|  |
| --- |
|  |
| Ski Tracking Application Database |
| Course Section: CS605.641.83  Spring, 2023 |
| Prepared by |
| **Christian Znidarsic** |
| **04/19/2023** |

Logo

Description automatically generated

|  |
| --- |
| Database Design Project Document |

**Table of Contents**

[1. Introduction 4](#_Toc132817658)

[1.1. Scope and Purpose of Document 4](#_Toc132817659)

[1.2. Project Objective 4](#_Toc132817660)

[2. System Requirements 4](#_Toc132817661)

[2.1 Hardware Requirements 4](#_Toc132817662)

[2.2 Software Requirements 4](#_Toc132817663)

[2.3 Functional Requirements 5](#_Toc132817664)

[2.4 Database Requirements 5](#_Toc132817665)

[3. Database Design Description 6](#_Toc132817666)

[3.1 Design Rationale 6](#_Toc132817667)

[3.2 E/R Model 7](#_Toc132817668)

[3.2.1 Entities 7](#_Toc132817669)

[3.2.2 Relationships 8](#_Toc132817670)

[3.2.3 E/R Diagram 10](#_Toc132817671)

[3.3 Relational Model 10](#_Toc132817672)

[3.3.1 Data Dictionary 10](#_Toc132817673)

[3.3.2 Integrity Rules 14](#_Toc132817674)

[3.3.3 Operational Rules 14](#_Toc132817675)

[3.3.4 Operations 15](#_Toc132817676)

[3.4 Security 15](#_Toc132817677)

[3.5 Database Backup and Recovery 16](#_Toc132817678)

[3.6 Using Database Design or CASE Tool 16](#_Toc132817679)

[3.7 Other Possible E/R Relationships 16](#_Toc132817680)

[4. Implementation Description 17](#_Toc132817681)

[4.1 Data Dictionary 17](#_Toc132817682)

[4.2 Advanced Features 19](#_Toc132817683)

[4.3 Queries 23](#_Toc132817684)

[4.3.1 All runs skied by skier in specified season at specified resort 23](#_Toc132817685)

[4.3.2 All runs skied by skier on specified day at specified resort 23](#_Toc132817686)

[4.3.3 All chairlifts taken by skier on specified day at specified resort 23](#_Toc132817687)

[4.3.4 Number of trails with specified rating skied by skier in specified season at specified resort 24](#_Toc132817688)

[4.3.5 Skier information 24](#_Toc132817689)

[4.3.6 All awards completed by skier in specified season at specified resort 24](#_Toc132817690)

[4.3.7 Status of all chairs at specified resort 24](#_Toc132817691)

[4.3.8 Resort information for given resort 25](#_Toc132817692)

[5. CRUD Matrix 25](#_Toc132817693)

[5.1 List of Entity Types 25](#_Toc132817694)

[5.2 List of Functions 26](#_Toc132817695)

[6. Concluding Remarks 27](#_Toc132817696)

[References 45](#_Toc132817697)

# Introduction

Explain your database project at high-level and/or why you choose this database topic.

## Scope and Purpose of Document

This document is meant to discuss the requirements, design, and implementation of the database. The purpose of this document is to serve as a written record to demonstrate the thinking process regarding the conceptual design, logical design, and implementation of the database, and to summarize the features that have been implemented.

The first section, System Requirements will discuss the requirements for implementing the database. In the next section, Database Design Description, we will go over a detailed desciption of the design of the database, including the design rational, an entity relationship model, and other aspects of the design. The next section is Implementation Description, which details how the database was implemented and the various triggers, functions and stored procedures that were added to it. Finally, this document contains a Crud Matrix section as well as Concluding Remarks.

## Project Objective

The objective of this project is to design and implement a database using a RDBMS that captures all informational aspects of the operations required by a mobile application owned by a fictional business entity operating in the ski industry. This mobile application is a ski tracking app that uses GPS to track and log various activities on a mountain. These activities include skiing trails and riding chairlifts. As a skier performs more of these actions, they earn awards associated with the runs they have skied. The mobile application is assumed to write to the database automatically, and it's design and implementation are outside the scope of this project.

# System Requirements

## Hardware Requirements

The minimum hardware requirements for running MySQL with MySQL Workbench are:

Table

Description automatically generated with medium confidence

## Software Requirements

To implement and run this database, MySQL is required and MySQL Workbench is suggested. The current version of MySQL Workbench is 8.0 and is recommended for use with MySQL 8.0. It also works with MySQL 5.7.

The operating system requirements for using MySQL with MySQL Workbench are detailed below:

Graphical user interface, text, application

Description automatically generated

## Functional Requirements

The functional requirements for the database are as follows:

* The database must be capable of holding new skier information when a new account is created
* The database must be able to store data for every run a skier skis
* The database must be able to store data for every chairlift a skier rides
* The database must be able to store data for various awards a skier may earn
* The ski data must support queries by skier, resort and season
* The ski data must support queries by skier, resort and day
* The ride data must support queries by skier, resort and day
* The award data must support queries by skier, resort and season
* The ski data must support queries to get total number of trails skied by skier, resort and rating
* The earn data must support queries to get total number of awards earned by skier, resort and season
* The ride data must support queries to get total number of chairlifts ridden by skier, resort and season
* The skier data must support queries by skier
* The resort data must support queries by resort
* The chairlift data must support updates to chair status, as well as reads for chair status
* The award data must support the creation and update of awards

## Database Requirements

This project uses MySQL database version 8.0.31.

# Database Design Description

## Design Rationale

The database schema design is centered primarily around the SKIER table. The SKIER table participates in three many-to-many relationships with TRAIL, CHAIRLIFT and AWARD.

The first is with TRAIL. A skier can ski zero or many trails, and a trail can be skied by zero or many skiers, so a SKI table is made to track every time a skier skis a trail. The SKI table contains two foreign keys, Trail\_Id and Skier\_Id. This way, each entry into SKI is associated with a skier and a trail.The second many-to-many relationship that SKIER participates in is with CHAIRLIFT. A skier may ride zero or many chairlifts, and a chairlift may be ridden by zero or many skiers. The third many-to-many relationship is with AWARD. A skier may make progress towards zero or many awards, and an award may be earned by zero or many skiers. This relationship results in an EARN table that keeps track of every award a given skier is making progress towards or has already earned.

These relationships are all non-defining. The SKI, EARN and RIDE tables all have UIDs. A skier may participate in zero or many of these relationships, and each instance of these relationships may only be associated with a single skier.

The RESORT table stores information about each resort. This table also references the CITY lookup table, which in turn references the STATE lookup table. This allows city codes to be used in the RESORT table rather than the actual city names. Thus, if any city name is changed, it must only be changed once in the CITY table, and not in all of the RESORT tuples.

The reference table CHAIR\_STATUS is also used by CHAIRLIFT to avoid storing many copies of the chair status descriptions in CHAIRLIFT.

All tables use artificial primary keys to differentiate between tuples in the table. Even tables such as RIDE, EARN and SKI use artificial primary keys. Using artificial primary keys in these tables avoids the use of composite primary keys, which in turn improves performance.

No relationships of the type (1,1) ---------- (1,m) exist in the schema. This relationship may sound like a reasonable business requirement, but it cannot be implemented in a relational database.

All relationships in the schema are non-identifying relationships. Identifying relationships were purposely avoided. In place of identifying relationships, non-identifying relationships were used with artificial primary keys on the SKI, EARN and RIDE tables so that these tables wouldn't rely on composity primary keys. Avoiding composite primary keys can help improve performance.

