JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Ski Tracking Application Database

Course Section: CS605.641.83
Spring, 2023

Prepared by

Christian Znidarsic
04/19/2023

Table of Contents

1. Introduction

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

1.1. 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.

1.2. 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.

2. System Requirements

2.1 Hardware Requirements

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

	Minimum	Recommended
CPU	64bit x86	Multi Core 64bit x86
RAM	4 GB	8 GB or higher
Display	1024×768	1920×1200 or higher

2.2 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:

		8.0
Operating System	**Architecture**	
Oracle Linux / Red Hat / CentOS		
Oracle Linux 9 / Red Hat Enterprise Linux 9	x86_64	•
Canonical		
Ubuntu 22.04 LTS	x86_64	•
Microsoft Windows Server		
Windows Server 2022	x86_64	•
Microsoft Windows		
Windows 11	x86_64	•
Apple		
macOS 13	x86_64, ARM 64	•
Various Linux		
Ubuntu Apt Repo		•
Fedora Yum Repo		•

2.3 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

2.4 Database Requirements

This project uses MySQL database version 8.0.31.

3. Database Design Description

3.1 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.

3.2 E/R Model

3.2.1 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.

3.2.2 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.

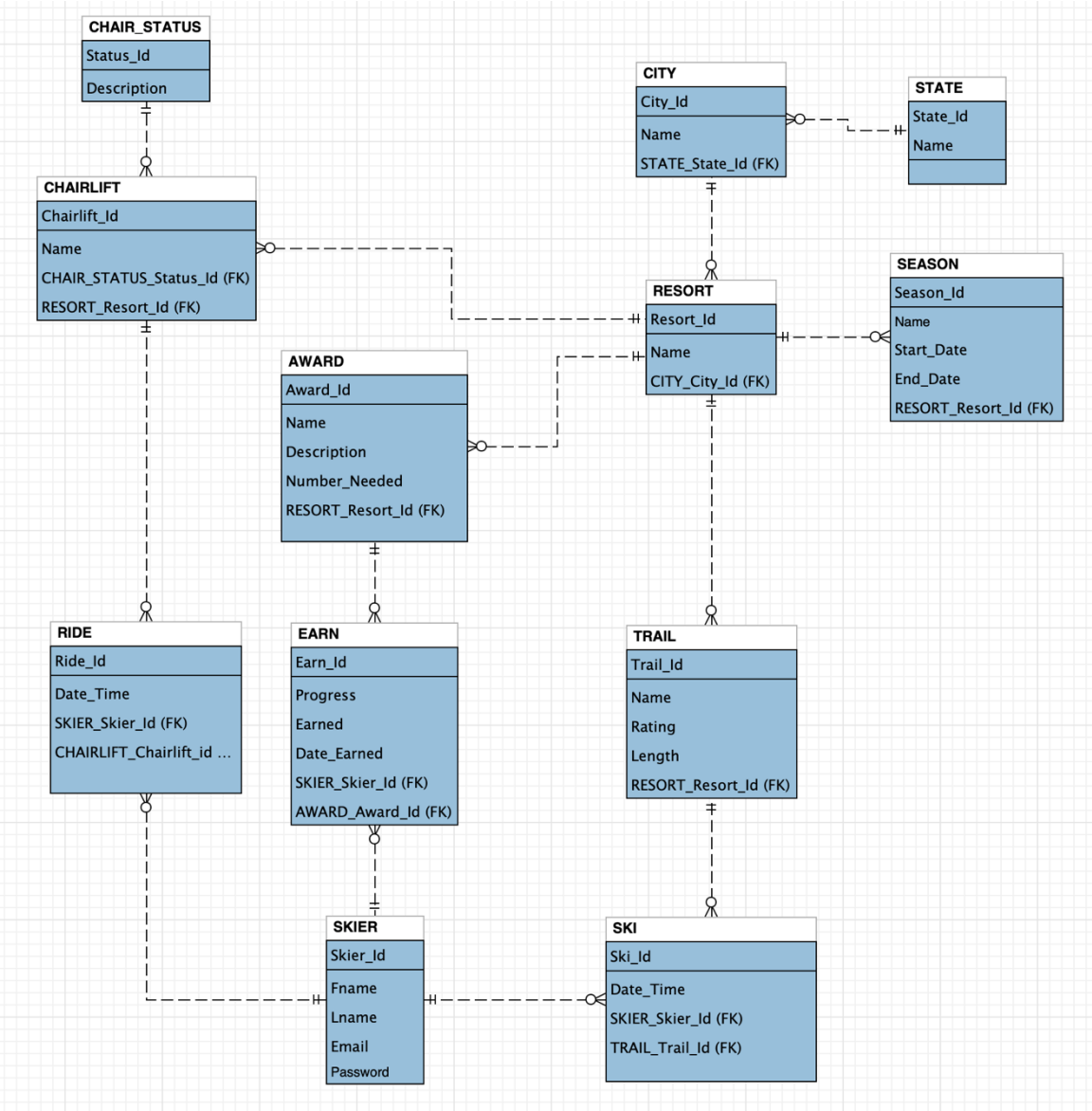
STATE (1,1) ---------- (0,m) CITY

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.

