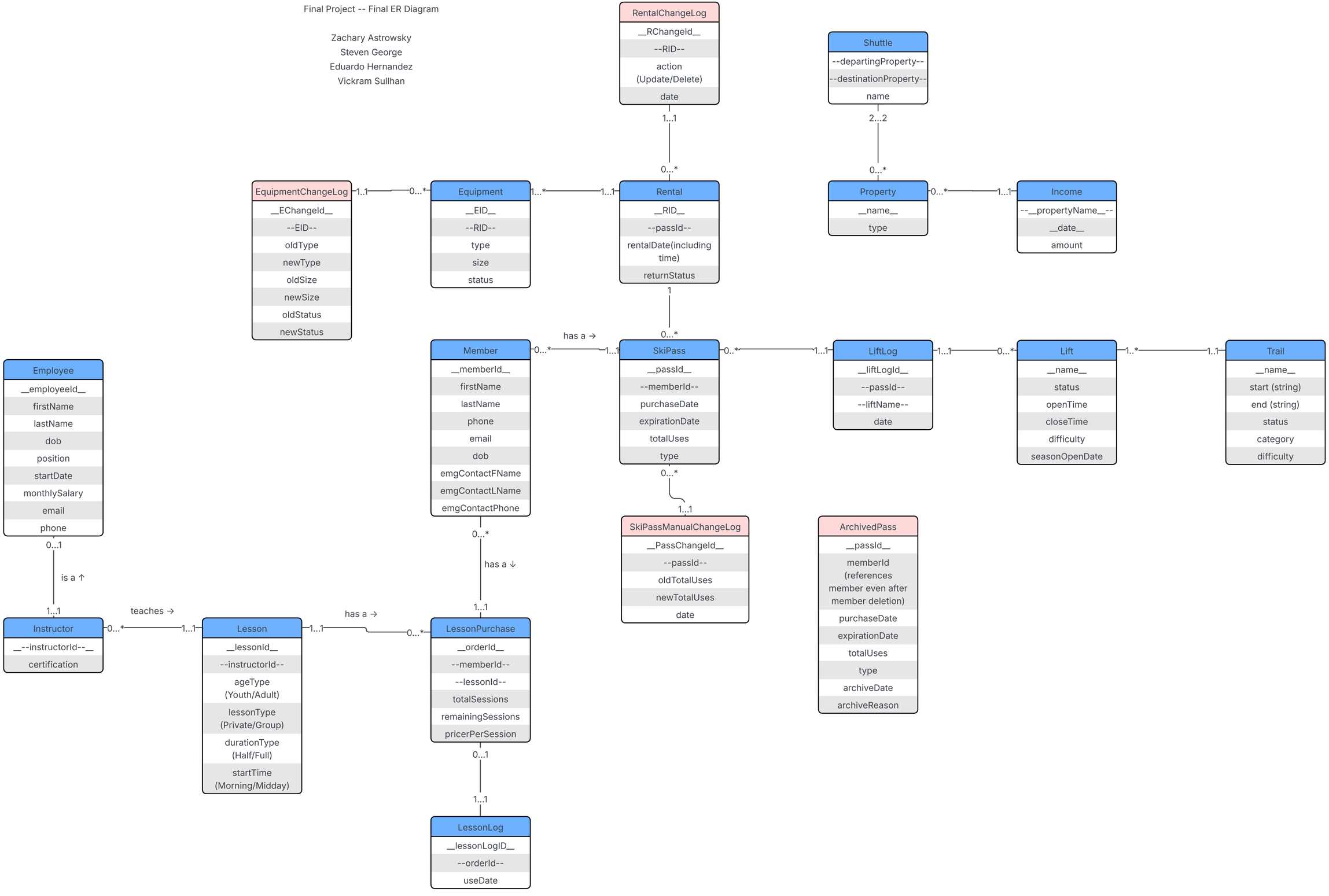
Conceptual database design: Your final E–R diagram along with your design rationale and any necessary high–level text description of the data model (e.g., constraints, or anything you were not able to show in the E–R diagram but that is necessary to help people understand your database design).