## E/R Model

### Entities

**AWARD**

The AWARD entity contains various awards associated with a given resort. Each resort may have different awards that are specific to that resort. Examples of awards include skiing 30 days in a season, or skiing all lifts in a resort. Each award has an Award\_Id, a name, a description, a number\_needed attribute which may or may not be applicable for a given award, and a Resort\_Id to indicate which resort it is associated with. Some awards may not be quantitative, and thus would not require a Number\_Needed value.

**CHAIRLIFT**

The CHAIRLIFT table contains all chairlifts at all resorts supported by the ski tracker. Each chairlift tuple has a name, Status\_Id, and a Resort\_Id.

**CHAIR\_STATUS**

The CHAIR\_STATUS table serves as a reference table for the CHAIRLIFT table. This table contains the various chairlift status messages, such as "Hold" or "Open" to identify the current state of each chairlift.

**CITY**

The CITY table is used as a reference table for the RESORT table. It contains the names of all cities with resorts in them, and their corresponding City\_Id values as well as State\_Id values.

**EARN**

The EARN table contains all awards currently being earned or that have been earned by a skier. Each award has an Earn\_Id as a unique identifier, as well as two foreign keys Skier\_Id and Award\_Id. The Progress attribute contains the progress value that a skier has achieved. For example, if a skier has ridden 5 out of the 10 chairlifts at a resort, then the Progress attribute would hold the value 5.

**RESORT**

The resort table contains tuples for each resort that is covered by the ski tracking app. Each resort has a Resort\_Id as a primary key, as well as a name and a City\_Id.

**RIDE**

The ride table contains entries for each time a skier rides a chairlift. The primary key is the Ride\_Id. The Skier\_Id and Chairlift\_Id are foreign keys. The RIDE table also has a Date\_Time value to identify when the ride took place.

**SEASON**

There is a tuple in the SEASON table for each season. Because different resorts have different opening and closing dates, it is important to track when each resort opens and closes for the year. This data helps when querying for season statistics for a given skier. Each tuple contains a Season\_Id, a Name, a Start\_Date, End\_Date and a Resort\_Id. The Name attribute is particularly important when querying season statistics. Each name is in the format "YYYY-YYYY". For example, a season may be named "2022-2023" so that when querying we can specifically request results from the 2022-2023 season.

**SKI**

The SKI table does a lot of the heavy lifting in this schema. Every time a skier skis a trail, a new entry is made in the SKI table to record the event. It logs the Skier\_Id, the Trail\_Id, and the Date and Time as Date\_Time that it was skied on. A Ski\_Id attribute serves as a UID for the table.

**SKIER**

The SKIER table contains data for all skiers who have registered using the app. Each skier tuple has a unique Skier\_Id, a Fname and Lname, an Email and a Password. The password is encrypted in the database.

**STATE**

The STATE table serves as a lookup table for the CITY table. It contains State\_Ids and their corresponding Name values.

**TRAIL**

The TRAIL table contains tuples for all trails across all resorts covered by the ski tracking app. Each trail tuple has a Trail\_Id as the primary key, a Name, a Rating (with domain "Green", "Blue", "Black", "Double Black"), a Length in miles, and a Resort\_Id indicating where the resort is located.

### Relationships

**AWARD (1,1) ---------- (0,m) EARN**

This relationship summarizes how each award can be earned by zero or many skiers, but each skier can only earn a single instance of a given award.

**CHAIRLIFT (1,1) ---------- (0,m) RIDE**

Each chairlift can be ridden by zero or many skiers, and thus can have zero or many RIDE instances associated with it. However, each RIDE instance can only be associated with a single chairlift.

**CHAIR\_STATUS (1,1) ---------- (0,m) CHAIRLIFT**

Each CHAIR\_STATUS instance can apply to zero or many chairlifts. On the other hand, each chairlift instance can only have a single status associated with it at any given time.

**CITY (1,1) ---------- (0,m) RESORT**

Each instance of CITY can have zero or many resorts in it. Each RESORT instance can only be located in a single city.

**RESORT (1,1) ---------- (0,m) AWARD**

Each ski RESORT can have zero or many AWARD instances associated with it. These awards may be specific to the resort. Each AWARD however can only belong to a single resort.

**RESORT (1,1) ---------- (0,m) CHAIRLIFT**

Every RESORT can have zero or many chairlifts in it. Each CHAIRLIFT instance can only be located in one resort.

**RESORT (1,1) ---------- (0,m) SEASON**

Each resort can have zero or many seasons associated with it. In the case of a brand new resort, it wouldn't yet have had any seasons. Every SEASON instance belongs to one and only one resort, because different resorts open and close at different times.

**RESORT (1,1) ---------- (0,m) TRAIL**

Every RESORT can have zero or many trails in it. Each TRAIL instance can only be located in a single resort.

**SKIER (1,1) ---------- (0,m) EARN**

Every SKIER can earn or be in the process of earning zero or many awards, and thus may have zero or many EARN instances associated with them. On the other hand, each instance of EARN is associated with only a single SKIER instance.

**SKIER (1,1) ---------- (0,m) RIDE**

Every SKIER can ride zero or many chairlifts, and thus may be associated with zero or many RIDE instances. Each RIDE instance is associated with only a single SKIER instance.

**SKIER (1,1) ---------- (0,m) SKI**

Each SKIER can ski zero or many trails, and thus may be associated with zero or many SKI instances. On the other hand, each trail that is skied results in a SKI instance that is associated with only a single skier.

**STATE (1,1) ---------- (0,m) CITY**

A given STATE may contain zero or many cities. Each CITY can only be located in a single state.

**TRAIL (1,1) ---------- (0,m) SKI**

Each TRAIL can be skied zero or many times by skiers. However, each SKI instance must refer to one and only one trail.

### E/R Diagram

Chart, box and whisker chart

Description automatically generated

## Relational Model

### Data Dictionary

**AWARD**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| Award\_Id | Primary Key of AWARD table | INT | 4 | Primary Key | Y | 1 to 4294967295 |
| Name | Name of Award | VARCHAR(45) | 45 | None | Y | 45 chars |
| Description | Description of an award | VARCHAR(45) | 45 | None | Y | 45 chars |
| Number\_Needed | The number of a given event needed to earn the award | INT | 4 | None | N | 1 to 4294967295 |
| RESORT\_Resort\_Id | Resort Id from RESORT table | INT | 4 | Foreign Key | Y | 1 to 4294967295 |

**CHAIRLIFT**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| Chairlift\_Id | Primary Key of CHAIRLIFT table | INT | 4 | Primary Key | Y | 1 to 4294967295 |
| Name | Name of Chairlift | VARCHAR(45) | 45 | None | Y | 45 chars |
| CHAIR\_STATUS\_Status\_Id | Status Id from STATUS table | VARCHAR(45) | 45 | Foreign Key | Y | 45 chars |
| RESORT\_Resort\_Id | Resort Id from RESORT table | VARCHAR(45) | 45 | Foreign Key | Y | 45 chars |

**CHAIR\_STATUS**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| Status\_Id | Primary Key for CHAIR\_STATUS table | INT | 4 | Primary Key | Y | 1 to 4294967295 |
| Description | A description of a chair status | VARCHAR(45) | 45 | None | Y | 45 chars |