3.2.3 E/R Diagram



3.3 Relational Model

3.3.1 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

3.3.2 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.

3.3.3 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.

3.3.4 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.

3.4 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.

3.5 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.

3.6 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.

3.7 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.

4. Implementation Description

4.1 Data Dictionary

AWARD

Field	Type	Null	Key	Default	Extra
Award_Id	int	NO	PRI	NULL	auto_increment
Name	varchar(45)	NO		NULL	
Description	varchar(45)	NO		NULL	
Number_Needed	int	YES		NULL	
RESORT_Resort_Id	int	NO	MUL	NULL	

CHAIRLIFT

Field	Type	Null	Key	Default	Extra
Chairlift_Id	int	NO	PRI	NULL	auto_increment
Name	varchar(45)	NO		NULL	
CHAIR_STATUS_Status_Id	int	NO	MUL	NULL	
RESORT_Resort_Id	int	NO	MUL	NULL	

CHAIR_STATUS

Field	Type	Null	Key	Default	Extra
Status_Id	int	NO	PRI	NULL	auto_increment
Description	varchar(45)	NO		NULL	

CITY

Field	Type	Null	Key	Default	Extra
City_Id	int	NO	PRI	NULL	auto_increment
Name	varchar(45)	YES		NULL	
STATE_State_Id	varchar(2)	NO	MUL	NULL	

EARN

Field	Type	Null	Key	Default	Extra
Earn_Id	int	NO	PRI	NULL	auto_increment
Progress	int	YES		NULL	
Earned	tinyint	NO		NULL	
Date_Earned	date	YES		NULL	
SKIER_Skier_Id	int	NO	MUL	NULL	
AWARD_Award_Id	int	NO	MUL	NULL	

RESORT

Field	Type	Null	Key	Default	Extra
Resort_Id	int	NO	PRI	NULL	auto_increment
Name	varchar(45)	YES		NULL	
CITY_City_Id	int	YES	MUL	NULL	

RIDE

Field	Type	Null	Key	Default	Extra
Ride_Id	int	NO	PRI	NULL	auto_increment
Date_Time	datetime	NO		NULL	
SKIER_Skier_Id	int	NO	MUL	NULL	
CHAIRLIFT_Chairlift_id	int	NO	MUL	NULL	

SEASON

Field	Type	Null	Key	Default	Extra
Season_Id	int	NO	PRI	NULL	auto_increment
Name	varchar(45)	NO		NULL	
Start_Date	date	NO		NULL	
End_Date	date	NO		NULL	
RESORT_Resort_Id	int	NO	MUL	NULL	

SKI

Field	Type	Null	Key	Default	Extra
Ski_Id	int	NO	PRI	NULL	auto_increment
Date_Time	datetime	NO		NULL	
SKIER_Skier_Id	int	NO	MUL	NULL	
TRAIL_Trail_Id	int	NO	MUL	NULL	

SKIER

Field	Type	Null	Key	Default	Extra
Skier_Id	int	NO	PRI	NULL	auto_increment
Fname	varchar(45)	NO		NULL	
Lname	varchar(45)	NO		NULL	
Email	varchar(45)	NO		NULL	
Password	binary(45)	NO		NULL	

STATE

Field	Type	Null	Key	Default	Extra
State_Id	varchar(2)	NO	PRI	NULL	
Name	varchar(45)	NO	PRI	NULL	

TRAIL

Field	Type	Null	Key	Default	Extra
Trail_Id	int	NO	PRI	NULL	auto_increment
Name	varchar(45)	NO		NULL	
Rating	varchar(45)	NO		NULL	
Length	float(4,1)	NO		NULL	
RESORT_Resort_Id	int	NO	MUL	NULL	

4.2 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 //
```

4.3 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.

4.3.1 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 ;
```

4.3.2 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 ;
```

4.3.3 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 ;
```

4.3.4 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 ;
```

4.3.5 Skier information

```
DELIMITER //
CREATE PROCEDURE get_skier_information (skierId INT)
BEGIN
SELECT Fname, Lname, Email
FROM SKIER
WHERE Skier_Id = skierId;
END ;
```

4.3.6 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 ;
```

4.3.7 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 ;
```

4.3.8 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 ;
```

