > 0),
    EventCategory VARCHAR(20) NOT NULL,
    NumberOfEvents SMALLINT NOT NULL CHECK(NumberOfEvents > 0),
    PRIMARY KEY (PackageID),
    FOREIGN KEY (EventCategory) REFERENCES EVENT_CATEGORY(EventCategory) ON DELETE CASCADE
);
CREATE TABLE PLACE(
    PlaceID INT,
    PStreetAdd VARCHAR(100) NOT NULL,
    PCity VARCHAR(20) NOT NULL,
    PProvince VARCHAR(20) NOT NULL,
    PZIPCode VARCHAR(9) NOT NULL,
    EstimatedHours SMALLINT CHECK(EstimatedHours > 0 AND EstimatedHours < 12),
    ChallengeLevel SMALLINT CHECK(ChallengeLevel >= 0 AND ChallengeLevel <= 10),
    Topography VARCHAR(10) NOT NULL,
    PRIMARY KEY (PlaceID)
);
CREATE TABLE MOUNTAIN(
    PlaceID INT,
    Elevation VARCHAR(15),
    PRIMARY KEY (PlaceID),
    FOREIGN KEY (PlaceID) REFERENCES PLACE(PlaceID) ON DELETE CASCADE
);
CREATE TABLE EVENT(
    EventCategory VARCHAR(20),
    EventDate DATE,
    StartingTime VARCHAR(8) NOT NULL,
    Duration SMALLINT CHECK(Duration > 0 AND Duration < 12) NOT NULL,
    PlaceID INT,
    PRIMARY KEY(EventCategory, EventDate),
    FOREIGN KEY (EventCategory) REFERENCES EVENT_CATEGORY(EventCategory) ON DELETE CASCADE,
    FOREIGN KEY (PlaceID) REFERENCES PLACE(PlaceID) ON DELETE SET NULL
);
CREATE TABLE PHOTO(
    Directory VARCHAR(60),
    EventCategory VARCHAR(20),
    EventDate DATE,
    PRIMARY KEY (Directory),
    FOREIGN KEY (EventCategory, EventDate) REFERENCES EVENT(EventCategory, EventDate) ON DELET
E SET NULL
);
CREATE TABLE CUSTOMER_ATTENDS_EVENTS(
    CustomerID INT,
    EventCategory VARCHAR(20),
    EventDate DATE,
    PRIMARY KEY (CustomerID,EventCategory, EventDate),
    FOREIGN KEY (CustomerID) REFERENCES CUSTOMER(CustomerID) ON DELETE CASCADE,
    FOREIGN KEY (EventCategory, EventDate) REFERENCES EVENT(EventCategory, EventDate) ON DELET
E CASCADE
);
CREATE TABLE CUSTOMER_ACQUIRES_PACKAGES(
    CustomerID INT,
    PackageID INT,
    DateOfAcquisition DATE NOT NULL,
    PRIMARY KEY (CustomerID, PackageID),
    FOREIGN KEY (CustomerID) REFERENCES CUSTOMER(CustomerID) ON DELETE CASCADE,
    FOREIGN KEY (PackageID) REFERENCES PACKAGES(PackageID) ON DELETE CASCADE
);
CREATE TABLE PAYMENT_IS_LINKED_TO_PACKAGE(
    PaymentID INT,
    PackageID INT,
    PRIMARY KEY (PaymentID,PackageID),
    FOREIGN KEY (PackageID) REFERENCES PACKAGES (PackageID) ON DELETE CASCADE,
    FOREIGN KEY (PaymentID) REFERENCES PAYMENT (PaymentID) ON DELETE CASCADE
);
CREATE TABLE CUSTOMER_TELEPHONE_NUMBERS(
    CustomerID INT,
    TelNumber NUMERIC(10,0) NOT NULL,
    PRIMARY KEY (CustomerID, TelNumber),
    FOREIGN KEY (CustomerID) REFERENCES CUSTOMER (CustomerID) ON DELETE CASCADE
);
```

7. Table Indexing

Ensuring the efficiency of our Database, indexes were created for each table to allow a quicker processing for common queries. For this step, the criteria used for the indexes considered properties that are **commonly searched** in each table and **do not** have their **values changed frequently**.