- ER diagram is the next page
- Most constraints on database inputs can be found in the logical database design section.
- Archived pass is basically a copy of SkiPass, so it uses the same pass id as its primary key since those are not reused in the database. It also contains a member id that may reference a deleted member, which is fine, since the archived pass is better for statistics and not used for tracking members.
- Instructor is a subclass of employee which allows us to store instructor specific data outside of the employee relation. If we had more employee types needed to complete the spec, we would include more relations like Instructor
- Shuttle, property, and income are listed on the spec so we included them, but they do not show up in any queries. Theoretically, to calculate profit you would sum up the incomes of each property over a time period, then subtract the monthly salaries of all employees, and add the price per session of lessons bought that month. You could also count used lessons, but we assume lessons are pre purchased, not reserved.

CSC 460
Dr. McCann
Final Project -- Final ER Diagram

Zachary Astrowsky
Steven George
Eduardo Hernandez
Vickram Sullhan



Logical database design: The conversion of your E–R schema into a relational database schema. Provide the schemas of the tables resulting from this step

Table SQL Schema:

```
CREATE TABLE ArchivedPass (
    passId          NUMBER PRIMARY KEY,
    memberId        NUMBER NOT NULL, -- REFERENCES Member(memberId),
    purchaseDate    DATE NOT NULL,
    expirationDate  DATE NOT NULL,
    totalUses       NUMBER NOT NULL,
    type            VARCHAR2(20) NOT NULL,
    archiveDate     DATE NOT NULL,
    archiveReason   VARCHAR2(255) NOT NULL
);
```

Employee SQL Schema:

```
CREATE TABLE Employee (
    employeeId NUMBER PRIMARY KEY,
    firstName VARCHAR2(30),
    lastName VARCHAR2(30),
    dob DATE,
    position VARCHAR2(30),
    startDate DATE,
    monthlySalary NUMBER(8,2),
    email VARCHAR2(100),
    phone VARCHAR2(20)
);

CREATE SEQUENCE employee_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER trg_employee_id
BEFORE INSERT ON Employee
FOR EACH ROW
BEGIN
    :NEW.employeeId := employee_seq.NEXTVAL;
END;
/
```

Equipment SQL Schema:

```
CREATE TABLE Equipment (
    EID NUMBER PRIMARY KEY,
    RID NUMBER REFERENCES Rental(RID) ON DELETE CASCADE,
    eType VARCHAR2(30) CHECK (
        eType IN ('Ski Boots', 'Ski Poles', 'Skis', 'Snowboard', 'Helmet')
    ),
    eSize VARCHAR2(20),
    eStatus VARCHAR2(20) CHECK (
        eStatus IN ('Available', 'Rented', 'Lost', 'Retired')
    )
```

```
);

CREATE SEQUENCE equipment_seq
    START WITH 1
    INCREMENT BY 1;


CREATE OR REPLACE TRIGGER trg_equipment_id
BEFORE INSERT ON Equipment
FOR EACH ROW
BEGIN
    :NEW.EID := equipment_seq.NEXTVAL;
END;
/
```

Equipment SQL Schema:

```
CREATE TABLE EquipmentChangeLog (
    EChangeId NUMBER PRIMARY KEY,
    EID NUMBER NOT NULL REFERENCES Equipment(EID)  ON DELETE CASCADE,
    oldType VARCHAR2(30)
        CHECK (oldType IN ('Ski Boots', 'Ski Poles', 'Skis', 'Snowboard',
'Helmet')),
    newType VARCHAR2(30)
        CHECK (newType IN ('Ski Boots', 'Ski Poles', 'Skis', 'Snowboard',
'Helmet')),
    oldSize VARCHAR2(20),
    newSize VARCHAR2(20),
    oldStatus VARCHAR2(20)
        CHECK (oldStatus IN ('Available', 'Rented')),
    newStatus VARCHAR2(20)
        CHECK (newStatus IN ('Available', 'Rented')),
    changeDate DATE DEFAULT SYSDATE
);

CREATE SEQUENCE echangelog_seq
START WITH 9001
INCREMENT BY 1;

CREATE OR REPLACE TRIGGER trg_echange_id
BEFORE INSERT ON EquipmentChangeLog
FOR EACH ROW
WHEN (NEW.EChangeId IS NULL)
BEGIN
    SELECT echangelog_seq.NEXTVAL INTO :NEW.EChangeId FROM dual;
END;
/
```

Income SQL Schema:

```
CREATE TABLE Income (
```

```
    propertyName  VARCHAR2(100) NOT NULL REFERENCES Property(name),
    incomeDate          DATE NOT NULL,
    amount        NUMBER NOT NULL,
    PRIMARY KEY (propertyName, incomeDate)
);
```

