ICT 128-048 Assignment: Relational Database

Part II – Database Creation & Use

Bruno Vieira Ribeiro

The ERD was not considerably changed from phase I of this project. All changes are listed bellow:

- The Foreign Key *S_ID* was removed from the *PAYMENT* table.
- Some *VARCHAR* variables had their length increased to accomodate data.

Queries on the database

1. As a customer I want to list all the action movies:

```
SELECT
    *
FROM
    CONTENT
WHERE
    type = 'movie' AND genre = 'action';
```

2. As a manager I want to know the number of movies currently available at the “no limit” tier:

```
SELECT
    COUNT(*)
FROM
    CONTENT
WHERE
    type = 'movie' AND tier = 'no-limits';
```

3. As a customer I want to change my tier from free to mid-level.

Assumptions: as a customer you can only change the tier for the account you are under. To change tiers, the customer will need to know: 'account ID'. In the code bellow, we are changing the tier for account with ID 1 (which was **free** tier in the original data).

```
UPDATE ACCOUNT
SET
    plan = 'mid-tier'
WHERE
    ACC_ID = 1;
```

4. As a customer I want to create a new account on the system at the free tier level.

This will be done in two steps here:

i) Creating the free account:

```
INSERT INTO
ACCOUNT (CT_CODE, plan, password)
VALUES ("BRA", "free", "123password")
```

ii) Link customer to new account by getting the *MAX(ACC_ID)* of the newly created account.

```
INSERT INTO CUSTOMER (ACC_ID, f_name, l_name, email)
SELECT MAX(ACC_ID), "Bruno", "Ribeiro", "me@place.ca"
FROM ACCOUNT;
```

5. As a customer, I want to add an additional email to the account (give another person access to the account).

Assumption: the customer knows the account id.

```
INSERT INTO CUSTOMER (ACC_ID, f_name, l_name, email)
VALUES (10001, "Natalia", "Sena", "her@place.ca");
```

6. As a manager I want to know all the horror movies where at least one customer has given it a 5 rating.

Assumption: Manager wants the ID of all movies under her request. The query will return the C_ID of these movies.

```
SELECT
    s.C_ID, c.type, c.genre, s.star_rate, COUNT(*) AS number_of_ratings
FROM
    STREAM AS s
INNER JOIN
    CONTENT AS c
        ON s.C_ID = c.C_ID
WHERE
    c.type = "movie" AND c.genre = "horror" AND s.star_rate = 5
GROUP BY
    s.C_ID, s.star_rate
HAVING
    number_of_ratings >= 1;
```

7. As a manager I want to change all the content from a particular distributor as unavailable because the contract with the distributor has expired.

Given an DIS_ID (100, in this example), set all *available* fields to 0

```
UPDATE AVAILABILITY AS a
INNER JOIN CONTENT AS c
    ON a.C_ID = c.C_ID
SET
    a.available = 0
WHERE c.DIS_ID = 100;
```

8. As a manager I want to delete a customer that is on a mid-tier plan whose subscription is paid for 4 more months.

Assumption: All subscription payments are for one year.

I had two understandings from this demand.

i) Understanding one: manager wants to delete all customers under description.

```
DELETE cx.*
FROM
    CUSTOMER AS cx
INNER JOIN
    ACCOUNT AS a
    ON cx.ACC_ID = a.ACC_ID
INNER JOIN
    PAYMENT AS p
    ON a.ACC_ID = p.ACC_ID
INNER JOIN
    DATE_KEEP as d
    ON p.D_ID = d.D_ID
WHERE
    purpose = "subs" AND plan = "mid-tier" AND date_time > (NOW() -
INTERVAL 8 MONTH);
```

ii) Understanding two: manager wants a specific customer under this description. To this purpose, here is a list of all customers under the description:

```
SELECT
    a.ACC_ID, cx.CX_ID, purpose, success, date_time, plan
FROM
    CUSTOMER AS cx
INNER JOIN
    ACCOUNT AS a
    ON cx.ACC_ID = a.ACC_ID
INNER JOIN
```

```
    PAYMENT AS p
    ON a.ACC_ID = p.ACC_ID
INNER JOIN
    DATE_KEEP as d
    ON p.D_ID = d.D_ID
WHERE
    purpose = "subs" AND plan = "mid-tier" AND date_time > (NOW() -
INTERVAL 8 MONTH);
```

Having this, manager could see the customer ID matching her demand and delete that customer.

ISSUE: The artificial data that was generated for testing here may not have any customers under this description.

