MAY 1, 2020

HIKING COMPANY DATABASE DEVELOPMENT

DO MAN UYEN NGUYEN
FABIO TURAZZI

Contents

1. User requirement Analysis

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, at the same time, ensures 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. Project Details

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.

Entities:

Customer

We store the following attributes of customers:

- 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);

Additionally, the entity will have one derived attribute (non-stored):

- payment balance (calculated using the packages acquired, from the Customer-Packages relationship, and the customer payments, from the Customer-Payment relationship).
  - This information will be important to control if the customer has pending payments to make.

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

Package:

There are different packages of payment. We consider that there are different event categories, and each package is related to one event category. Also, the packages contain different amounts of hikes, for example, 10 hikes for $500, 15 hikes for $700 and so on. For the package entity, we will store:

- package ID (Surrogate Key);
- package price.

Place:

Place will be a superclass with two subclasses below it (Mountain and Plain), in an attribute-defined specialization. We store the following information about the Place superclass:

- place id (Surrogate Key);
  - Necessary to distinguish different tracks in a given location.
- address (composed of street address, city, province, zip code);
- topography;
  - This is the defining attribute in the specialization;
- challenge level;

- estimated time of completion.

Place Subclasses – Mountain and Plain:

Only the subclass Mountain has a local attribute, shown below:

- elevation.

Event / Event Category:

Event will be a weak entity, that will have 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 make the assumption that events from different categories may take place in the same date, but events from the same category can't, so the date can be used as a partial key for the weak entity. We will store the following attributes for the entities:

Event

- event date (partial key);
- time;
- duration;

Event_Category

- event category;
- required equipment.

Payment:

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. We store:

- payment ID;
- payment date;
- payment amount.

Photos:

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

- directory;

Relationships

Customer - Package

For the relationship between customers and packages (customers acquire packages), we will store the date of the acquisition (from which we can calculate the package's expiration date for that customer). The relationship will also have a derived attribute (non-stored) called Events Balance, calculated using the number of events a customer attended and the total events his packages have. This attribute is important to keep track of the customer's packages usage.

This relationship's cardinality will be many-to-many, since many customers will acquire a package and it's possible for them to acquire additional packages, if necessary (after his current hired package runs out of balance). Additionally, both the customers and packages will have partial participation, since new customers may not acquire any packages and some packages may not be acquired by anyone.

Customer - Event

Here, we will store information about members who participated in any event. When the event was done, we would like to store photos taken on that event and share it with only those who participated in the event.

This will configure a relationship between customers and events (customers participate in events), with a many-to-many relationship, since several participants will attend an event, and each customer attends multiple events. In this relationship, events exert a total participation, since every event will have participants, but customers will have a partial participation, since we'll have customers who didn't attend events yet.

Payment - Customer

This relationship (customer makes payments) has a one-to-many relationship (since customers can perform several different payments, but each payment is performed by a single customer). The payment entity's participation will be total, since all payments we're done by a given customer, and the customer's participation is total, since all customers are required to pay at least an entry fee to register.

Payment - Package

This relationship (payment is linked to package) has a many-to-many relationship, since a payment may cover multiple packages and a package may be paid for in multiple stages. Additionally, both participations are partial, since some payments will not be related to packages (in case of entry fees) and some packages may not be purchased by anyone at a given moment.

Package – Event Category

This relationship (event category is included in package) has a 1-to-many relationship, since each package contains 1 event category, but multiple packages may refer to the same category (with different number of events). Both participations in this relationship will be total, since every package will be related to an event category and vice-versa. We also store package's number of hikes/events for this relationship.

Event – Event Category

This is the identifying relationship for events (event belongs to event category), and has a many-to-one relationship, since multiple events belong to a single category, but each event has only one category. Both participations are total, since all events and categories are contained in this relationship.

Event – Photo

This relationship (event includes photos) has a one-to-many relationship, since events will have multiple photos and each photo will be related to a single event. Both entities have total participation in this relationship, since all events will have photos and all stored photos will be from a given event.