Table SQL Schema:

```
CREATE TABLE Instructor (
    instructorId NUMBER PRIMARY KEY REFERENCES Employee(employeeId),
    certification VARCHAR2(10)
        CHECK (certification IN ('Level I', 'Level II', 'Level III'))
);
```

Lesson SQL Schema:

```
CREATE TABLE Lesson (
    lessonId NUMBER PRIMARY KEY,
    instructorId NUMBER NOT NULL REFERENCES Instructor(instructorId),
    ageType VARCHAR2(10) CHECK (ageType IN ('Youth', 'Adult')),
    lessonType VARCHAR2(10) CHECK (lessonType IN ('Private', 'Group')),
    durationType VARCHAR2(10) CHECK (durationType IN ('Half', 'Full')),
    startTime VARCHAR2(10) CHECK (startTime IN ('Morning', 'Midday'))
);

CREATE SEQUENCE lesson_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER trg_lesson_id
BEFORE INSERT ON Lesson
FOR EACH ROW
BEGIN
    :NEW.lessonId := lesson_seq.NEXTVAL;
END;
/
```

Lesson SQL Schema:

```
CREATE TABLE LessonLog (
    lessonLogId NUMBER PRIMARY KEY,
    orderId NUMBER NOT NULL REFERENCES LessonPurchase(orderId) on delete
cascade,
    useDate DATE
);

CREATE SEQUENCE lesson_log_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER trg_lesson_log_id
BEFORE INSERT ON LessonLog
FOR EACH ROW
BEGIN
    :NEW.lessonLogId := lesson_log_seq.NEXTVAL;
END;
```

```
/
```

Lesson Purchase SQL Schema:

```
CREATE TABLE LessonPurchase (
    orderId NUMBER PRIMARY KEY,
    memberId NUMBER NOT NULL REFERENCES Member(memberId) on delete cascade,
    lessonId NUMBER NOT NULL REFERENCES Lesson(lessonId),
    totalSessions NUMBER(3) CHECK (totalSessions > 0),
    remainingSessions NUMBER(3) CHECK (remainingSessions >= 0),
    pricePerSession NUMBER(6,2) CHECK (pricePerSession >= 0)
);

CREATE SEQUENCE lesson_purchase_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER trg_lesson_purchase_id
BEFORE INSERT ON LessonPurchase
FOR EACH ROW
BEGIN
    :NEW.orderId := lesson_purchase_seq.NEXTVAL;
END;
/
```

Lift SQL Schema:

```
CREATE TABLE Lift (
    name VARCHAR2(100) PRIMARY KEY,
    status VARCHAR2(20) NOT NULL CHECK (status IN ('open', 'closed',
'maintenance')),
    openTime DATE NOT NULL, -- DATE stores time
    closeTime DATE NOT NULL,
    difficulty VARCHAR2(20) NOT NULL CHECK (difficulty IN ('beginner',
'intermediate', 'expert')),
    seasonOpenDate DATE NOT NULL
);
```

LiftLog SQL Schema:

```
CREATE TABLE LiftLog (
    liftLogId     NUMBER PRIMARY KEY,
    passId        NUMBER NOT NULL REFERENCES SkiPass(passId) ON DELETE CASCADE,
    liftName      VARCHAR2(50) REFERENCES Lift(name),
    liftLogDate   DATE
);

CREATE SEQUENCE liftlog_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER liftlog_before_insert
BEFORE INSERT ON LiftLog
FOR EACH ROW
BEGIN
    SELECT liftlog_seq.NEXTVAL INTO :new.liftLogId FROM dual;
```

```
END;
/

CREATE OR REPLACE TRIGGER liftlog_after_insert
AFTER INSERT ON LiftLog
FOR EACH ROW
BEGIN
    UPDATE SkiPass
    SET totalUses = totalUses + 1
    WHERE passId = :new.passId;
END;
/
```

