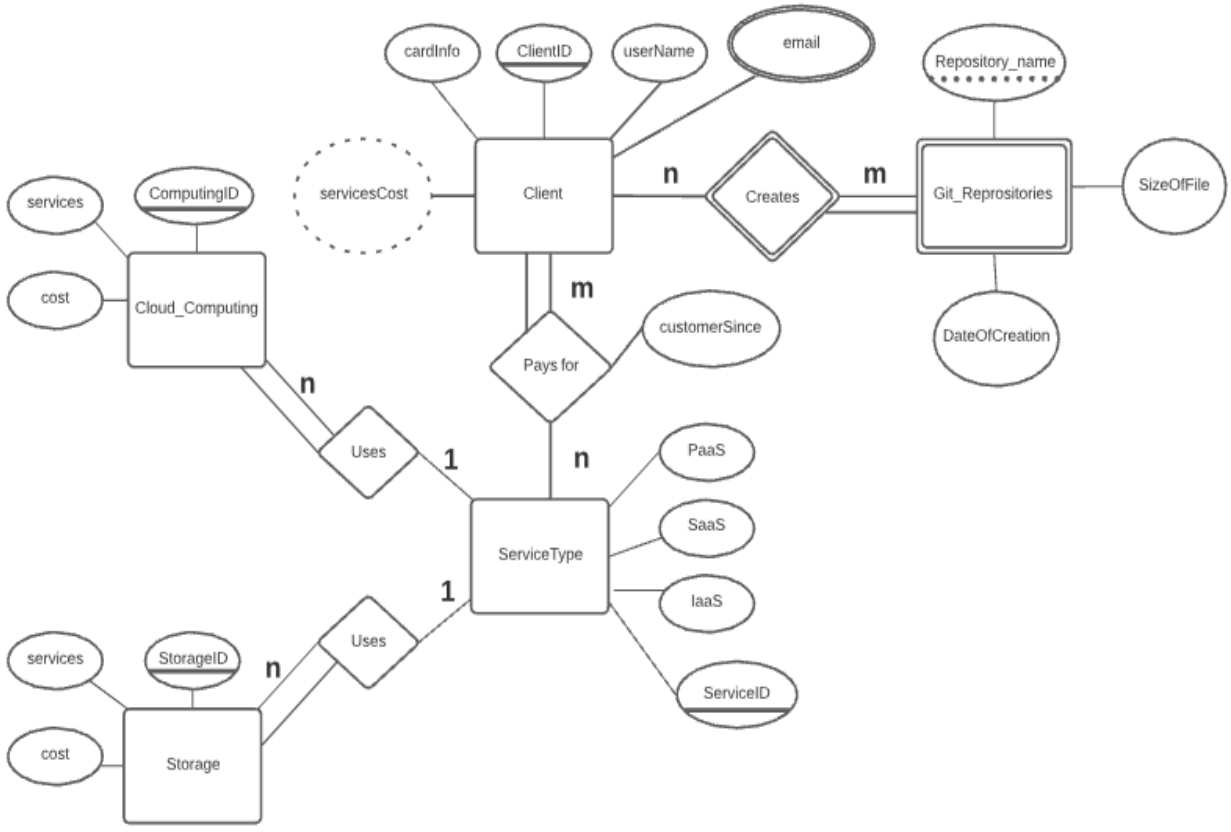
Benjamin Harris

AWS Database:

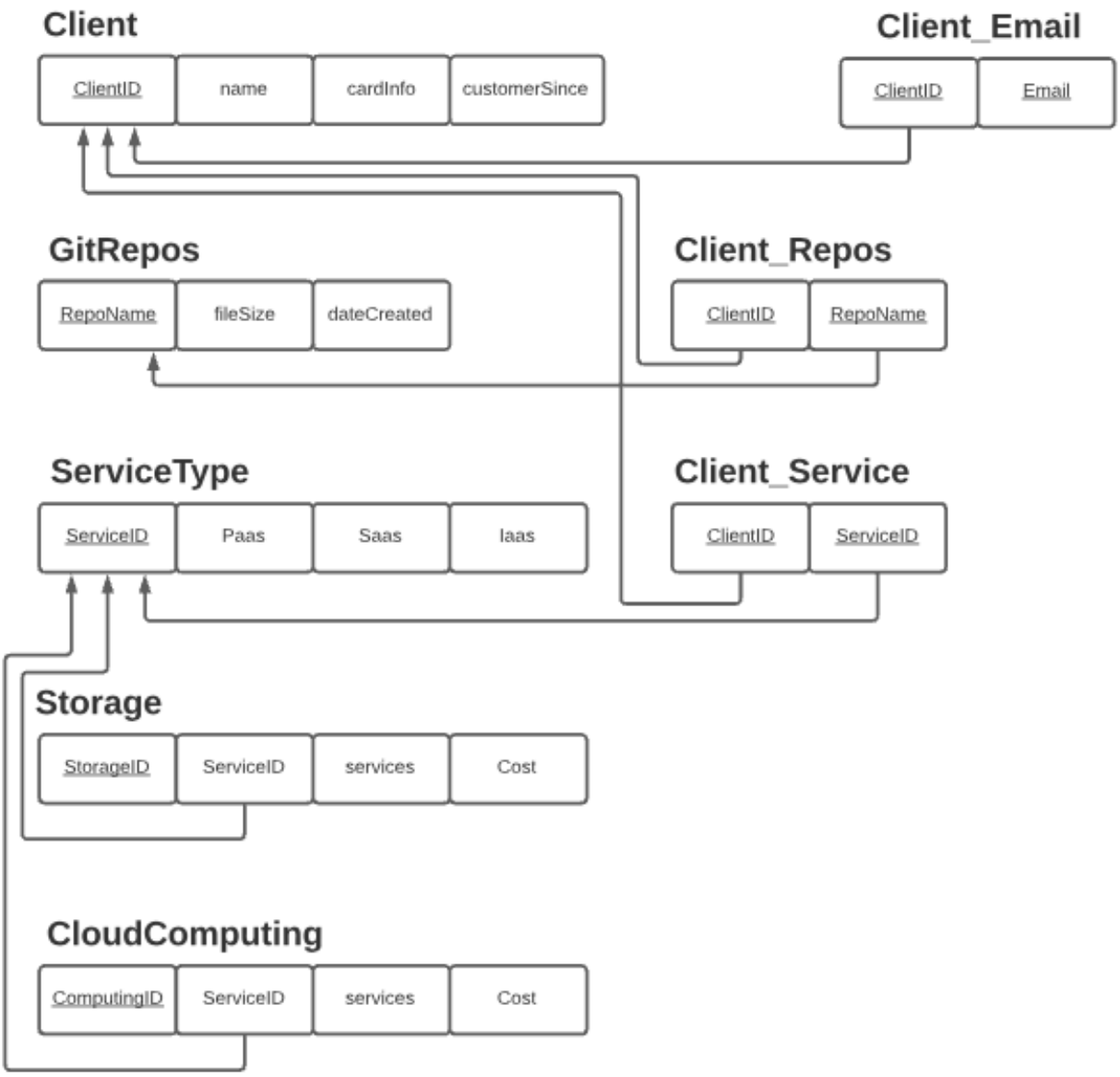
The Purpose of this Database is to provide an example of a smallscale implementation for Amazon Web Services. The database will track each Clients username, card info, email, service costs and unique ID. The database will also track repositories made by cliental. Each client may create zero or more Git-Repositories; 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