```
CREATE UNIQUE INDEX CUSTOMER_NAME_INDEX ON CUSTOMER (FName);
CREATE UNIQUE INDEX PAYMENT_DATE_INDEX ON PAYMENT (PaymentDate);
CREATE UNIQUE INDEX PACKAGE_CATEGORY_INDEX ON PACKAGE (EventCategory);
CREATE UNIQUE INDEX PLACE_CITY_INDEX ON PLACE (PCity);
CREATE UNIQUE INDEX PLACE_CHALLENGE_INDEX ON PLACE (ChallengeLevel);
CREATE UNIQUE INDEX EVENT_DATE_INDEX ON EVENT (EventDate);
CREATE UNIQUE INDEX PHOTO_DATE_INDEX ON PHOTO (EventDate);
CREATE UNIQUE INDEX CUSTOMER_ATTENDANCE_INDEX ON CUSTOMER_ATTENDS_EVENTS (EventDate);
CREATE UNIQUE INDEX DATE_INDEX ON CUSTOMER_ACQUIRES_PACKAGES (DateOfAcquisition);
CREATE UNIQUE INDEX PAYMENT_PACKAGE_INDEX ON PAYMENT_IS_LINKED_TO_PACKAGE (PackageID);
CREATE UNIQUE INDEX PHONE_CUSTOMER_INDEX ON CUSTOMER_TELEPHONE_NUMBERS (CustomerID);
```

8. Inserting Values in Tables

The following SQL statements for table population represent one sample command for each table. The rest of the insert statements can be found in the **data/Populate_Tables.sql** file of this project directory.

```sql
INSERT INTO Customer VALUES (1, 'John', 'Doe', '11/10/1981', '8960 Street 6', 'Vancouver','Bri
tish Columbia', 'A3T BC2', '04/06/2019');
INSERT INTO PLACE VALUES (1, '5500 Cookie Ave', 'Vancouver', 'British Columbia', 'BC2 A3T', 4,
8, 'Mountain');
INSERT INTO PAYMENT VALUES (1, '03/28/2017', 1900.00, 4);
INSERT INTO EVENT_CATEGORY VALUES ('Hiking Overhaul', 'Complete Hiking Gear');
INSERT INTO PACKAGES VALUES (1, 1900, 'Hiking Overhaul', 10);
INSERT INTO MOUNTAIN VALUES (1, 'High');
INSERT INTO EVENT VALUES ('Hiking Overhaul', '09/22/2017', '08:00 AM', 8, 1);
INSERT INTO PHOTO VALUES ('D:/Photos/Hiking Overhaul/09_22_2017 Photo 1', 'Hiking Overhaul', '
09/22/2017');
INSERT INTO CUSTOMER_ATTENDS_EVENTS VALUES (1, 'Hiking Overhaul', '09/22/2017');
INSERT INTO CUSTOMER_ACQUIRES_PACKAGES VALUES (4, 1, '03/25/2017');
INSERT INTO PAYMENT_IS_LINKED_TO_PACKAGE VALUES (1, 1);
INSERT INTO CUSTOMER_TELEPHONE_NUMBERS VALUES (1, 7785162003);
```

9. Statements for Dropping Tables

```sql
DROP TABLE IF EXISTS PHOTO;
DROP TABLE IF EXISTS CUSTOMER_ATTENDS_EVENTS;
DROP TABLE IF EXISTS PAYMENT_IS_LINKED_TO_PACKAGE;
DROP TABLE IF EXISTS CUSTOMER_TELEPHONE_NUMBERS;
DROP TABLE IF EXISTS CUSTOMER_ACQUIRES_PACKAGES;
DROP TABLE IF EXISTS MOUNTAIN;
DROP TABLE IF EXISTS PAYMENT;
DROP TABLE IF EXISTS PACKAGES;
DROP TABLE IF EXISTS EVENT;
DROP TABLE IF EXISTS EVENT_CATEGORY;
DROP TABLE IF EXISTS PLACE;
DROP TABLE IF EXISTS CUSTOMER;
```

10. Creating Efficient Queries and Reports

```
-- Join & Aggregate functions: Calculate each customer’s monetary balance
SELECT PAYMENT.CustomerID, SUM(DISTINCT PAYMENT.PaymentAmount) AS "Total Paid", SUM(DISTINCT P
ACKAGES.PackagePrice) AS "Total Purchased",
        SUM(DISTINCT PAYMENT.PaymentAmount) - SUM(DISTINCT PACKAGES.PackagePrice) AS "Balance"
    FROM PAYMENT INNER JOIN CUSTOMER_ACQUIRES_PACKAGES
    ON CUSTOMER_ACQUIRES_PACKAGES.CustomerID = PAYMENT.CustomerID
    INNER JOIN PACKAGES ON PACKAGES.PackageID = CUSTOMER_ACQUIRES_PACKAGES.PackageID
        GROUP BY PAYMENT.CustomerID;