Event - Place

The relationship between events and places (events are located in a place) has a many-to-one cardinality, since more than one event may happen in a given location throughout the year, but each event happens in a single location. Event will have total participation (since every event must have a place) and place will have partial participation (since we may store some hiking locations in which no event happened yet).

3. Assumptions about 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 won't have any acquired 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** – each event happens in a single location, but more than one event may happen in a given 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 won't have attended an event at a given moment;
    - **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 we're 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.

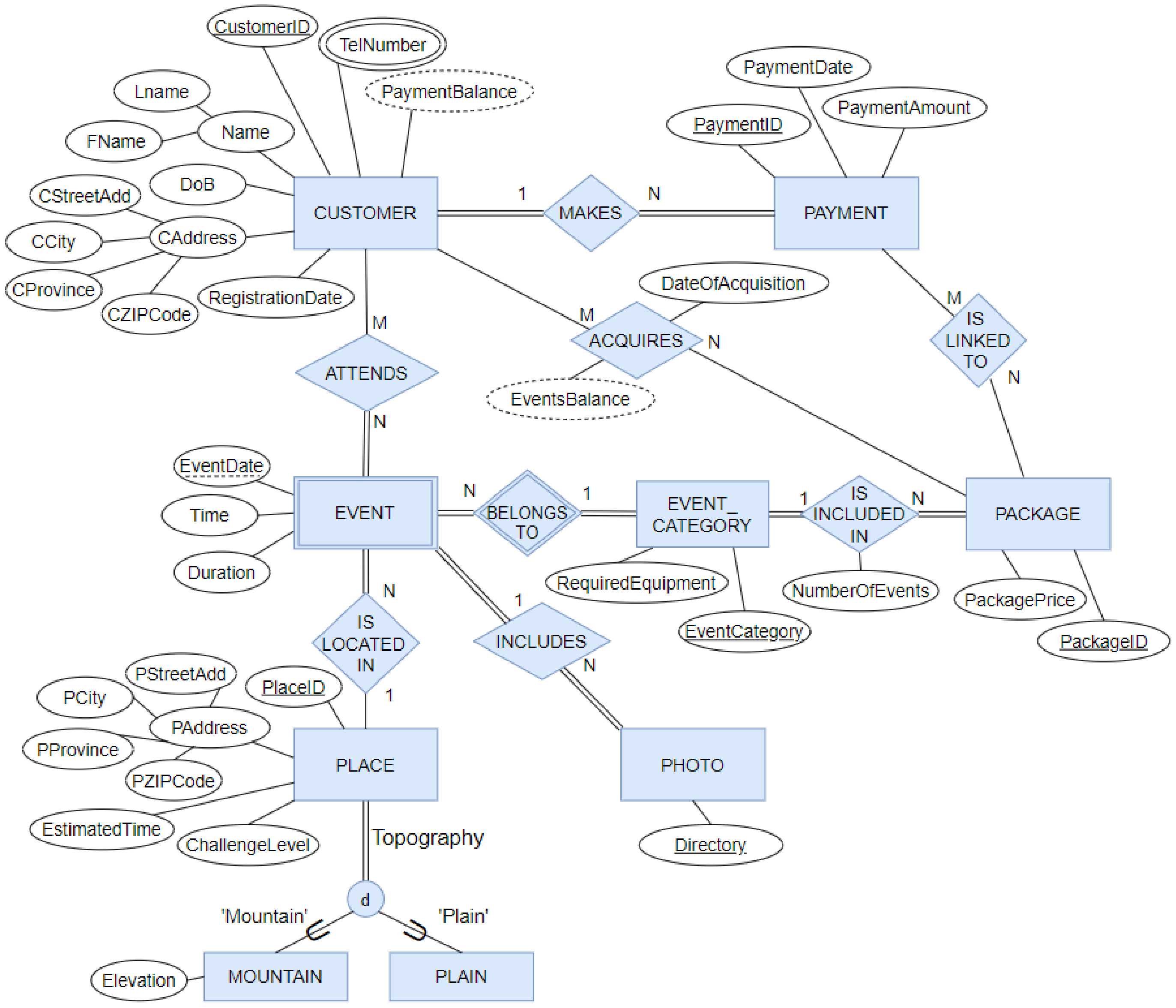
- Assumption 6 – **Event Category-Package relationship (Event Category is included in Package)**
    - **Cardinality: One-to-many** – Each package contains 1 event category, but multiple packages may refer to the same category (with different number of events);
    - **Package Participation: Total** – 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.