**CITY**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| City\_Id | Primary Key for CITY table | INT | 4 | Primary Key | Y | 1 to 4294967295 |
| Name | Name of City | VARCHAR(45) | 45 | None | Y | 45 chars |
| STATE\_State\_Id | State Id from STATE table | VARCHAR(2) | 2 | Foreign Key | Y | 2 chars |

**EARN**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| Earn\_Id | Primary Key of EARN table | INT | 4 | Primary Key | Y | 1 to 4294967295 |
| Progress | Integer representing amount of progress towards award | INT | 4 | None | N | 1 to 4294967295 |
| Earned | Boolean representing if award was earned or not | TINYINT | 1 | None | Y | 0 or 1 |
| Date\_Earned | Date when award was earned. Null if not earned | DATE | 3 | None | N | 'YYYY-MM-DD' |
| SKIER\_Skier\_Id | Skier Id from SKIER table | INT | 4 | Foreign Key | Y | 1 to 4294967295 |
| AWARD\_Award\_Id | Award Id from AWARD table | INT | 4 | Foreign Key | Y | 1 to 4294967295 |

**RESORT**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| Resort\_Id | Primary Key of RESORT table | INT | 4 | Primary Key | Y | 1 to 4294967295 |
| Name | Name of Resort | VARCHAR(45) | 45 | None | N | 45 chars |
| CITY\_City\_Id | City Id from CITY table | INT | 4 | None | N | 1 to 4294967295 |

**RIDE**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| Ride\_Id | Primary Key of RIDE table | INT | 4 | Primary Key | Y | 1 to 4294967295 |
| Date\_Time | The date and time the ride took place | DATETIME | 8 | None | Y | 'YYYY-MM-DD hh:mm:ss' |
| SKIER\_Skier\_Id | Skier Id from SKIER table | INT | 4 | Foreign Key | Y | 1 to 4294967295 |
| CHAIRLIFT\_Chairlift\_Id | Chairlift Id from CHAIRLIFT table | INT | 4 | Foreign Key | Y | 1 to 4294967295 |

**SEASON**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| Season\_Id | Primary Key of SEASON table | INT | 4 | Primary Key | Y | 1 to 4294967295 |
| Name | Name of Season | VARCHAR(45) | 45 | None | Y | 45 chars |
| Start\_Date | Starting date of season | DATE | 3 | None | Y | 'YYYY-MM-DD' |
| End\_Date | Ending date of season | DATE | 3 | None | Y | 'YYYY-MM-DD' |

**SKI**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| Ski\_Id | Primary Key of the SKI table | INT | 4 | Primary Key | Y | 1 to 4294967295 |
| Date\_Time | The date and time the ski took place | DATETIME | 8 | None | Y | 'YYYY-MM-DD hh:mm:ss' |
| SKIER\_Skier\_Id | Skier Id from SKIER table | INT | 4 | Foreign Key | Y | 1 to 4294967295 |
| TRAIL\_Trail\_Id | Trail Id from the TRAIL table | INT | 4 | Foreign Key | Y | 1 to 4294967295 |

**SKIER**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| Skier\_Id | Primary Key of SKIER table | INT | 4 | Primary Key | Y | 1 to 4294967295 |
| Fname | First name of the skier | VARCHAR(45) | 45 | None | Y | 45 chars |
| Lname | Last name of the skier | VARCHAR(45) | 45 | None | Y | 45 chars |
| Email | Email of the Skier | VARCHAR(45) | 45 | None | Y | 45 chars |
| Password | Password of the Skier | BINARY(45) | 45 | None | Y | 45 binary chars |

**STATE**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| State\_Id | Primary Key for STATE table | VARCHAR(2) | 2 | Primary Key | Y | 2 chars |
| Name | Name of State | VARCHAR(45) | 45 | None | Y | 45 chars |

**TRAIL**

| Column Name | Description | Data Type | Size (bytes) | Constraint Type | Not Null? | Valid Values |
| --- | --- | --- | --- | --- | --- | --- |
| Trail\_Id | Primary Key of TRAIL table | INT | 4 | Primary Key | Y | 1 to 4294967295 |
| Name | Name of Trail | VARCHAR(45) | 45 | None | Y | 45 chars |
| Rating | Rating of Trail | VARCHAR(45) | 45 | None | Y | 'Green', 'Blue', 'Black', or 'Double Black' |
| Length | Length of Trail in miles | FLOAT(4,1) | 4 | None | Y | 000.0 to 999.9 |
| RESORT\_Resort\_Id | Resort Id from RESORT table | INT | 4 | Foreign Key | Y | 1 to 4294967295 |

### Integrity Rules

Mandatory fields were handled using NOT NULL constraints in the CREATE statements when creating the database tables. These constraints ensure that no data can be inserted into the table without a value for all attributes marked NOT NULL.

Valid values for attributes are ensured using triggers which are set to run before any insert is performed on a given table. The main trigger of this type is set on the TRAIL table. It only allows insertion into the table if the Rating value is within the set domain of the attribute. This domain is 'Green', 'Blue', 'Black', and 'Double Black'. MySQL also ensures that all attributes adhere to the specified data type before insertion into a table.

Referential integrity constraints are also set using FOREIGN KEY constraint keywords in the CREATE statements when creating the database tables. These constraints ensure referential integrity across associated tables. Namely, they ensure that any referenced data does not get deleted in the source table. Many foreign key constraints exist in the database.

### Operational Rules

Operational rules are dictated by relationships between tables, foreign key constraints and triggers.

The cardinalities of the relationships between tables determine the number of foreign key values can exist for a given primary key. Most all relationships in the schema are (1,1) --> (1,M) relationships. For example, one or many cities can be associated with the state primary key 'CA' because a (1,1) --> (1,M) relationship exists from STATE to CITY. This relationship is defined in the DDL statements using the FOREIGN KEY keyword, which establishes the STATE primary key State\_Id as a foreign key in the CITY table. This constraint allows many duplicate State\_Id values to exist in the CITY table. Also, if the State\_Id 'CA' is referenced by any CITY tuples, the corresponding 'CA' STATE tuple cannot be deleted. This is another constraint created by the FOREIGN KEY statement.

Triggers are also used to put constraints on certain operations. For example, we cannot insert a TRAIL record if the Rating value is not one of the four values listed above. Another trigger that we have implemented ensures that no skier instance can be created using an email that already exists in the database. The first and last names of the new skier are not considered, since multiple people can have the same first and last names. Only the email is considered when preventing duplicate skier accounts from being created.

The third trigger that enforces an operational constraint is placed on the EARN table. This trigger ensures that no duplicate tuples are inserted into the EARN table. In other words, no EARN tuple can be inserted with the same Skier\_Id and Award\_Id as a tuple that already exists in the table. These are examples of operational constraints that we have put on our database.

### Operations

Operations that will be performed on the database include insert, delete, update and retrieve operations.

Insert operations will primarily be performed on the SKI, RIDE and EARN tables. These tables will be written to whenever one of these events occur and needs to be logged in the database. Records will also be inserted into the SKIER table whenever a new skier creates an account. The SEASON table will be written to once for each resort each new season. Tables such as RESORT, CHAIRLIFT, TRAIL and AWARD will also have insertions performed on them, but there are expected to be many fewer insertions on these tables than there will be on tables such as SKI, as trails and chairlifts are not created as often. Tables such as CHAIR\_STATUS, CITY, STATE are rarely expected to be written to, as these ta

Delete operations are only expected to be performed in uncommon scenarios. For most of our tables, the intention is for data to be saved for historical purposes, and thus would not benefit from being deleted.