5. CRUD Matrix

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12
F1										CU		
F2									C			
F3							C					
F4					CU							
F5								R	R			R
F6									R			R
F7		R					R					
F8	R				R			R				
F9									R			R
F10	R				R							
F11		R					R					
F12										R		
F13						R						
F14		RU										
F15	CU											

5.1 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

5.2 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

6. 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_ZER
O_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');
```

Cities

Name	State_Id
Olympic Valley	CA
Mammoth Lakes	CA
Big Bear	CA
Snowmass Village	CO
Winter Park	CO

Chair_Statu

Description
Open
Hold
Closed

Resort

Name	City_Id
Palisades Tahoe	1
Mammoth Mountain	2
Big Bear	3
Snowmass	4
Winter Park	5

Chairlift

Name	Status_Id	Resort_Id
KT-22	1	1
Facelift Express	2	1
Granite Chief	3	1
Chair 23	1	2
Cloud 9 Express	1	2
High-Five Express	1	2
East-Mountain Express	1	3
All-Mountain Express	1	3
Skyline Creek	3	3

Season

Name	Start_Date	End_Date	Resort_Id
2020-2021	2020-11-15	2021-05-15	1
2021-2022	2021-11-10	2022-05-30	1
2022-2023	2022-11-01	2023-07-04	1
2020-2021	2020-11-15	2021-05-15	2
2021-2022	2021-11-10	2022-05-30	2
2022-2023	2022-11-01	2023-07-04	2
2020-2021	2020-11-15	2021-05-15	3
2021-2022	2021-11-10	2022-05-30	3
2022-2023	2022-11-01	2023-05-15	3

Award

Name	Description	Number_Needed	RESORT_Resort_Id
Trail Enthusiast	Ski 10 runs in a season	10	1
Trail Enthusiast	Ski 10 runs in a season	10	2
Trail Enthusiast	Ski 10 runs in a season	10	3

Trail

Name	Rating	Length	RESORT_Resort_Id
Snow Flower	Green	0.5	1
Siberia Bowl	Black	0.4	1
Emigrant Gully	Blue	0.4	1
Cornice Bowl	Black	0.7	2
Sunshine Glades	Black	0.8	2
Hully Gully	Blue	0.4	2
Westridge Park	Blue	0.6	3
Dicky's	Black	0.4	3
Summit Run	Green	0.8	3

Skier

Fname	Lname	Email	Password
John	Smith	jsmith@email.com	salfkjsdf
Frank	Doe	frank.doe@email.com	sdlknelsknfsd
Jimmy	Dean	dean.jimmy@email.com	ofivniwaenwef
Rick	Sanchez	rsanchez@email.com	sdfnweifnw
Don	Draper	draper.don@email.com	savpinsdsklfe

Ride

Date_Time	SKIER_Skier_Id	CHAIRLIFT_Chairlift_Id
2020-12-15	1	4
2020-12-21	1	8
2022-12-10	1	8
2023-03-15	1	6
2023-01-09	2	2
2022-01-02	2	1
2023-11-15	2	4
2020-12-21	2	6
2022-12-10	3	7
2020-12-15	3	3
2020-12-21	3	9
2022-12-10	3	7
2023-03-15	4	5
2020-12-15	4	1
2020-12-21	4	6
2022-12-10	4	3
2023-03-15	5	7
2020-12-15	5	8
2020-12-21	5	4
2022-12-10	5	7
2023-03-15	6	8
2023-01-09	6	9
2022-01-02	6	6
2023-11-15	6	1
2020-12-21	7	5
2022-12-10	7	3
2022-01-02	7	4
2020-12-15	7	7

Earn

Progress	Earned	Date_Earned	SKIER_Skier_Id	AWARD_Award_Id
9	FALSE	null	1	1
4	FALSE	null	1	2
5	FALSE	null	2	1
1	FALSE	null	3	1
10	TRUE	2021-01-19	4	2
10	TRUE	2023-03-02	5	3
5	FALSE	null	5	2
4	FALSE	null	6	1
10	TRUE	2022-01-23	6	2
10	TRUE	2023-03-09	7	1
10	TRUE	2020-01-29	7	3

Ski

Date_Time	SKIER_Skier_Id	TRAIL_Trail_Id
2021-12-15	1	4
2020-12-04	1	8
2022-12-28	1	8
2021-12-15	1	6
2020-11-04	1	2
2022-03-28	1	1
2021-02-15	1	4
2021-12-15	1	6
2020-12-04	1	7
2022-12-28	2	3
2021-12-15	2	9
2020-11-04	2	7
2022-03-28	2	5
2021-02-15	2	1
2020-11-04	2	6
2022-02-28	2	3
2020-11-04	2	7
2022-02-28	2	8
2021-12-15	3	4
2020-12-04	3	7
2022-12-28	3	8
2021-12-15	3	9
2020-11-04	3	6
2022-03-28	3	1
2021-02-15	3	5
2020-11-04	3	3
2022-02-28	3	4
2021-12-15	4	7
2020-12-04	4	6
2021-12-15	4	5
2020-12-04	4	8
2022-12-28	4	9
2021-12-15	4	3
2021-12-15	4	4
2020-12-04	4	5
2022-12-28	4	1
2021-12-15	5	2
2020-11-04	5	1
2022-03-28	5	7
2021-02-15	5	8
2020-11-04	5	9
2022-02-28	5	3
2022-12-28	5	8
2021-12-15	5	2
2020-11-04	6	7
2022-03-28	6	6
2021-02-15	6	3
2020-11-04	6	5
2022-02-28	6	4
2021-12-15	6	7
2020-11-04	7	8
2022-03-28	7	2
2021-02-15	7	1
2020-11-04	7	3
2022-02-28	7	4

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 ;
```

```
2     CALL get_ski_by_resort_season(3, 2, '2021-2022')
3
```