-- Anti join: Show customers that did not acquire any packages
SELECT DISTINCT(CustomerID) AS "Customers Without Packages" FROM CUSTOMER
    LEFT JOIN CUSTOMER_ACQUIRES_PACKAGES ON CUSTOMER.CustomerID = CUSTOMER_ACQUIRES_PACKAGES.C
ustomerID
        WHERE PackageID = NULL;

-- Window functions & create view: Querying customer data for RFM analysis
CREATE VIEW CUSTOMER_RFM AS SELECT CustomerID,
    SUM(PaymentAmount) OVER
         (PARTITION BY CustomerID) AS "Monetary Value",
    COUNT(PaymentID) OVER
         (PARTITION BY CustomerID) AS "Frequency Value",
    MAX(PaymentDate) OVER
         (PARTITION BY CustomerID) AS "Recency Value"
  FROM PAYMENT
  ORDER BY "Total Paid";

-- Nested query: Identify customers that attended events of 2 different types
SELECT CustomerID, COUNT(CustomerID) AS "Events Attended", EventCategory FROM CUSTOMER_ATTENDS
_EVENTS
    WHERE (EventCategory = 'Easy Peasy Hike' AND CustomerID IN (SELECT CustomerID FROM CUSTOME
R_ATTENDS_EVENTS WHERE EventCategory = 'Easy Stroll'))
    OR (EventCategory = 'Easy Stroll' AND CustomerID IN (SELECT CustomerID FROM CUSTOMER_ATTEN
DS_EVENTS WHERE EventCategory = 'Easy Peasy Hike'))
        GROUP BY CustomerID, EventCategory;
```

```
-- With Statement: Calculate customer event balance and show the ones > 8
WITH Event_Balance(CustomerID, EventCategory, EventsAcquired, EventsAttended, Balance) AS
            (SELECT CAP.CUSTOMERID, PKG.EVENTCATEGORY, SUM(DISTINCT PKG.NUMBEROFEVENTS), COUNT
(DISTINCT CAE.EVENTCATEGORY),
            (SUM(DISTINCT PKG.NUMBEROFEVENTS) - COUNT(DISTINCT CAE.EVENTCATEGORY))
            FROM (CUSTOMER_ACQUIRES_PACKAGES CAP INNER JOIN PACKAGES PKG ON CAP.PACKAGEID = PK
G.PACKAGEID)),
    SELECT CustomerID, EventCategory, EventsAcquired, EventsAttended, Balance
    FROM Event_Balance
    ORDER BY Balance;

-- Case Statement: Calculate taxes to be paid, considering year of payment.
SELECT PaymentDate, PaymentID, PaymentAmount,
    CASE
        WHEN YEAR(PaymentDate) <= 2015 THEN PaymentAmount*0.2
        WHEN YEAR(PaymentDate) <= 2017 THEN PaymentAmount*0.25
        WHEN YEAR(PaymentDate) <= 2020 THEN PaymentAmount*0.3
    END AS "Total Taxes"
    FROM PAYMENT ORDER BY PaymentDate;

-- Aggregation with substring: Show packages acquired by month to identify seasonality.
SELECT SUBSTR(REGISTRATIONDATE,1,2) AS "Month", COUNT(CUSTOMERID) AS "Packages Acquired"
    FROM CUSTOMER GROUP BY SUBSTR(REGISTRATIONDATE,1,2)
        ORDER BY SUBSTR(REGISTRATIONDATE,1,2);

-- Aggregation: Show photos to be sent for each customer, given their event attendance.
SELECT CAE.CUSTOMERID, P.DIRECTORY AS "Photos to Send"
    FROM PHOTO P INNER JOIN CUSTOMER_ATTENDS_EVENTS CAE ON P.EVENTDATE = CAE.EVENTDATE AND P.E
VENTCATEGORY=CAE.EVENTCATEGORY
        ORDER BY CAE.CUSTOMERID;

-- Update statement: Update price of 2 event categories, increasing it by 10%.
UPDATE PACKAGES SET PackagePrice = PackagePrice * 1.1
    WHERE EventCategory = 'Hiking Overhaul' OR EventCategory = 'Intermediate Hike';

-- Delete statement: Delete a type of package.
DELETE FROM PACKAGES
    WHERE EventCategory = 'Easy Peasy Hike';

-- Create view: Create a view of photos from advanced level events.
CREATE VIEW AdvancedPhotos AS SELECT * FROM Photo
    WHERE EventCategory = 'Hiking Overhaul' OR  EventCategory = 'Intermediate Hike';
```