Update operations will be performed on the CHAIRLIFT table to update the chairlift status. These updates will occur periodically throughout the day as the chair status changes. Also, it is expected that Resorts, Trails and Chairlifts will change names from time to time, so we expect to perform updates on these tables as well. The SKIER table will also be updated in the event that a user changes their personal information such as their password.

Retrieve operations will be performed when a skier wishes to see their historical ski data. These operations are expected to occur often, as they comprise the majority of the application functionality. In particular, we expect to retrieve often from the SKI, RIDE and EARN tables. We will also retrieve from tables such as SEASON, AWARD, TRAIL, and CHAIRLIFT using join operations.

## Security

Important security measures include:

**User Access**

The database is hosted locally on my machine at port 3306. Access for user 'root' is password protected to prevent unwanted access to the database.

**Sensitive Data**

Sensitive user data is encrypted for security purposes. In our case, the only sensitive data is user password information. We encrypt this data using AES\_ENCRYPT in MySQL. To accommodate encrypted information, the Password attribute of the SKIER table is of data type BINARY(45).

**SQL Injection**

SQL injection will be handled at the application level. All queries made to the database must be constructed in ways that do not allow SQL injection. For example, direct string concatenation must be avoided. Instead, queries should be constructed using query building tools that do not allow SQL injection, such as the Hibernate Criteria API in the case of a Java Enterprise Edition application.

## Database Backup and Recovery

MySQL supports numerous database backup and recovery options. For those who subscribe to MySQL Enterprise Edition, there is a MySQL Enterprise Backup product that does physical backups of your databases. For non-subscribers, there are still options. One of these options is performing incremental backups using the binary log. Binary log files can be created at intervals. They contain all the information you need in order to replicate any changes to the database that were made before the backup. For the purposes of this project we have not implemented backup and recovery for the database.

## Using Database Design or CASE Tool

The majority of this project was created using MySQL Workbench. MySQL Workbench was used to implement the database, as well as to create the ERD.

We created the ERD in Workbench using the EERD tool, then converted it into a database using the Forward Engineer tool. After converting our ERD to a database, we made further changes manually such as adding triggers and stored procedures.

We also used Excel for creating sample data. Sample data for all tables aside from the RIDE, SKI, EARN and STATE tables were first made in excel and then imported to our database as csv files.

## Other Possible E/R Relationships

**STATE (1,1) ---------- (0,m) RESORT**

Initially, the database was designed with a relationship directly between STATE and RESORT. However, it was decided that because CITY is dependent on STATE, a relationship should exist between CITY and STATE instead.

**SEASON (1,1) ---------- (0,m) SKI**

Initially the database was designed with a relationship directly between SEASON and SKI. This meant that the SKI table contained a foreign key column called SEASON\_Season\_Id. However, when performing database normalization, specifically when converting to Third Normal Form, it was decided that the SEASON\_Season\_Id attribute was functionally dependent upon the Date\_Time attribute in the SKI table. Thus, a transitive dependency existed in the SKI table. To remove this transitive dependency and convert to Third Normal Form, the relationship between SEASON and SKI was removed. Instead, to query SKI instances that occurred in a given season, we must directly reference the SEASON table and use a BETWEEN operation to choose only SKI instances that occurred between the Start\_Date and End\_Date of the SEASON instance. This can be seen in the section containing queries for the database.

# Implementation Description

## Data Dictionary

**AWARD**

**Table

Description automatically generated**

**CHAIRLIFT**

**Table

Description automatically generated**

**CHAIR\_STATUS**

**Table

Description automatically generated**

**CITY**

**Table

Description automatically generated**

**EARN**

**Table

Description automatically generated**

**RESORT**

**Table

Description automatically generated**

**RIDE**

**Table

Description automatically generated**

**SEASON**

**Table

Description automatically generated**

**SKI**

**Table

Description automatically generated**

**SKIER**

**Table

Description automatically generated**

**STATE**

**Table

Description automatically generated**

**TRAIL**

**Table

Description automatically generated**

## Advanced Features

**Triggers**

**1. before\_trail\_insert**

The before\_trail\_insert trigger is used to control the domain of the 'Rating' column. The domain can only include 'Green', 'Blue', 'Black', and 'Double Black'. The before\_trail\_insert trigger checks to make sure that the new 'Rating' value that has been passed is in this domain. If it is not, then the new TRAIL instance is not inserted into the table, and an error message is returned describing the violation.

DELIMITER //

CREATE TRIGGER before\_trail\_insert BEFORE INSERT ON TRAIL

FOR EACH ROW

BEGIN

IF NEW.Rating != 'Green' AND NEW.Rating != 'Blue' AND NEW.Rating != 'Black' AND NEW.Rating != 'Double Black'

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot add or update row: only Green, Blue, Black and Double Black are allowed as rating values';

END IF;

END;

**2. before\_skier\_insert**

The before\_skier\_insert trigger is used every time a new skier is created. It checks to see if the provided email already exists in the database. This trigger ensures that there are no duplicate accounts made, and that each skier can only make a single account using their email.

DELIMITER //

CREATE TRIGGER before\_skier\_insert BEFORE INSERT ON SKIER

FOR EACH ROW

BEGIN

IF (EXISTS (SELECT \* FROM SKIER WHERE Email = NEW.Email))

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot add or update row: user email already exists';

END IF;

END;

**3. before\_earn\_insert**

The before\_earn\_insert trigger is called before all insert operations are performed on the EARN table. It checks to see if an EARN instance already exists for the provided Skier\_Id and Award\_Id. If an instance already exists, then the insert operation is declined and an error message is returned describing why.

DELIMITER //

CREATE TRIGGER before\_earn\_insert BEFORE INSERT ON EARN

FOR EACH ROW

BEGIN

IF (EXISTS (SELECT \* FROM EARN WHERE SKIER\_Skier\_Id = NEW.SKIER\_Skier\_Id AND AWARD\_Award\_Id = NEW.AWARD\_Award\_Id))

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot add or update row: \'earn\' instance is already created for given skier and award';

END IF;

END;

**Stored Procedures**

**1. add\_ski\_instance (Date\_Time, Skier\_Id, Trail\_Id)**

The add\_ski\_instance stored procedure is used for creating new SKI instances. This stored procedure is meant to be called whenever a skier skis a run and it is entered into the database. The stored procedure makes this action more simple to perform rather than writing a full INSERT operation.

DELIMITER //

CREATE PROCEDURE add\_ski\_instance (\_Date\_Time DATETIME, \_Skier\_Id INT, \_Trail\_Id INT)

BEGIN

INSERT INTO SKI (Date\_Time, SKIER\_Skier\_Id, TRAIL\_Trail\_Id)

VALUES (\_Date\_Time, \_Skier\_Id, \_Trail\_Id);

END ;

**2. add\_ride\_instance (Date\_Time, Skier\_Id, Trail\_Id)**

The add\_ride\_instance stored procedure is used whenever a skier rides a chairlift. This stored procedure is meant to make the action of documenting ride instances more simple, and will be called by the client application.

DELIMITER //

CREATE PROCEDURE add\_ride\_instance (\_Date\_Time DATETIME, \_Skier\_Id INT, \_Chairlift\_Id INT)

BEGIN

INSERT INTO RIDE (Date\_Time, SKIER\_Skier\_Id, CHAIRLIFT\_Chairlift\_Id)

VALUES (\_Date\_Time, \_Skier\_Id, \_Chairlift\_Id);

END ;

**3. create\_new\_skier (Fname, Lname, Email, Password)**

The create\_new\_skier stored procedure is used any time the client application wants to add a new skier record to the database. This makes the process of adding more users easier to call from the client application.

