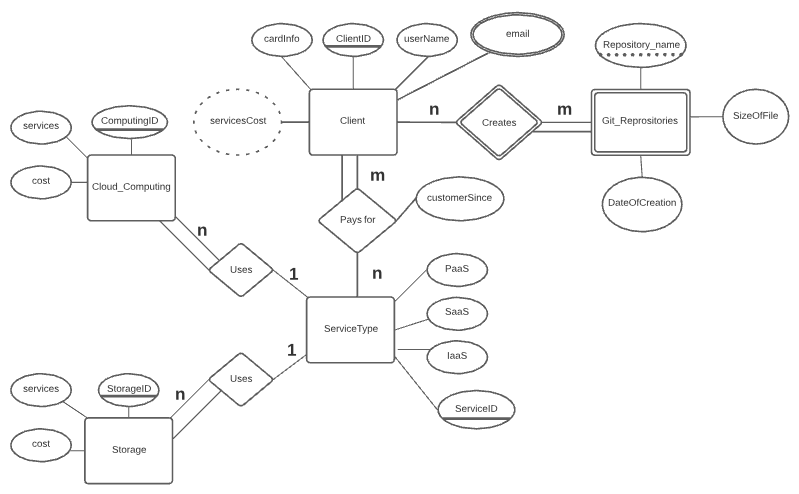
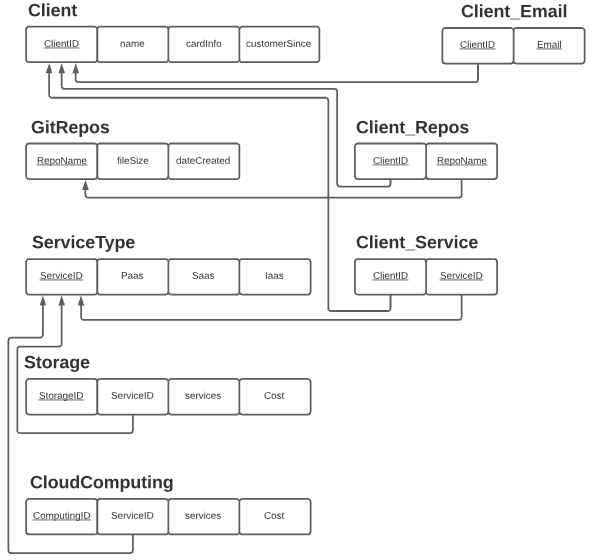
AWS Database:

The Purpose of this Database is to provide an example of a smallscale implementation for Amazon Web Services. The database will track each Clients username, card info, email, service costs and unique ID. The database will also track repositories made by cliental. Each client may create zero or more Git-Repositories; however, a repository must be created by a client. Repositories may be created by multiple clients and are not required have unique names and therefore must be tied to the client(s) that create it. The database will track the repositories name, date of creation and the size of the file. Every client must pay for one or more services while every service does not need to be tied to every client. The database will track the types of services paid for as Iaas, Paas, and Saas. The clients purchase will be tracked using a unique serviceID. The database will also track the date in which the person or corporation became a client as customerSince. The database will track the storage power and computing power paid for by the client. Storage power and computing power must be paid for as part of one of the types of services: Iaas, Paas, and Saas. The client may purchase zero or more types of storage or computing power services. The database will use a StorageID and CloudComputingID to track each of the services paid for by the client for storage power and computing power separately.

**Entity-Relationship Diagram**



**Relational Schema Diagram**



**Query 1: fn\_ServiceBudget**

Reason: Uses a function that has three functions nested in it to add Storage(fn\_storageCost),

CloudComputing(fn\_compCost), and Repository(fn\_repoCost), costs to show the total payment for services alloted to each ClientID.

USE bharris\_cs355fa21;

DELIMITER //

CREATE OR REPLACE FUNCTION fn\_ServiceBudget

(

ClientID INT,

ServiceID INT

)

RETURNS DECIMAL (10,2) UNSIGNED

BEGIN

DECLARE totalDue DECIMAL (10,2) UNSIGNED;

DECLARE serviceCost DECIMAL (10,2) UNSIGNED;

DECLARE repoName VARCHAR (75);

DECLARE repoCost DECIMAL (5,2) UNSIGNED;

SELECT fn\_compCost(ServiceID) + fn\_storageCost(ServiceID)

INTO serviceCost;

IF serviceCost IS NULL THEN

SET serviceCost = 0;

END IF;

SELECT SUM(fn\_repoCost(gr.RepoName))

INTO repoCost

FROM Client AS c

JOIN Client\_Repos as cr

ON c.ClientID = cr.ClientID

JOIN GitRepos AS gr

ON gr.RepoName = cr.RepoName

WHERE c.ClientID = ClientID;

IF repoCost IS NULL THEN

SET repoCost = 0;