Member SQL Schema:

```
CREATE TABLE Member (
      memberId           NUMBER PRIMARY KEY,
      firstName          VARCHAR2(50) NOT NULL,
      lastName           VARCHAR2(50) NOT NULL,
      phone              VARCHAR2(20) NOT NULL,
      email              VARCHAR2(100) NOT NULL,
      dob                DATE NOT NULL,
      emgContactFName  VARCHAR2(50) NOT NULL,
      emgContactLName  VARCHAR2(50) NOT NULL,
      emgContactPhone  VARCHAR2(20) NOT NULL
);

CREATE SEQUENCE member_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER member_before_insert
BEFORE INSERT ON Member
FOR EACH ROW
BEGIN
      SELECT member_seq.NEXTVAL INTO :new.memberId FROM dual;
END;
/
```

Property SQL Schema:

```
CREATE TABLE Property (
    name    VARCHAR2(100) PRIMARY KEY,
    type    VARCHAR2(50) NOT NULL
);
```

Rental SQL Schema:

```
CREATE TABLE Rental (
    RID NUMBER PRIMARY KEY,
    passId NUMBER NOT NULL REFERENCES SkiPass(passId) ON DELETE CASCADE,
    rentalDate DATE,
    returnStatus VARCHAR2(20) CHECK (returnStatus IN ('Rented', 'Available'))
);
```

```
CREATE SEQUENCE rental_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER rental_before_insert
BEFORE INSERT ON Rental
FOR EACH ROW
WHEN (NEW.RID IS NULL)
BEGIN
    SELECT rental_seq.NEXTVAL INTO :NEW.RID FROM dual;
END;
/
```

RentalChangeLog SQL Schema:

```
CREATE TABLE RentalChangeLog (
    RChangeId NUMBER PRIMARY KEY,
    RID NUMBER UNIQUE NOT NULL REFERENCES Rental(RID) ON DELETE CASCADE,
    action VARCHAR2(10) CHECK (action IN ('Update', 'Delete')),
    rentalChangeDate DATE,
    changedBy VARCHAR2(50),
    changeReason VARCHAR2(200),
    oldReturnStatus VARCHAR2(20),
    newReturnStatus VARCHAR2(20)
);

CREATE SEQUENCE rentalChangeLog_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER rentalChangeLog_before_insert
BEFORE INSERT ON RentalChangeLog
FOR EACH ROW
BEGIN
    SELECT rentalChangeLog_seq.NEXTVAL INTO :new.rChangeId FROM dual;
END;
/
```

Shuttle SQL Schema:

```
CREATE TABLE Shuttle (
    name                VARCHAR2(100) PRIMARY KEY,
    departingProperty   VARCHAR2(100) NOT NULL REFERENCES Property(name),
    destinationProperty VARCHAR2(100) NOT NULL REFERENCES Property(name)
);
```

SkiPass SQL Schema:

```
CREATE TABLE SkiPass (
    passId          NUMBER PRIMARY KEY,
    memberId        NUMBER NOT NULL REFERENCES Member(memberId),
    purchaseDate    DATE NOT NULL,
    expirationDate  DATE NOT NULL,
    totalUses       NUMBER DEFAULT 0 NOT NULL,
```

```
    type            VARCHAR2(10) NOT NULL CHECK (type IN ('1 day', '2 day', '4
day', 'season'))
);

CREATE SEQUENCE skiPass_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER skiPass_before_insert
BEFORE INSERT ON SkiPass
FOR EACH ROW
BEGIN
    SELECT skiPass_seq.NEXTVAL INTO :new.passId FROM dual;
END;
/
```

SkiPassManualChangeLog SQL Schema:

```
CREATE TABLE SkiPassManualChangeLog (
    PassChangeId    NUMBER PRIMARY KEY,
    passId          NUMBER NOT NULL REFERENCES SkiPass(passId)  ON DELETE
CASCADE,
    oldTotalUses    NUMBER NOT NULL,
    newTotalUses    NUMBER NOT NULL,
    changeDate      DATE NOT NULL
);

CREATE SEQUENCE passChange_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER passChange_before_insert
BEFORE INSERT ON SkiPassManualChangeLog
FOR EACH ROW
BEGIN
    SELECT passChange_seq.NEXTVAL INTO :new.PassChangeId FROM dual;
END;
/
```

Trail SQL Schema:

```
CREATE TABLE Trail (
    name VARCHAR2(100) PRIMARY KEY,
    startLocation VARCHAR2(100) NOT NULL,
    endLocation VARCHAR2(100) NOT NULL,
    status VARCHAR2(20) NOT NULL CHECK (status IN ('open', 'closed')),
    category VARCHAR2(20) NOT NULL CHECK (category IN ('groomed', 'park',
'moguls', 'glade')),
    difficulty VARCHAR2(20) NOT NULL CHECK (difficulty IN ('beginner',
'intermediate', 'expert'))
);
```