DELIMITER //

CREATE PROCEDURE create\_new\_skier (\_Fname VARCHAR(45), \_Lname VARCHAR(45), \_Email VARCHAR(45), \_Password BINARY(45))

BEGIN

INSERT INTO SKIER (Fname, Lname, Email, Password)

VALUES (\_Fname, \_Lname, \_Email, \_Password);

END ;

**Functions**

**1. get\_num\_runs (skierId INT, resortId INT, seasonName)**

The get\_num\_runs function is used to retrieve the number of runs a given skier has skied at a particular resort in a specified season. The number is returned as an INT.

DELIMITER //

CREATE FUNCTION get\_num\_runs (skierId INT, resortId INT, seasonName VARCHAR(45))

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE numRuns INT;

SELECT COUNT(\*) INTO numRuns FROM (

SELECT \* FROM SKI

WHERE SKIER\_Skier\_Id = skierId AND Date\_Time BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND TRAIL\_Trail\_Id IN (SELECT Trail\_Id FROM TRAIL WHERE RESORT\_Resort\_Id = resortId)

) skiInstances;

RETURN numRuns;

END //

**2. get\_num\_lifts (skierId INT, resortId INT, seasonName VARCHAR(45))**

The get\_num\_lifts function is used to retrieve the number of lifts a given skier has ridden at a particular resort in a specified season. The number is returned as an INT.

DELIMITER //

CREATE FUNCTION get\_num\_lifts (skierId INT, resortId INT, seasonName VARCHAR(45))

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE numLifts INT;

SELECT COUNT(\*) INTO numLifts FROM (

SELECT \* FROM RIDE

WHERE SKIER\_Skier\_Id = skierId AND Date\_Time BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND CHAIRLIFT\_Chairlift\_Id IN (SELECT Chairlift\_Id FROM CHAIRLIFT WHERE RESORT\_Resort\_Id = resortId)

) chairInstances;

RETURN numLifts;

END //

**3. get\_num\_awards (skierId INT, resortId INT, seasonName VARCHAR(45))**

The get\_num\_awards function is used to retrieve the number of awards a given skier has earned at a particular resort in a specified season. The number is returned as an INT.

DELIMITER //

CREATE FUNCTION get\_num\_awards (skierId INT, resortId INT, seasonName VARCHAR(45))

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE numAwards INT;

SELECT COUNT(\*) INTO numAwards FROM (

SELECT \* FROM EARN

WHERE SKIER\_Skier\_Id = skierId AND Date\_Earned BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND AWARD\_Award\_Id IN (SELECT Award\_Id FROM AWARD WHERE RESORT\_Resort\_Id = resortId)

) awardInstances;

RETURN numAwards;

END //

## Queries

The queries are shown here as procedures to aid in readability using variables. The procedures were created in the database, but are not included in the above section (4.2. Advanced Features) in order to avoid redundancy.

### All runs skied by skier in specified season at specified resort

DELIMITER //

CREATE PROCEDURE get\_ski\_by\_resort\_season (resortId INT, skierId INT, seasonName VARCHAR(45))

BEGIN

SELECT TRAIL.Name, SKI.Date\_Time FROM SKI JOIN TRAIL ON SKI.TRAIL\_Trail\_Id = TRAIL.Trail\_Id

WHERE SKI.Date\_Time BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

SKI.SKIER\_Skier\_Id = skierId

AND

SKI.TRAIL\_Trail\_Id IN (SELECT Trail\_Id FROM TRAIL WHERE RESORT\_Resort\_Id = resortId);

END ;

### All runs skied by skier on specified day at specified resort

DELIMITER //

CREATE PROCEDURE get\_ski\_by\_resort\_day (resortId INT, skierId INT, date DATE)

BEGIN

SELECT TRAIL.Name, SKI.Date\_Time FROM SKI JOIN TRAIL ON SKI.TRAIL\_Trail\_Id = TRAIL.Trail\_Id

WHERE SKI.Date\_Time = date

AND

SKI.SKIER\_Skier\_Id = skierId

AND

SKI.TRAIL\_Trail\_Id IN (SELECT Trail\_Id FROM TRAIL WHERE RESORT\_Resort\_Id = resortId);

END ;

### All chairlifts taken by skier on specified day at specified resort

DELIMITER //

CREATE PROCEDURE get\_ride\_by\_resort\_day (resortId INT, skierId INT, date DATE)

BEGIN

SELECT CHAIRLIFT.Name, RIDE.Date\_Time FROM RIDE JOIN CHAIRLIFT ON RIDE.CHAIRLIFT\_Chairlift\_Id = CHAIRLIFT.Chairlift\_Id

WHERE RIDE.Date\_Time = date

AND

RIDE.SKIER\_Skier\_Id = skierId

AND

RIDE.CHAIRLIFT\_Chairlift\_Id IN (SELECT Chairlift\_Id FROM CHAIRLIFT WHERE RESORT\_Resort\_Id = resortId);

END ;

### Number of trails with specified rating skied by skier in specified season at specified resort

DELIMITER //

CREATE PROCEDURE get\_runs\_by\_rating\_and\_season (resortId INT, skierId INT, rating VARCHAR(45), seasonName VARCHAR(45))

BEGIN

SELECT COUNT(\*) FROM (

SELECT \* FROM SKI JOIN TRAIL ON SKI.TRAIL\_Trail\_Id = TRAIL.Trail\_Id

WHERE SKI.Date\_Time BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

SKI.SKIER\_Skier\_Id = skierId

AND

SKI.TRAIL\_Trail\_Id IN (SELECT Trail\_Id FROM TRAIL WHERE RESORT\_Resort\_Id = resortId)

AND

TRAIL.Rating = rating) greensSkied;

END ;

### Skier information

DELIMITER //

CREATE PROCEDURE get\_skier\_information (skierId INT)

BEGIN

SELECT Fname, Lname, Email

FROM SKIER

WHERE Skier\_Id = skierId;

END ;

### All awards completed by skier in specified season at specified resort

DELIMITER //

CREATE PROCEDURE get\_earn\_by\_season (skierId INT, resortId INT, seasonName VARCHAR(45))

BEGIN

SELECT AWARD.Name, AWARD.Description, EARN.Date\_Earned FROM EARN JOIN AWARD ON EARN.AWARD\_Award\_Id = AWARD.Award\_Id

WHERE EARN.SKIER\_Skier\_Id = skierId AND EARN.Earned = TRUE AND EARN.Date\_Earned BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND EARN.AWARD\_Award\_Id IN (SELECT Award\_Id FROM AWARD WHERE RESORT\_Resort\_Id = resortId);

END ;

### Status of all chairs at specified resort

DELIMITER //

CREATE PROCEDURE get\_chair\_status\_by\_resort (resortId INT)

BEGIN

SELECT CHAIRLIFT.Name, CHAIR\_STATUS.Description

FROM CHAIRLIFT JOIN CHAIR\_STATUS ON CHAIRLIFT.CHAIR\_STATUS\_Status\_Id = CHAIR\_STATUS.Status\_Id

WHERE RESORT\_Resort\_Id = resortId;

END ;