END IF;

SET totalDue = serviceCost + repoCost;

RETURN totalDue;

END //

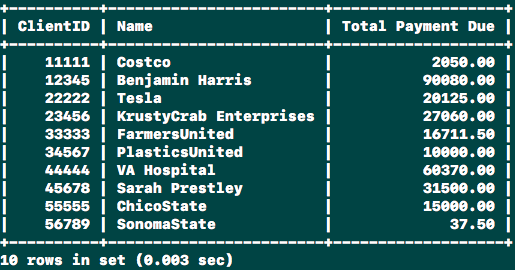
DELIMITER ;

MySQL:

SELECT cs.ClientID,

name AS 'Name', fn\_serviceBudget(cs.ClientID, cs.ServiceID) AS 'Total Payment Due'

FROM Client\_Service cs JOIN Client c ON cs.ClientID = c.ClientID;



**Query 2: Cliental\_Portfolio**

Reason: : Uses a view with the functions used earlier, group by, several joins to show all of the services and costs related to each client ID.

USE bharris\_cs355fa21;

CREATE OR REPLACE VIEW Cliental\_Portfolio AS

SELECT

cs.ClientID,

fn\_compCost(ServiceID) AS 'Computing Cost',

fn\_storageCost(ServiceID) AS 'Storage Cost',

SUM(fn\_repoCost(RepoName)) AS 'Repository Storage Cost',

fn\_serviceBudget(cs.ClientID, ServiceID) AS 'Total Due'

FROM Client\_Service AS cs

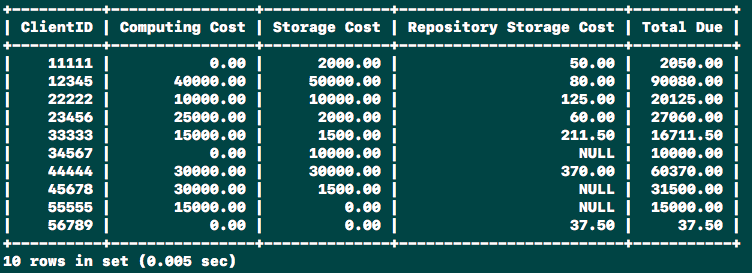
LEFT JOIN Client\_Repos AS cr

ON cs.ClientID = cr.ClientID

GROUP BY cs.ClientID;

MySQL:

SELECT \* FROM Cliental\_Portfolio;



**Query 3: Display\_Services**

Reason: Uses a view to show all of the sevices provided by the company

USE bharris\_cs355fa21;

CREATE OR REPLACE VIEW Display\_Services AS

(SELECT services

FROM CloudComputing)

UNION

(SELECT services

FROM Storage);

MySQL:

SELECT \* FROM Display\_Services;



**Query 4: Clients\_NoServices**

Reason: Uses a view to show every ClientID not tied to any service provided by the company

SELECT

cs.ClientID AS 'ClientID',

cs.ServiceID AS 'ServiceID'

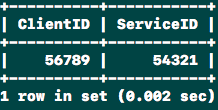
FROM Client\_Service AS cs

GROUP BY cs.ClientID

HAVING fn\_compCost(ServiceID) + fn\_storageCost(ServiceID) <= 0;

MySQL:

SELECT \* FROM Clients\_NoServices;



**Query 5: sp\_NewClient**

Reason: The Heart of the operation! Uses a procedure to create a new client within the database. Makes certain that the ClientID, ServiceID or Email are not already in use (Enforces PK Constraints). Determines the tables to populate data within and does so accordingly. First I use queries to show the difference made.

USE bharris\_cs355fa21;

DROP PROCEDURE IF EXISTS sp\_newClient;

DELIMITER //

CREATE PROCEDURE sp\_newClient

(

IN cID INT,

IN sID INT,

IN name VARCHAR (50),

IN servicePurchased VARCHAR(100),

IN cardInfo CHAR(4),

IN email VARCHAR(50),

OUT output VARCHAR(100)

)

BEGIN

DECLARE CUSTOM\_ERROR CONDITION FOR SQLSTATE '45000';

DECLARE clientFound BOOLEAN DEFAULT FALSE;

DECLARE emailFound BOOLEAN DEFAULT FALSE;

DECLARE serviceIDFound BOOLEAN DEFAULT FALSE;

DECLARE serviceFound BOOLEAN DEFAULT FALSE;

DECLARE storageCost DECIMAL (10,2);

DECLARE compCost DECIMAL (10,2);

-- Check if clientID exists

SET clientFound = (SELECT EXISTS(

SELECT ClientID

FROM Client\_Service

WHERE ClientID = cID

));

-- Make sure the ClientID does not exist (Enforce PK Constraint)

IF clientFound THEN

SET output = (SELECT CONCAT('ClientID belongs to: ',

(SELECT name FROM Client WHERE ClientID = cID)));