Normalization analysis: For each of your entity sets (tables), provide all of the FDs of the table and justify why your the table adheres to 3NF / BCNF

ArchivedPass

passId → memberId
passId → purchaseDate
passId → expirationDate
passId → totalUses
passId → type
passId → archiveDate
passId → archiveReason

3NF Justification:

All fields are non-unique – a pass could have all the same data if say it was purchases on the same date but used at different times. Therefore passId is the only primary key and is a superkey of R and all attributes only have passId as their determinant, justifying 3NF.

Employee

employeeId → firstName
employeeId → lastName
employeeId → dob
employeeId → position
employeeId → startDate
employeeId → monthlySalary
employeeId → email
employeeId → phone

3NF Justification

Employees could share all the same values assuming we do not give out unique company emails, and nothing else is unique to a single person. Thus, there are no transitive dependencies. employeeId is the only primary key, a superkey, and the only attribute on the LHS of all FD's, satisfying condition (a) of the 3NF definition

Equipment

EID → RID
EID → eType
EID → eSize
EID → eStatus

3NF Justification

All attributes except EID are non unique, RID included since multiple items of equipment may be rented for the same rental. Therefore EID is the primary key and a superkey and is the only attribute on the LHS of any FD so we satisfy (a) of our 3NF definition.

EquipmentChangeLog

EChangeId → EID
EChangeId → oldType
EChangeId → newType
EChangeId → oldSize
EChangeId → newSize
EChangeId → oldStatus
EChangeId → newStatus
EChangeId → changeDate

3NF Justification

No attributes are unique meaning there are no transitive relationships. EChangeId is the only primary key, and the only LHS of all FD's, meaning condition (a) of 3NF is satisfied.

Income

propertyName, incomeDate → amount

3NF Justification:

PropertyName and incomeDate are not unique identifiers on their own in the relation, but income is tracked daily, so combined with the property name creates a primary key, which is a superkey and makes up the LHS of the only FD placing it in 3NF

Instructor

instructorId → certification

3NF Justification:

An instructor is just a subclass of the employee relation, but separating it into a a new relation means we don't have to put the sometimes unused attribute “certification” in the employee relation, since non-instructor employees will not be certified.

So, instructorId is a primary key and is the superkey identifying the only other attribute in the relation which is not unique to any instructor, making the relation satisfy condition (a) of our 3NF definition

Lift

name → status
name → openTime
name → closeTime
name → difficulty
name → seasonOpenDate

3NF Justification:

All fields are non-unique – lifts may have the same open and close times, may be open and closed at the same time, and may be of the same difficulty and open at the same time in the season. Names are unique by design: we don’t want to confuse customers. Therefore name is the only primary key and is a superkey of R and all attributes only have name as their determinant, justifying 3NF.

LiftLog

3NF Justification:

liftLogId → passId
liftLogId → liftName
liftLogId → liftLogDate

All fields are non-unique – a person may ride the same lift multiple times on the same date. Therefore liftLogId is the only primary key and is a superkey of R and all attributes only have passId as their determinant, justifying 3NF.

Lesson

lessonId → instructorId
lessonId → ageType
lessonId → lessonType
lessonId → durationType
lessonId → startTime

3NF Justification:

All fields are non unique - multiple instructors can teach lessons of the same attribute values, and an instructor could teach multiple different lessons. Thus, lessonId is the only attribute that uniquely identifies a lesson. Each FD only has lessonId on the LHS, making condition (a) satisfied.

LessonLog

lessonLogId → orderId
lessonLogId → useDate

3NF Justification:

All fields are non-unique – while orderId's are unique to purchases, if a LessonPurchase has many uses, each use will create a new lesson log pointing to that order. One morning and one afternoon lesson (which would be a strange but possible choice) would create identical useDates. Therefore lessonLogId is the only primary key and is a superkey of R and all attributes only have passId as their determinant, justifying 3NF.