### Resort information for given resort

DELIMITER //

CREATE PROCEDURE get\_resort\_info (resortId INT)

BEGIN

SELECT RESORT.Name AS Resort\_Name, CITY.Name AS City\_Name, STATE.Name AS State\_Name

FROM

RESORT JOIN CITY ON RESORT.CITY\_City\_Id = CITY.City\_Id

JOIN STATE ON CITY.STATE\_State\_Id = STATE.State\_Id

WHERE RESORT.Resort\_Id = resortId;

END ;

# CRUD Matrix

Chart, scatter chart

Description automatically generated

## List of Entity Types

**E1**

AWARD

**E2**

CHAIRLIFT

**E3**

CHAIR\_STATUS

**E4**

CITY

**E5**

EARN

**E6**

RESORT

**E7**

RIDE

**E8**

SEASON

**E9**

SKI

**E10**

SKIER

**E11**

STATE

**E12**

TRAIL

## List of Functions

**F1**

The database must be capable of holding new skier information when a new account is created

**F2**

The database must be able to store data for every run a skier skis

**F3**

The database must be able to store data for every chairlift a skier rides

**F4**

The database must be able to store data for various awards a skier may earn

**F5**

The ski data must support queries by skier, resort and season

**F6**

The ski data must support queries by skier, resort and day

**F7**

The ride data must support queries by skier, resort and day

**F8**

The award data must support queries by skier, resort and season

**F9**

The ski data must support queries to get total number of trails skied by skier, resort and rating

**F10**

The earn data must support queries to get total number of awards earned by skier, resort and season

**F11**

The ride data must support queries to get total number of chairlifts ridden by skier, resort and season

**F12**

The skier data must support queries by skier

**F13**

The resort data must support queries by resort

**F14**

The chairlift data must support updates to chair status, as well as reads for chair status

**F15**

The award data must support the creation and update of awards

# Concluding Remarks

While working on this project I have learned many things about database design. For one, I learned the importance of starting with a sound design before implementing it. Once you begin creating tables, it is much more difficult to make changes to your schema design than it is when you are still in the conceptual design phase.

I also learned how to enforce functional requirements on the database. For example, writing triggers, stored procedures and functions. Triggers helped me enforce data formatting on insert operations, stored procedures helped simplify common queries, and functions helped create easy ways to extract specific data or values from the database.

Working on this project was a great way to apply what I have learned throughout this course to a real application. In particular, I learned how to apply the various normalization steps to my database conceptual design. I also was able to gain experience working with referential integrity constraints and other types of constraints like NOT NULL. I also gained experience working with various common datatypes in MySQL such as BINARY and DATETIME.

One thing I would like to add if I had more time to work on the project is more awards. Due to the complexity of implementing the automatic award generation using triggers, I decided to attribute this functionality to the application, rather than my database. It would have been interesting to work on this problem some more and maybe figure out a way to implement it reasonably in the database.Appendices

Appendix A - DDL, INSERT, SELECT Statements

**CREATE Statements**

**Creating the tables:**

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema SKI\_TRACKER\_3

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema SKI\_TRACKER\_3

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `SKI\_TRACKER\_3` DEFAULT CHARACTER SET utf8 ;

USE `SKI\_TRACKER\_3` ;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`SKIER`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`SKIER` (

`Skier\_Id` INT NOT NULL AUTO\_INCREMENT,

`Fname` VARCHAR(45) NOT NULL,

`Lname` VARCHAR(45) NOT NULL,

`Email` VARCHAR(45) NOT NULL,

`Password` BINARY(45) NOT NULL,

PRIMARY KEY (`Skier\_Id`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`STATE`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`STATE` (

`State\_Id` VARCHAR(2) NOT NULL,

`Name` VARCHAR(45) NOT NULL,

PRIMARY KEY (`State\_Id`, `Name`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`CITY`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`CITY` (

`City\_Id` INT NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(45) NULL,

`STATE\_State\_Id` VARCHAR(2) NOT NULL,

PRIMARY KEY (`City\_Id`),

INDEX `fk\_CITY\_STATE1\_idx` (`STATE\_State\_Id` ASC) VISIBLE,

CONSTRAINT `fk\_CITY\_STATE1`

FOREIGN KEY (`STATE\_State\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`STATE` (`State\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`RESORT`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`RESORT` (

`Resort\_Id` INT NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(45) NULL,

`CITY\_City\_Id` INT NULL,

PRIMARY KEY (`Resort\_Id`),

INDEX `fk\_RESORT\_CITY1\_idx` (`CITY\_City\_Id` ASC) VISIBLE,

CONSTRAINT `fk\_RESORT\_CITY1`

FOREIGN KEY (`CITY\_City\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`CITY` (`City\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`TRAIL`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`TRAIL` (

`Trail\_Id` INT NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(45) NOT NULL,

`Rating` VARCHAR(45) NOT NULL,

`Length` FLOAT(4,1) NOT NULL,

`RESORT\_Resort\_Id` INT NOT NULL,

PRIMARY KEY (`Trail\_Id`),

INDEX `fk\_TRAIL\_RESORT1\_idx` (`RESORT\_Resort\_Id` ASC) VISIBLE,

CONSTRAINT `fk\_TRAIL\_RESORT1`

FOREIGN KEY (`RESORT\_Resort\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`RESORT` (`Resort\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`SKI`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`SKI` (

`Ski\_Id` INT NOT NULL AUTO\_INCREMENT,

`Date\_Time` DATETIME NOT NULL,

`SKIER\_Skier\_Id` INT NOT NULL,

`TRAIL\_Trail\_Id` INT NOT NULL,

INDEX `fk\_SKI\_SKIER1\_idx` (`SKIER\_Skier\_Id` ASC) VISIBLE,

INDEX `fk\_SKI\_TRAIL1\_idx` (`TRAIL\_Trail\_Id` ASC) VISIBLE,

PRIMARY KEY (`Ski\_Id`),

CONSTRAINT `fk\_SKI\_SKIER1`

FOREIGN KEY (`SKIER\_Skier\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`SKIER` (`Skier\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_SKI\_TRAIL1`

FOREIGN KEY (`TRAIL\_Trail\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`TRAIL` (`Trail\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`SEASON`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`SEASON` (

`Season\_Id` INT NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(45) NOT NULL,

`Start\_Date` DATE NOT NULL,

`End\_Date` DATE NOT NULL,

`RESORT\_Resort\_Id` INT NOT NULL,

PRIMARY KEY (`Season\_Id`),

INDEX `fk\_SEASON\_RESORT1\_idx` (`RESORT\_Resort\_Id` ASC) VISIBLE,

CONSTRAINT `fk\_SEASON\_RESORT1`

FOREIGN KEY (`RESORT\_Resort\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`RESORT` (`Resort\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`AWARD`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`AWARD` (

`Award\_Id` INT NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(45) NOT NULL,

`Description` VARCHAR(45) NOT NULL,

`Number\_Needed` INT NULL,

`RESORT\_Resort\_Id` INT NOT NULL,

PRIMARY KEY (`Award\_Id`),

INDEX `fk\_AWARD\_RESORT1\_idx` (`RESORT\_Resort\_Id` ASC) VISIBLE,

CONSTRAINT `fk\_AWARD\_RESORT1`

FOREIGN KEY (`RESORT\_Resort\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`RESORT` (`Resort\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`EARN`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`EARN` (