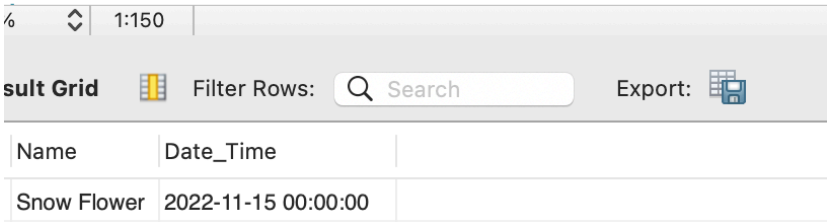
1:150

sult Grid Filter Rows: Search Export:

Name	Date_Time
Summit Run	2022-01-31 00:00:00
Dicky's	2022-01-31 00:00:00

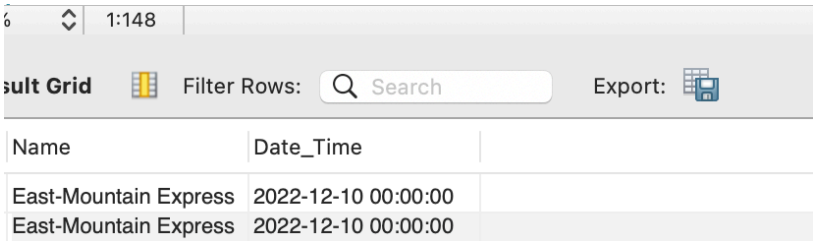
```
DELIMITER //
CREATE PROCEDURE get_ski_by_resort_day (resortId INT, skierId INT, date DATE)
BEGIN
SELECT TRAIL.Name, SKI.Date_Time FROM SKI JOIN TRAIL ON SKI.TRAIL_Trail_Id =
TRAIL.Trail_Id
WHERE SKI.Date_Time = date
   AND
   SKI.SKIER_Skier_Id = skierId
   AND
   SKI.TRAIL_Trail_Id IN (SELECT Trail_Id FROM TRAIL WHERE RESORT_Resort_Id = resortId);
END ;
```

```
3    CALL get_ski_by_resort_day(1, 3, '2022-11-15')
```




```
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 ;
```

```
3    CALL get_ride_by_resort_day(3, 3, '2022-12-10')
```




```
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 ;
```

```
5    CALL get_runs_by_rating_and_season (3, 1, 'Black', '2022-2023')
```

24:153

sult Grid Filter Rows: Search Export:

COUNT(*)
1

```
DELIMITER //
CREATE PROCEDURE get_skier_information (skierId INT)
BEGIN
SELECT Fname, Lname, Email
FROM SKIER
WHERE Skier_Id = skierId;
END ;
```

```
9    CALL get_skier_information(5)
```

1:157

sult Grid Filter Rows: Search

Fname	Lname	Email
Don	Draper	draper.don@email.com

```
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 ;
```

```
3    CALL get_earn_by_season (3, 2, '2021-2022')
```

34:180

sult Grid Filter Rows: Search Export:

Name	Description	Date_Earned
Trail Enthusiast	Ski 10 runs in a season	2022-01-23

```
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 ;
```

```
CALL get_chair_status_by_resort(1)
```

1:182

sult Grid Filter Rows: Search Ex

Name	Description
KT-22	Open
Facelift Express	Hold
Granite Chief	Closed

```
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 ;
```

```
CALL get_resort_info(1)
```

1:178

sult Grid Filter Rows: Search

Resort_Name	City_Name	State_Name
Palisades Tahoe	Olympic Valley	California

References

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