```sql
-- Création de la table Moniteur
CREATE TABLE `Moniteur` (
   `idMoniteur` INTEGER AUTO_INCREMENT,
   `Prenom` VARCHAR(50),
   `nom` VARCHAR(50),
   PRIMARY KEY(`idMoniteur`)
);

-- Création de la table Poney
CREATE TABLE `Poney` (
   `idPoney` INTEGER AUTO_INCREMENT,
   `nomPoney` VARCHAR(50),
   `charge_max` DECIMAL(5,2),
   PRIMARY KEY(`idPoney`)
);

-- Création de la table Adherent
CREATE TABLE `Adherent` (
   `idAdherent` INTEGER AUTO_INCREMENT,
   `poids` DECIMAL(5,2),
   `nom` VARCHAR(50),
   `cotisation` BOOLEAN,
   `Telephone` VARCHAR(20) CHECK (Telephone REGEXP '^[0-9]{10}$'),
   PRIMARY KEY(`idAdherent`)
);

-- Création de la table CoursProgramme
CREATE TABLE `CoursProgramme` (
   `idCours` INTEGER AUTO_INCREMENT,
   `Duree` INTEGER CHECK (`Duree` <= 2 ),
   `DateJour` DATE,
   `Semaine` INTEGER,
   `Heure` TIME,
   `Prix` DECIMAL(10,2),
   `Niveau` VARCHAR(20),
   `NbPersonne` INTEGER CHECK (`NbPersonne` <= 10),
   PRIMARY KEY(`idCours`)
);

-- Création de la table CoursRealise avec la clé étrangère vers
CoursProgramme
CREATE TABLE `CoursRealise` (
   `idCoursRealise` INTEGER AUTO_INCREMENT,
```

```
    `DateJour` DATE NOT NULL,
    `Semaine` INTEGER,
    `Mois` INTEGER,
    `idCours` INTEGER,
    PRIMARY KEY(`idCoursRealise`),
    FOREIGN KEY(`idCours`) REFERENCES `CoursProgramme`(`idCours`) ON
UPDATE NO ACTION ON DELETE NO ACTION
);

-- Création de la table Reserver avec les clés étrangères
CREATE TABLE `Reserver` (
    `idReserver` INTEGER AUTO_INCREMENT,
    `idCoursRealise` INTEGER,
    `idAdherent` INTEGER,
    `idPoney` INTEGER,
    PRIMARY KEY(`idReserver`),
    FOREIGN KEY(`idCoursRealise`) REFERENCES
`CoursRealise`(`idCoursRealise`) ON UPDATE NO ACTION ON DELETE NO
ACTION,
    FOREIGN KEY(`idAdherent`) REFERENCES `Adherent`(`idAdherent`) ON
UPDATE NO ACTION ON DELETE NO ACTION,
    FOREIGN KEY(`idPoney`) REFERENCES `Poney`(`idPoney`) ON UPDATE NO
ACTION ON DELETE NO ACTION
);

-- Création de la table Anime avec les clés étrangères
CREATE TABLE `Anime` (
    `idMoniteur` INTEGER NOT NULL,
    `idCours` INTEGER NOT NULL,
    PRIMARY KEY(`idMoniteur`, `idCours`),
    FOREIGN KEY(`idMoniteur`) REFERENCES `Moniteur`(`idMoniteur`) ON
UPDATE NO ACTION ON DELETE NO ACTION,
    FOREIGN KEY(`idCours`) REFERENCES `CoursRealise`(`idCoursRealise`)
ON UPDATE NO ACTION ON DELETE NO ACTION
);


-- Permet de verifier que l'on peut faire une reservation

DELIMITER $$

CREATE TRIGGER `before_insert_reserver`
BEFORE INSERT ON `Reserver`
```

```
FOR EACH ROW
BEGIN
   DECLARE adherent_poids DECIMAL(5,2);
   DECLARE poney_charge_max DECIMAL(5,2);
   DECLARE cotisation_valide BOOLEAN;

   -- Récupération du poids de l'adhérent
   SELECT poids, cotisation INTO adherent_poids, cotisation_valide FROM
Adherent WHERE idAdherent = NEW.idAdherent;

   -- Vérification de la cotisation
   IF cotisation_valide = FALSE THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La cotisation de