SIGNAL CUSTOM\_ERROR

SET MESSAGE\_TEXT = 'ClientID is unavailable';

END IF;

-- Make sure the ServiceID does not exist (Enforce PK Constraint)

SET serviceID Found = (SELECT EXISTS(

SELECT ServiceID

FROM Client\_Service

WHERE ServiceID = sID

));

IF serviceID Found THEN

SIGNAL CUSTOM\_ERROR

SET MESSAGE\_TEXT = 'ServiceID is unavailable';

END IF;

-- Check to see if the email is in use (Enforce Uniqueness Constraint)

SET emailFound = (SELECT EXISTS(

SELECT Email

FROM Client\_Email

WHERE ClientID = cID

));

IF emailFound THEN

SIGNAL CUSTOM\_ERROR

SET MESSAGE\_TEXT = 'Email is unavailable.';

END IF;

-- Create new user

INSERT INTO Client (ClientID, name, cardInfo) VALUES (cID, name, cardInfo);

INSERT INTO ServiceType VALUES (sID, TRUE, FALSE, FALSE);

INSERT INTO Client\_Email VALUES (cID, email);

INSERT INTO Client\_Service VALUES (cID, sID);

-- Insert Storage or Cloudcomputing service data

-- Make sure the Storage service exists in storage

SET serviceFound = (SELECT EXISTS(

SELECT services

FROM Storage

WHERE services LIKE servicePurchased

));

IF serviceFound THEN

SELECT DISTINCT cost

INTO storageCost

FROM Storage

WHERE services LIKE servicePurchased;

INSERT INTO Storage (ServiceID, services, cost) VALUES (sID, servicePurchased, storageCost);

END IF;

-- Make sure the Computing serice exists in CloudComputing

SET serviceFound = (SELECT EXISTS(

SELECT services

FROM CloudComputing

WHERE services LIKE servicePurchased

));

IF serviceFound THEN

SELECT DISTINCT cost

INTO compCost

FROM CloudComputing

WHERE services LIKE servicePurchased;

INSERT INTO CloudComputing (ServiceID, services, cost) VALUES (sID, servicePurchased, compCost);

END IF;

-- Paramaterized View! Would be super cool but cant use:

-- SET output = (SELECT \* FROM Cliental\_Portfolio WHERE ClientID = cID);

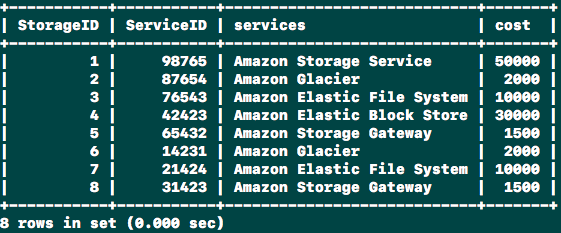
END //

DELIMITER ;

First Display all Storage services

MySQL:

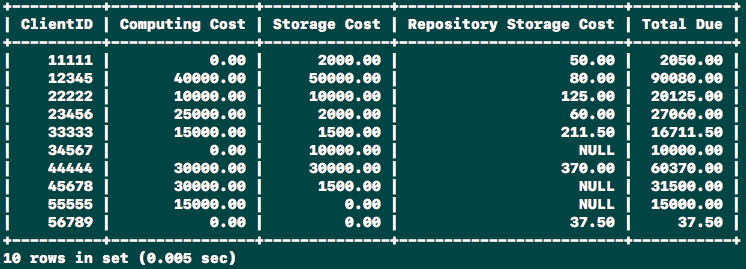
SELECT \* FROM Storage;



Display all Clients paying for services

MySQL:

SELECT \* FROM Cliental\_Portfolio;



AFTER PROCEDURE:

MySQL:

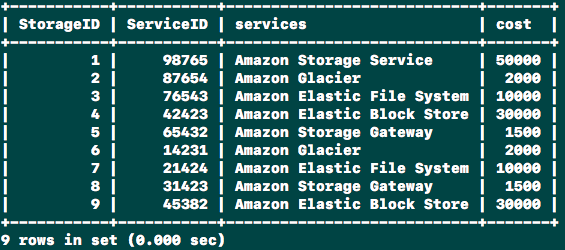
CALL sp\_NewClient(67891, 45382, 'Pete Moss', 'Amazon Elastic Block Store', '5382', 'Kickrocks@seawolf.edu', @out);

Query OK, 6 rows affected (0.002 sec)

ServiceID 45382 has been added and is paying for Amazon Elastic Block Store for 3000.

MySQL:

SELECT \* FROM Storage;



ClientID 67891 has been added and is paying 3000 for storage costs i.e.(The cost of Amazon Elastic Block Store, the service purchased)

MySQL:

SELECT \* FROM Cliental\_Portfolio;