`Earn\_Id` INT NOT NULL AUTO\_INCREMENT,

`Progress` INT NULL,

`Earned` TINYINT NOT NULL,

`Date\_Earned` DATE NULL,

`SKIER\_Skier\_Id` INT NOT NULL,

`AWARD\_Award\_Id` INT NOT NULL,

INDEX `fk\_EARN\_SKIER1\_idx` (`SKIER\_Skier\_Id` ASC) VISIBLE,

INDEX `fk\_EARN\_AWARD1\_idx` (`AWARD\_Award\_Id` ASC) VISIBLE,

PRIMARY KEY (`Earn\_Id`),

CONSTRAINT `fk\_EARN\_SKIER1`

FOREIGN KEY (`SKIER\_Skier\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`SKIER` (`Skier\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_EARN\_AWARD1`

FOREIGN KEY (`AWARD\_Award\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`AWARD` (`Award\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`CHAIR\_STATUS`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`CHAIR\_STATUS` (

`Status\_Id` INT NOT NULL AUTO\_INCREMENT,

`Description` VARCHAR(45) NOT NULL,

PRIMARY KEY (`Status\_Id`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`CHAIRLIFT`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`CHAIRLIFT` (

`Chairlift\_Id` INT NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(45) NOT NULL,

`CHAIR\_STATUS\_Status\_Id` INT NOT NULL,

`RESORT\_Resort\_Id` INT NOT NULL,

PRIMARY KEY (`Chairlift\_Id`),

INDEX `fk\_CHAIRLIFT\_CHAIR\_STATUS1\_idx` (`CHAIR\_STATUS\_Status\_Id` ASC) VISIBLE,

INDEX `fk\_CHAIRLIFT\_RESORT1\_idx` (`RESORT\_Resort\_Id` ASC) VISIBLE,

CONSTRAINT `fk\_CHAIRLIFT\_CHAIR\_STATUS1`

FOREIGN KEY (`CHAIR\_STATUS\_Status\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`CHAIR\_STATUS` (`Status\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_CHAIRLIFT\_RESORT1`

FOREIGN KEY (`RESORT\_Resort\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`RESORT` (`Resort\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `SKI\_TRACKER\_3`.`RIDE`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `SKI\_TRACKER\_3`.`RIDE` (

`Ride\_Id` INT NOT NULL AUTO\_INCREMENT,

`Date\_Time` DATETIME NOT NULL,

`SKIER\_Skier\_Id` INT NOT NULL,

`CHAIRLIFT\_Chairlift\_id` INT NOT NULL,

INDEX `fk\_RIDE\_SKIER1\_idx` (`SKIER\_Skier\_Id` ASC) VISIBLE,

INDEX `fk\_RIDE\_CHAIRLIFT1\_idx` (`CHAIRLIFT\_Chairlift\_id` ASC) VISIBLE,

PRIMARY KEY (`Ride\_Id`),

CONSTRAINT `fk\_RIDE\_SKIER1`

FOREIGN KEY (`SKIER\_Skier\_Id`)

REFERENCES `SKI\_TRACKER\_3`.`SKIER` (`Skier\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_RIDE\_CHAIRLIFT1`

FOREIGN KEY (`CHAIRLIFT\_Chairlift\_id`)

REFERENCES `SKI\_TRACKER\_3`.`CHAIRLIFT` (`Chairlift\_Id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

**Creating the Triggers:**

CREATE TRIGGER before\_trail\_insert BEFORE INSERT ON TRAIL

FOR EACH ROW

BEGIN

IF NEW.Rating != 'Green' AND NEW.Rating != 'Blue' AND NEW.Rating != 'Black' AND NEW.Rating != 'Double Black'

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot add or update row: only Green, Blue, Black and Double Black are allowed as rating values';

END IF;

END;

DELIMITER //

CREATE TRIGGER before\_skier\_insert BEFORE INSERT ON SKIER

FOR EACH ROW

BEGIN

IF (EXISTS (SELECT \* FROM SKIER WHERE Email = NEW.Email))

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot add or update row: user email already exists';

END IF;

END;

DELIMITER //

CREATE TRIGGER before\_earn\_insert BEFORE INSERT ON EARN

FOR EACH ROW

BEGIN

IF (EXISTS (SELECT \* FROM EARN WHERE SKIER\_Skier\_Id = NEW.SKIER\_Skier\_Id AND AWARD\_Award\_Id = NEW.AWARD\_Award\_Id))

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot add or update row: \'earn\' instance is already created for given skier and award';

END IF;

END;

**Creating the Stored Procedures:**

DELIMITER //

CREATE PROCEDURE add\_ski\_instance (\_Date\_Time DATETIME, \_Skier\_Id INT, \_Trail\_Id INT)

BEGIN

INSERT INTO SKI (Date\_Time, SKIER\_Skier\_Id, TRAIL\_Trail\_Id)

VALUES (\_Date\_Time, \_Skier\_Id, \_Trail\_Id);

END ;

DELIMITER //

CREATE PROCEDURE add\_ride\_instance (\_Date\_Time DATETIME, \_Skier\_Id INT, \_Chairlift\_Id INT)

BEGIN

INSERT INTO RIDE (Date\_Time, SKIER\_Skier\_Id, CHAIRLIFT\_Chairlift\_Id)

VALUES (\_Date\_Time, \_Skier\_Id, \_Chairlift\_Id);

END ;

DELIMITER //

CREATE PROCEDURE create\_new\_skier (\_Fname VARCHAR(45), \_Lname VARCHAR(45), \_Email VARCHAR(45), \_Password BINARY(45))

BEGIN

INSERT INTO SKIER (Fname, Lname, Email, Password)

VALUES (\_Fname, \_Lname, \_Email, \_Password);

END ;

**Creating the Stored Functions:**

DELIMITER //

CREATE FUNCTION get\_num\_runs (skierId INT, resortId INT, seasonName VARCHAR(45))

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE numRuns INT;

SELECT COUNT(\*) INTO numRuns FROM (

SELECT \* FROM SKI

WHERE SKIER\_Skier\_Id = skierId AND Date\_Time BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND TRAIL\_Trail\_Id IN (SELECT Trail\_Id FROM TRAIL WHERE RESORT\_Resort\_Id = resortId)

) skiInstances;

RETURN numRuns;

END //

DELIMITER //

CREATE FUNCTION get\_num\_lifts (skierId INT, resortId INT, seasonName VARCHAR(45))

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE numLifts INT;

SELECT COUNT(\*) INTO numLifts FROM (

SELECT \* FROM RIDE

WHERE SKIER\_Skier\_Id = skierId AND Date\_Time BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND CHAIRLIFT\_Chairlift\_Id IN (SELECT Chairlift\_Id FROM CHAIRLIFT WHERE RESORT\_Resort\_Id = resortId)

) chairInstances;

RETURN numLifts;

END //

DELIMITER //

CREATE FUNCTION get\_num\_awards (skierId INT, resortId INT, seasonName VARCHAR(45))

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE numAwards INT;

SELECT COUNT(\*) INTO numAwards FROM (

SELECT \* FROM EARN

WHERE SKIER\_Skier\_Id = skierId AND Date\_Earned BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND AWARD\_Award\_Id IN (SELECT Award\_Id FROM AWARD WHERE RESORT\_Resort\_Id = resortId)

) awardInstances;

RETURN numAwards;

END //

**INSERT Statements**

Most of the data insertion was performed using the Table Data Import Wizard in MySQL Workbench with CSV files exported from Excel spreadsheets. The only table data that was inserted manually was the STATE table. This operation is included below as an example of an INSERT statement. Both the Excel and CSV files have been included in the submission. In addition, screenshots of the CSV files are included below. Note how there are no Primary Keys included in the data, since all Primary Keys are set to AUTO\_GENERATE, with the exception of the STATE table.