l\'adhérent n\'est pas à jour.';
   END IF;

   -- Récupération de la charge maximale du poney
   SELECT charge_max INTO poney_charge_max FROM Poney WHERE idPoney =
NEW.idPoney;

   -- Vérification du poids de l'adhérent par rapport à la charge
maximale du poney
   IF adherent_poids > poney_charge_max THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le poids de
l\'adhérent dépasse la charge maximale du poney.'';
   END IF;

END$$

DELIMITER ;


DELIMITER |

CREATE TRIGGER verif_moniteur_cours
BEFORE INSERT ON Anime
FOR EACH ROW
BEGIN
   DECLARE heureDebutCoursNew TIME;
   DECLARE heureFinCoursNew TIME;
   DECLARE moniteurConflit INTEGER;
```

```
    SELECT Heure, ADDTIME(Heure, SEC_TO_TIME(Duree * 3600)) INTO
heureDebutCoursNew, heureFinCoursNew
    FROM CoursProgramme
    WHERE idCours = NEW.idCours;

    SELECT COUNT(*)
    INTO moniteurConflit
    FROM CoursProgramme cp
    JOIN CoursRealise cr ON cr.idCours = cp.idCours
    JOIN Anime a ON a.idCours = cr.idCoursRealise
    WHERE a.idMoniteur = NEW.idMoniteur
    AND cr.DateJour = (SELECT DateJour FROM CoursRealise WHERE
idCoursRealise = NEW.idCours)
    AND (
        (heureDebutCoursNew BETWEEN cp.Heure AND ADDTIME(cp.Heure,
SEC_TO_TIME(cp.Duree * 3600)))
        OR
        (heureFinCoursNew BETWEEN cp.Heure AND ADDTIME(cp.Heure,
SEC_TO_TIME(cp.Duree * 3600)))
        OR
        (cp.Heure BETWEEN heureDebutCoursNew AND heureFinCoursNew)
    );

    IF moniteurConflit > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Un moniteur ne peut pas faire plusieurs
cours en même temps.';
    END IF;

END|

DELIMITER ;

-- Trigger pour vérifier qu'un cours n'est pas réservé par trop de
personnes
DELIMITER |
CREATE TRIGGER cours_plein
BEFORE INSERT ON Reserver
FOR EACH ROW
BEGIN
    DECLARE nbPersonneTotale INTEGER;
    DECLARE nbPersonneInscrite INTEGER;
```

```
   SELECT COUNT(idAdherent),NbPersonne INTO nbPersonneInscrite,
nbPersonneTotale
   FROM Reserver NATURAL join CoursRealise
   WHERE idCoursRealise = NEW.idCoursRealise;

   IF nbPersonneInscrite > nbPersonneTotale THEN
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT = 'Le nombre d''adhérents inscrits est
supérieur au nombre total de places disponibles.';
   END IF;
END
|

DELIMITER ;

-- Trigger pour vérifier qu'un poney n'est pas trop lourd pour un
adhérent
DELIMITER |
CREATE TRIGGER Poids_Trop_Lourd
BEFORE INSERT ON Reserver
FOR EACH ROW
BEGIN
   DECLARE poidsAdherent DECIMAL(5,2);
   DECLARE chargeMaxPoney DECIMAL(5,2);
   SELECT poids, charge_max INTO poidsAdherent, chargeMaxPoney
   FROM Adherent NATURAL join RESERVER NATURAL JOIN PONEY WHERE
idAdherent = NEW.idAdherent;

   IF poidsAdherent > chargeMaxPoney THEN
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT = 'Le poids de l''adhérent dépasse la charge
maximale du poney.';
   END IF;
END |
delimiter ;

-- Trigger pour vérifier qu'un poney n'est pas en état de repos