LessonPurchase

orderId → memberId
orderId → lessonId
orderId → totalSessions
orderId → remainingSessions
orderId → pricePerSession

3NF Justification:

All fields are non-unique – a member could have many lesson purchases, lessons have a finite number of ids so can appear multiple times, and the remaining could certainly be non-unique. Therefore orderId is the only primary key and is a superkey of R and all attributes only have passId as their determinant, justifying 3NF.

Member

memberId → firstName
memberId → lastName
memberId → phone
memberId → email
memberId → dob
memberId → emgContactFName
memberId → emgContactLName
memberId → emgContactPhone

3NF Justification:

Member is the only primary key and is thus a superkey.
All attributes except primary key may not be unique, thus no transitive relationships. Only superkey on LHS, therefore 3NF.

Property

name → type

3NF Justification:

All properties are uniquely named (i.e. “Parking Lot A”) but can have the same type (i.e. Parking Lot) so name is the primary key and a superkey and makes up the LHS, placing this relation in 3NF

Rental

rid → passId
rid → rentalDate
rid → returnStatus

3NF Justification:

passId is not a unique identifier because there could be multiple returned rentals with the same pass id. rentalDate is not a unique identifier because a passholder could rent skis in the morning and a snowboard in the afternoon. The same holds for status which becomes non unique as soon as the rental is returned. Therefore rid is the only primary key, is a superkey, and all the FDs are non-transitive and have the single primary key on the LHS, so the relation is in 3NF.

RentalChangeLog

A relation R is in third normal form if, for every non trivial functional dependency X -> A that holds in R, (A) X is a superkey of R, or (b) A is a prime attribute of R

RChangeId → RID
RChangeId → rentalChangeDate
RChangeId → changedBy
RChangeId → changeReason
RChangeId → oldReturnStatus
RChangeId → newReturnStatus

3NF Justification:

All fields are non-unique to a change. Therefore RChangeId is the only primary key and is a superkey of R and all attributes only have RChangeId as their determinant, justifying 3NF.

Shuttle

name → departingProperty
name → destinationProperty

3NF Justification:

Each route has a unique name making it a superkey. There could be two varieties of shuttles, etc. that go on the same route, so the departing and destination property are not unique. This means both FDs have the superkey name on the LHS satisfying condition (a)

SkiPass

passId → memberId
passId → purchaseDate
passId → expirationDate
passId → totalUses
passId → type

3NF Justification:

passId is a superkey. memberId is not a hidden primary key since a member could have many passes. Everything else is also non unique and thus there are no transitive relationships. Remaining uses can be calculated from type and total uses and was thus removed to preserve 3NF.

SkiPassManualChangeLog

passChangeId → passId
passChangeId → oldTotalUses
passChangeId → newTotalUses
passChangeId → changeDate

3NF Justification:

Everything is non-unique, so passChangeId is the only primary key and thus a superkey.
All FD’s have this superkey on the LHS, so 3NF is upheld.

Trail

name → status
name → startLocation
name → endLocation
name → status
name → category
name → difficulty

3NF Justification:

All fields are non-unique – trails can start and end at the same places, be open or closed together, and be of the same difficulty and category. Names are unique by design: we don't want to confuse customers. Therefore name is the only primary key and is a superkey of R and all attributes only have name as their determinant, justifying 3NF.

Query description: Describe your self–designed query. Specifically, what question is it answer- ing, and what is the utility of including such a query in the system

```
SELECT eq.EID as EquipmentID,
                eq.ETYPE as EquipmentType,
                eq.ESIZE as EquipmentSize,
                eq.ESTATUS as CurrentStatus,
                log.OLDTYPE,
                log.NEWTYPE,
                log.OLDSIZE,
                log.NEWSIZE,
                log.OLDSTATUS,
                log.NEWSTATUS,
                TO_CHAR(log.changeDate, 'YYYY-MM-DD HH24:MI:SS') AS
                     ChangeDate
FROM EquipmentChangeLog log
JOIN Equipment eq ON log.EID = eq.EID
WHERE eq.EID = ? ORDER BY log.changeDate DESC;
```

This query retrieves the equipment change history for a specific piece of equipment identified by the user input equipment id. It shows the modifications made over time, like type, size, and status. This allows us to evaluate the performance of specific equipment types and evaluate life cycles which is beneficial for tracking which equipment performs the best. For example, low maintenance equipment is ideal and most cost effective. It can tell us how well employees track changes. For example, boots shouldn't change size, but if they do we might have issues with proper logging.