INSERT into STATE values

('AK', 'Alaska'),

('AL', 'Alabama'),

('AZ', 'Arizona'),

('AR', 'Arkansas'),

('CA', 'California'),

('CO', 'Colorado'),

('CT', 'Connecticut'),

('DE', 'Delaware'),

('DC', 'District of Columbia'),

('FL', 'Florida'),

('GA', 'Georgia'),

('HI', 'Hawaii'),

('ID', 'Idaho'),

('IL', 'Illinois'),

('IN', 'Indiana'),

('IA', 'Iowa'),

('KS', 'Kansas'),

('KY', 'Kentucky'),

('LA', 'Louisiana'),

('ME', 'Maine'),

('MD', 'Maryland'),

('MA', 'Massachusetts'),

('MI', 'Michigan'),

('MN', 'Minnesota'),

('MS', 'Mississippi'),

('MO', 'Missouri'),

('MT', 'Montana'),

('NE', 'Nebraska'),

('NV', 'Nevada'),

('NH', 'New Hampshire'),

('NJ', 'New Jersey'),

('NM', 'New Mexico'),

('NY', 'New York'),

('NC', 'North Carolina'),

('ND', 'North Dakota'),

('OH', 'Ohio'),

('OK', 'Oklahoma'),

('OR', 'Oregon'),

('PA', 'Pennsylvania'),

('PR', 'Puerto Rico'),

('RI', 'Rhode Island'),

('SC', 'South Carolina'),

('SD', 'South Dakota'),

('TN', 'Tennessee'),

('TX', 'Texas'),

('UT', 'Utah'),

('VT', 'Vermont'),

('VA', 'Virginia'),

('WA', 'Washington'),

('WV', 'West Virginia'),

('WI', 'Wisconsin'),

('WY', 'Wyoming');

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

A picture containing building

Description automatically generated

**SELECT Statements**

The supported SELECT statements were all implemented as procedures. Included below are the CREATE statements that created the procedures that contain the SELECT statements. In addition, screenshots are provided after each SELECT statement showing the results of calling the stored procedure with sample inputs.

DELIMITER //

CREATE PROCEDURE get\_ski\_by\_resort\_season (resortId INT, skierId INT, seasonName VARCHAR(45))

BEGIN

SELECT TRAIL.Name, SKI.Date\_Time FROM SKI JOIN TRAIL ON SKI.TRAIL\_Trail\_Id = TRAIL.Trail\_Id

WHERE SKI.Date\_Time BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

SKI.SKIER\_Skier\_Id = skierId

AND

SKI.TRAIL\_Trail\_Id IN (SELECT Trail\_Id FROM TRAIL WHERE RESORT\_Resort\_Id = resortId);

END ;

Graphical user interface, text, application

Description automatically generated

DELIMITER //

CREATE PROCEDURE get\_ski\_by\_resort\_day (resortId INT, skierId INT, date DATE)

BEGIN

SELECT TRAIL.Name, SKI.Date\_Time FROM SKI JOIN TRAIL ON SKI.TRAIL\_Trail\_Id = TRAIL.Trail\_Id

WHERE SKI.Date\_Time = date

AND

SKI.SKIER\_Skier\_Id = skierId

AND

SKI.TRAIL\_Trail\_Id IN (SELECT Trail\_Id FROM TRAIL WHERE RESORT\_Resort\_Id = resortId);

END ;

Graphical user interface

Description automatically generated with low confidence

DELIMITER //

CREATE PROCEDURE get\_ride\_by\_resort\_day (resortId INT, skierId INT, date DATE)

BEGIN

SELECT CHAIRLIFT.Name, RIDE.Date\_Time FROM RIDE JOIN CHAIRLIFT ON RIDE.CHAIRLIFT\_Chairlift\_Id = CHAIRLIFT.Chairlift\_Id

WHERE RIDE.Date\_Time = date

AND

RIDE.SKIER\_Skier\_Id = skierId

AND

RIDE.CHAIRLIFT\_Chairlift\_Id IN (SELECT Chairlift\_Id FROM CHAIRLIFT WHERE RESORT\_Resort\_Id = resortId);

END ;

Graphical user interface

Description automatically generated with medium confidence

DELIMITER //

CREATE PROCEDURE get\_runs\_by\_rating\_and\_season (resortId INT, skierId INT, rating VARCHAR(45), seasonName VARCHAR(45))

BEGIN

SELECT COUNT(\*) FROM (

SELECT \* FROM SKI JOIN TRAIL ON SKI.TRAIL\_Trail\_Id = TRAIL.Trail\_Id

WHERE SKI.Date\_Time BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

SKI.SKIER\_Skier\_Id = skierId

AND

SKI.TRAIL\_Trail\_Id IN (SELECT Trail\_Id FROM TRAIL WHERE RESORT\_Resort\_Id = resortId)

AND

TRAIL.Rating = rating) greensSkied;

END ;

Graphical user interface, text, application

Description automatically generated

DELIMITER //

CREATE PROCEDURE get\_skier\_information (skierId INT)

BEGIN

SELECT Fname, Lname, Email

FROM SKIER

WHERE Skier\_Id = skierId;

END ;

Graphical user interface, text, application

Description automatically generated

DELIMITER //

CREATE PROCEDURE get\_earn\_by\_season (skierId INT, resortId INT, seasonName VARCHAR(45))

BEGIN

SELECT AWARD.Name, AWARD.Description, EARN.Date\_Earned FROM EARN JOIN AWARD ON EARN.AWARD\_Award\_Id = AWARD.Award\_Id

WHERE EARN.SKIER\_Skier\_Id = skierId AND EARN.Earned = TRUE AND EARN.Date\_Earned BETWEEN

(SELECT Start\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND

(SELECT End\_Date FROM SEASON

WHERE RESORT\_Resort\_Id = resortId AND Name = seasonName)

AND EARN.AWARD\_Award\_Id IN (SELECT Award\_Id FROM AWARD WHERE RESORT\_Resort\_Id = resortId);

END ;

Graphical user interface, text, application

Description automatically generated

DELIMITER //

CREATE PROCEDURE get\_chair\_status\_by\_resort (resortId INT)

BEGIN

SELECT CHAIRLIFT.Name, CHAIR\_STATUS.Description

FROM CHAIRLIFT JOIN CHAIR\_STATUS ON CHAIRLIFT.CHAIR\_STATUS\_Status\_Id = CHAIR\_STATUS.Status\_Id

WHERE RESORT\_Resort\_Id = resortId;

END ;

Table

Description automatically generated

DELIMITER //

CREATE PROCEDURE get\_resort\_info (resortId INT)

BEGIN

SELECT RESORT.Name AS Resort\_Name, CITY.Name AS City\_Name, STATE.Name AS State\_Name

FROM

RESORT JOIN CITY ON RESORT.CITY\_City\_Id = CITY.City\_Id

JOIN STATE ON CITY.STATE\_State\_Id = STATE.State\_Id

WHERE RESORT.Resort\_Id = resortId;

END ;

Graphical user interface, application

Description automatically generated with medium confidence

References

[1] https://www.mysql.com/support/supportedplatforms/workbench.html