9. As a manager I want to find the average star rating for every genre of all the tv-series that were viewed in the last month.

```
SELECT
    c.type, c.genre, AVG(s.star_rate) AS avg_star_rating
FROM
    STREAM AS s
INNER JOIN
    CONTENT AS c
USING (C_ID)
WHERE
    c.type = "tv-serie"
    AND
    c.last_viewed > (NOW() - INTERVAL 1 MONTH)
GROUP BY
    c.genre;
```

Data Integrity and security measures

Other than the NOT NULL constraints defined in the tavle creation, the following additional constraints were implemented:

- To set the possible values for the tier (or plans) in the ACCOUNT and CONTENT tables:

```
ALTER TABLE ACCOUNT
ADD CONSTRAINT chk_plan CHECK (plan IN ('free','mid-tier','no-limits'))
```

```
ALTER TABLE CONTENT
ADD CONSTRAINT chk_tier CHECK (tier IN ('free','mid-tier','no-limits'))
```

- To set the possible options for payment type in the PAYMENT_TYPE table:

```
ALTER TABLE PAYMENT_TYPE
ADD CONSTRAINT chk_paytype CHECK (type IN ('credit', 'debit', 'OPS'))
```

To insert data into the DATE_KEEP table, which is date table keeping track of all time-related operations in the system, we used a **trigger** to automatically detect insertions into the STREAM and PAYMENT tables and do the corresponding insertion in the DATE_KEEP table:

- Trigger to update DATE_KEEP after insertion into STREAM:

```
CREATE TRIGGER stream_date_trigg
    BEFORE INSERT
    ON STREAM FOR EACH ROW
    INSERT INTO DATE_KEEP(D_ID, date_time)
    VALUES(NEW.D_ID,
           FROM_UNIXTIME( UNIX_TIMESTAMP('2010-04-30 14:53:27') + FLOOR(0 +
(RAND() * 25920000) ) )
    );
```

NOTE: For the purpose of generating our data artificially, the value for the *date_time* feature was chosen randomly from the end of April 2010 up to 300 days forward (25920000 seconds). In a "real" application, this would just be the current time (`NOW()`).

- Trigger to update DATE_KEEP after insertion into PAYMENT:

```
CREATE TRIGGER payment_date_trigg
    BEFORE INSERT
    ON PAYMENT FOR EACH ROW
    INSERT INTO DATE_KEEP(D_ID, date_time)
    VALUES(NEW.D_ID,
           FROM_UNIXTIME( UNIX_TIMESTAMP('2010-04-30 14:53:27') + FLOOR(0 +
(RAND() * 25920000) ) )
    );
```

NOTE: Same issue with the randomly generated dates for testing purposes.

- Trigger to prevent more than 4 customers in each account.

```
DELIMITER //
CREATE TRIGGER PreventMoreCustomers BEFORE INSERT ON CUSTOMER
FOR EACH ROW
BEGIN
    IF(
        (SELECT COUNT(*)
         FROM ACCOUNT
         WHERE ACC_ID = NEW.ACC_ID
         GROUP BY ACC_ID) > 4
    ) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Account maxed out';
    END IF;
END //
DELIMITER ;
```

NOTE: If a new customer tries to be inserted into an account that already has 4 customers, the message "*Account maxed out*" will be displayed.

Restrictions:

- No checks were implemented on passwords.
- When a customer or account is deleted, we decided to keep their payment and streaming information to have the historical data on our content.
- No checks were implemented to validate e-mails.
- Customer, account, content and distributor IDs are all auto-incremented integers so they carry no real information.