DELIMITER |
CREATE TRIGGER Poney_repose
BEFORE INSERT ON Reserver
FOR EACH ROW
BEGIN
   DECLARE conflict_count INT;
```

```
    SELECT COUNT(*)
    INTO conflict_count
    FROM Reserver
    WHERE idPoney = NEW.idPoney
    AND DateJour = NEW.DateJour
    AND TIMESTAMPADD(HOUR, Duree, Heure) <= TIMESTAMPADD(HOUR, -1,
NEW.Heure);

    -- Si on trouve un conflit, on lève une erreur
    IF conflict_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Le poney est déjà réservé dans l\'heure
précédente ou suivante.'';
    END IF;
END;
|
DELIMITER ;
-- Trigger pour vérifier qu'un adhérent à payer sa cotisation lors
d'une réservation
DELIMITER |
CREATE TRIGGER Cotisation_Pas_Payer
BEFORE INSERT ON Reserver
FOR EACH ROW
DECLARE
    cotisation BOOLEAN;
BEGIN
SELECT cotisation INTO cotisation
FROM Adherent
WHERE idAdherent = NEW.idAdherent;
IF cotisation = FALSE THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'L''adhérent n''a pas payé sa cotisation.';
END IF;
END;
```

```sql
-- Suppression des tables si elles existent déjà pour éviter les
conflits
DROP TABLE IF EXISTS `Anime`;
DROP TABLE IF EXISTS `Reserver`;
DROP TABLE IF EXISTS `CoursRealise`;
DROP TABLE IF EXISTS `CoursProgramme`;
DROP TABLE IF EXISTS `Adherent`;
DROP TABLE IF EXISTS `Poney`;
DROP TABLE IF EXISTS `Moniteur`;
```

```sql
-- Insertion des données dans la table Moniteur
INSERT INTO `Moniteur` (`Prenom`, `nom`) VALUES
('Jean', 'Dupont'),
('Marie', 'Durand'),
('Pierre', 'Martin');

-- Insertion des données dans la table Poney
INSERT INTO `Poney` (`nomPoney`, `charge_max`) VALUES
('Bella', 50.00),
('Charlie', 55.00),
('Daisy', 60.00);

-- Insertion des données dans la table Adherent
INSERT INTO `Adherent` (`poids`, `nom`, `cotisation`, `Telephone`)
VALUES
(45.50, 'Alice', TRUE, '0601020304'),
(52.00, 'Bob', TRUE, '0605060708'),
(48.00, 'Clara', TRUE, '0608091011');

-- Insertion des données dans la table CoursProgramme
INSERT INTO `CoursProgramme` (`Duree`, `DateJour`, `Semaine`, `Heure`,
`Prix`, `Niveau`, `NbPersonne`) VALUES
(2, '2023-11-15', 46, '10:00:00', 50.00, 'Débutant', 8),
(3, '2023-11-16', 46, '14:00:00', 75.00, 'Intermédiaire', 6),
(1, '2023-11-17', 46, '16:00:00', 40.00, 'Avancé', 4);
--  Pour vérifier le trigger moniteur verif_cours (4, '2023-11-15', 46,
'10:00:00', 50.00, 'Débutant', 8);

-- Insertion des données dans la table CoursRealise
INSERT INTO `CoursRealise` (`DateJour`, `Semaine`, `Mois`, `idCours`)
VALUES
('2023-11-15', 46, 11, 1),
('2023-11-16', 46, 11, 2),
('2023-11-17', 46, 11, 3);
--  pour vérifier le trigger moniteur verif_cours('2023-11-15', 46, 11,
4);

-- Insertion des données dans la table Reserver
INSERT INTO `Reserver` (`idCoursRealise`, `idAdherent`, `idPoney`)
VALUES
(1, 1, 1),
(1, 2, 2),
(2, 1, 3),
```

```
(2, 3, 1),
(3, 2, 2),
(3, 3, 3);

-- Insertion des données dans la table Anime
INSERT INTO `Anime` (`idMoniteur`, `idCours`) VALUES
(1, 1),
(2, 2),
(3, 3);
```