4. EER Modeling Diagram

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

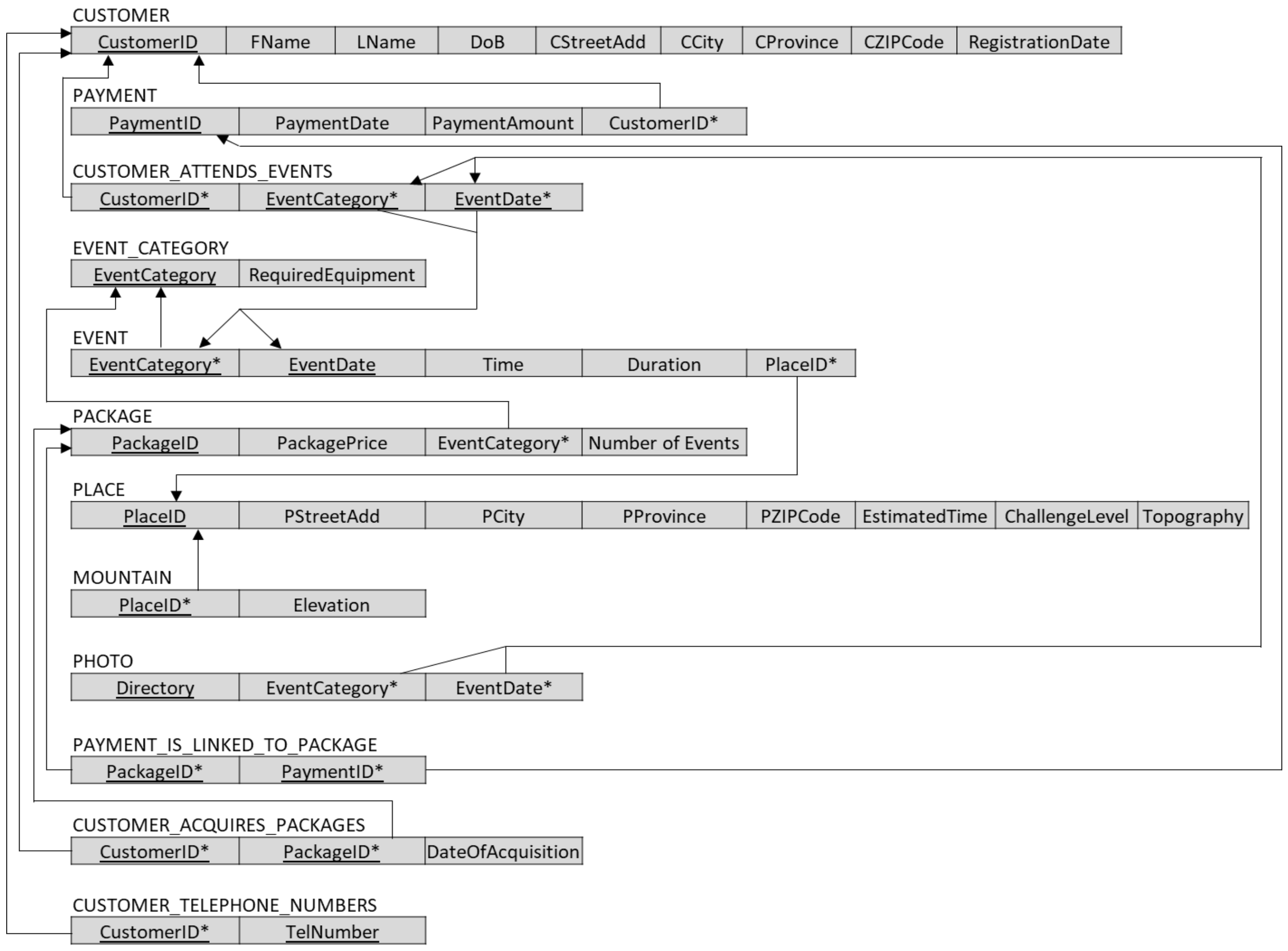


5. ER-Model Mapping to Database Relational Schema

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

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

The schema below shows the aforementioned relations, indicating the respective references of each foreign key. For the following schema, the relations are reorganized to provide a better visualization of each connection. Nevertheless, it <u>does not reflect the order in which the mapping was made</u>, which <u>can be seen in the previous list</u>.