Database implementation

- Creating the database and tables:

```
CREATE DATABASE ict128;

CREATE TABLE IF NOT EXISTS `COUNTRY` (
  `CT_CODE` VARCHAR(3),
  `ct_name` VARCHAR(100) NOT NULL,
  `continent` VARCHAR(10) NOT NULL,
  `currency` VARCHAR(4) NOT NULL,
  PRIMARY KEY (`CT_CODE`)
);

CREATE TABLE IF NOT EXISTS `ACCOUNT` (
  `ACC_ID` INT AUTO_INCREMENT,
  `CT_CODE` VARCHAR(3) NOT NULL,
  `plan` VARCHAR(9) NOT NULL,
  `password` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`ACC_ID`),
  FOREIGN KEY (`CT_CODE`) REFERENCES `COUNTRY`(`CT_CODE`)
);

CREATE TABLE IF NOT EXISTS `CUSTOMER` (
  `CX_ID` INT AUTO_INCREMENT,
  `ACC_ID` INT NOT NULL,
  `f_name` VARCHAR(20) NOT NULL,
  `l_name` VARCHAR(50) NOT NULL,
  `email` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`CX_ID`),
  FOREIGN KEY (`ACC_ID`) REFERENCES `ACCOUNT`(`ACC_ID`)
```

```
);

CREATE TABLE IF NOT EXISTS `DISTRIBUTOR` (
  `DIS_ID` INT AUTO_INCREMENT,
  `CT_CODE` VARCHAR(3) NOT NULL,
  PRIMARY KEY (`DIS_ID`),
  FOREIGN KEY (`CT_CODE`) REFERENCES `COUNTRY`(`CT_CODE`)
);

CREATE TABLE IF NOT EXISTS `CONTENT` (
  `C_ID` INT AUTO_INCREMENT,
  `DIS_ID` INT NOT NULL,
  `last_viewed` DATETIME,
  `type` VARCHAR(20),
  `tier` VARCHAR(9) NOT NULL,
  `description` TEXT,
  `genre` VARCHAR(20),
  PRIMARY KEY (`C_ID`),
  FOREIGN KEY (`DIS_ID`) REFERENCES `DISTRIBUTOR`(`DIS_ID`)
);

CREATE TABLE IF NOT EXISTS `DATE_KEEP` (
  `D_ID` INT,
  `date_time` DATETIME NOT NULL,
  PRIMARY KEY (`D_ID`)
);

CREATE TABLE IF NOT EXISTS `PAYMENT_TYPE` (
  `PT_ID` INT AUTO_INCREMENT,
  `type` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`PT_ID`)
);

CREATE TABLE IF NOT EXISTS `STREAM` (
  `S_ID` INT AUTO_INCREMENT,
  `ACC_ID` INT NOT NULL,
  `C_ID` INT NOT NULL,
  `P_ID` INT,
  `D_ID` INT NOT NULL,
  `perc_watched` INT NOT NULL,
  `star_rate` INT,
  PRIMARY KEY (`S_ID`),
  FOREIGN KEY (`C_ID`) REFERENCES `CONTENT`(`C_ID`),
  FOREIGN KEY (`D_ID`) REFERENCES `DATE_KEEP`(`D_ID`),
  FOREIGN KEY (`ACC_ID`) REFERENCES `ACCOUNT`(`ACC_ID`)
);

CREATE TABLE IF NOT EXISTS `PAYMENT` (
  `P_ID` INT AUTO_INCREMENT,
  `ACC_ID` INT NOT NULL,
  `PT_ID` INT NOT NULL,
  `D_ID` INT NOT NULL,
  `amount` DECIMAL(6,2) NOT NULL,
  `purpose` VARCHAR(10) NOT NULL,
```

```sql
  `success` BOOL NOT NULL,
  `expiration` DATE,
  PRIMARY KEY (`P_ID`),
  FOREIGN KEY (`ACC_ID`) REFERENCES `ACCOUNT`(`ACC_ID`),
  FOREIGN KEY (`D_ID`) REFERENCES `DATE_KEEP`(`D_ID`),
  FOREIGN KEY (`PT_ID`) REFERENCES `PAYMENT_TYPE`(`PT_ID`)
);

CREATE TABLE IF NOT EXISTS `AVAILABILITY` (
  `C_ID` INT NOT NULL,
  `CT_CODE` VARCHAR(3) NOT NULL,
  `available` BOOL NOT NULL,
  `available_date` DATE,
  `price` DECIMAL(5,2),
  PRIMARY KEY (C_ID, CT_CODE),
  FOREIGN KEY (`CT_CODE`) REFERENCES `COUNTRY`(`CT_CODE`),
  FOREIGN KEY (`C_ID`) REFERENCES `CONTENT`(`C_ID`)
);

CREATE TABLE IF NOT EXISTS `CONTACTS` (
  `CONTACT_ID` INT AUTO_INCREMENT,
  `DIS_ID` INT NOT NULL,
  `f_name` VARCHAR(20) NOT NULL,
  `l_name` VARCHAR(50) NOT NULL,
  `location` VARCHAR(20),
  `dept` VARCHAR(20),
  `email` VARCHAR(100) NOT NULL,
  `phone#` INT NOT NULL,
  PRIMARY KEY (`CONTACT_ID`),
  FOREIGN KEY (`DIS_ID`) REFERENCES `DISTRIBUTOR`(`DIS_ID`)
);
```

Generating data

All data was artificially generated using a python script named **generating-data.py**.

The required modules are:

- `mysql.connector` -> to interact with the database
- `numpy` -> to generated numerical data
- `pandas` -> to read and write data on countries
- `silly` -> to generated descriptions and locations
- `faker` -> to generate names and emails