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

6. 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 in 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.
- CUSTOMER_ACQUIRES_PACKAGES:
  - { CustomerID, PackageID } -> DateOfAcquisition
  - Both customer and package ID are required to determine the date of acquisition in this relation.
- PAYMENT_IS_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_NUMBERS:
    - { 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, since none of them present Transitive Functional Dependencies, since none of the 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.

7. 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
    - CUSTOMERID ON DELETE SET NULL: The payments must be kept in the database even when a customer no longer uses the services

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
- EVENTCATEGORY ON DELETE CASCADE: each package contains 1 event category. Thus, once the event category is deleted, the package is no longer available.

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
    - EVENTCATEGORY and EVENTDATE ON DELETE SET NULL: the pictures are kept in the database even when the events or the event categories are deleted.

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
    - For STARTINGTIME, due to the lack of support for different time formats, the best option is to use VARCHAR as the type in order to make it usable on different platforms.
    - PLACEID ON DELETE SET NULL: events must be kept in the database even when a place is no longer registered to hold hiking events
    - EVENTCATEGORY ON DELETE CASCADE: the event only exists if it belongs to an event category.

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
  - CUSTOMERID, EVENTCATEGORY, EVENTDATE ON DELETE CASCADE: they are all prime attributes so they cannot be null.
  - ON DELETE CASCADE: These records lose their meaning when the corresponding records they reference are deleted.

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
  - CUSTOMERID, PACKAGEID ON DELETE CASCADE: This record loses its meaning when the corresponding record it references is deleted.

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
  - PAYMENTID, PACKAGEID ON DELETE CASCADE: This record loses its meaning when the corresponding record it references is deleted.

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
  - CUSTOMERID ON DELETE CASCADE: This record loses its meaning when the corresponding record it references is deleted.

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
    - PLACEID ON DELETE CASCADE: This record loses its meaning when the corresponding record it references is deleted.

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

8. Creating Database and Tables - SQL DDL

File: Create_Tables.sql

9. Inserting Values in Tables

File: Populate_Tables.sql

10.SQL Queries and Views

File: SQL_Commands.sql | SQL_Commands.txt

Queries' description (in file's order):

- Calculate each customer's balance, by joining CUSTOMER_ACQUIRES_PACKAGES, PACKAGES and PAYMENT;
- Show number of events performed in each place, including place information, by joining PLACE and EVENTS;
- Show customers with a negative balance, by joining CUSTOMER_ACQUIRES_PACKAGES, PACKAGES and PAYMENT;
- Calculate each customer's event balance (events acquired – events attended), joining CUSTOMER_ATTENDS_EVENTS, CUSTOMER_ACQUIRES_PACKAGES and PACKAGES. Show customers with balance > 8;
- Show number of participants in each event;
- Show number of packages acquired by month, to identify a potential seasonality;
- Show total number of events performed on each year;
- Show photos to be sent for each customer, given their event attendance;
- Update price of 2 event categories, increasing it by 10%;
- Show events attended by each customer, separated by event category;
- Delete a type of package;
- Delete places from a given city;
- Create a view of photos from advanced level events;
- Create a view of photos from beginner level events;
- Calculate total amount payed by each customer;
- Calculate taxes to be paid, considering payments made;
- Identify customers that attended events of 2 different types (Easy Stroll and Easy Peasy Hike);
- Show number of packages acquired by each customer;
- Show customers which didn't attend any event in 2019 or 2020;
- Show number of events acquired by each customer, separating by event category;