Benjamin Harris

Relational Schema Diagram



Benjamin Harris

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_AWS;
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;
```

ClientID	Name	Total Payment Due
11111	Costco	2050.00
12345	Benjamin Harris	90080.00
22222	Tesla	20125.00
23456	KrustyCrab Enterprises	27060.00
33333	FarmersUnited	16711.50
34567	PlasticsUnited	10000.00
44444	VA Hospital	60370.00
45678	Sarah Prestley	31500.00
55555	ChicoState	15000.00
56789	SonomaState	37.50

10 rows in set (0.003 sec)

Query 2: Clental_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_AWS;
CREATE OR REPLACE VIEW Clental_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;
```

ClientID	Computing Cost	Storage Cost	Repository Storage Cost	Total Due
11111	0.00	2000.00	50.00	2050.00
12345	40000.00	50000.00	80.00	90080.00
22222	10000.00	10000.00	125.00	20125.00
23456	25000.00	2000.00	60.00	27060.00
33333	15000.00	1500.00	211.50	16711.50
34567	0.00	10000.00	NULL	10000.00
44444	30000.00	30000.00	370.00	60370.00
45678	30000.00	1500.00	NULL	31500.00
55555	15000.00	0.00	NULL	15000.00
56789	0.00	0.00	37.50	37.50

10 rows in set (0.005 sec)

Query 3: Display_Services

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

```
USE bharris_AWS;
CREATE OR REPLACE VIEW Display_Services AS
(SELECT services
FROM CloudComputing)
UNION
(SELECT services
FROM Storage);
```

MySQL:

```
SELECT * FROM Display_Services;
```

```
+-----------------------------------+
| services                          |
+-----------------------------------+
| Amazon EC2 - Virtual Servers      |
| Amazon Lightsail - Manage Servers |
| Amazon VMware Cloud               |
| AWS Batch                         |
| Amazon Lamba                      |
| Amazon Storage Service            |
| Amazon Glacier                    |
| Amazon Elastic File System        |
| Amazon Elastic Block Store        |
| Amazon Storage Gateway            |
+-----------------------------------+
10 rows in set (0.001 sec)
```

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;
```

```
+----------+-----------+
| ClientID | ServiceID |
+----------+-----------+
|    56789 |     54321 |
+----------+-----------+
1 row in set (0.002 sec)
```

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_AWS;
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);
```

Benjamin Harris

```
 -- 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

Benjamin Harris

MySQL:

SELECT * FROM Storage;

StorageID	ServiceID	services	cost
1	98765	Amazon Storage Service	50000
2	87654	Amazon Glacier	2000
3	76543	Amazon Elastic File System	10000
4	42423	Amazon Elastic Block Store	30000
5	65432	Amazon Storage Gateway	1500
6	14231	Amazon Glacier	2000
7	21424	Amazon Elastic File System	10000
8	31423	Amazon Storage Gateway	1500

8 rows in set (0.000 sec)

Display all Clients paying for services

MySQL:

SELECT * FROM Cliental_Portfolio;

ClientID	Computing Cost	Storage Cost	Repository Storage Cost	Total Due
11111	0.00	2000.00	50.00	2050.00
12345	40000.00	50000.00	80.00	90080.00
22222	10000.00	10000.00	125.00	20125.00
23456	25000.00	2000.00	60.00	27060.00
33333	15000.00	1500.00	211.50	16711.50
34567	0.00	10000.00	NULL	10000.00
44444	30000.00	30000.00	370.00	60370.00
45678	30000.00	1500.00	NULL	31500.00
55555	15000.00	0.00	NULL	15000.00
56789	0.00	0.00	37.50	37.50

10 rows in set (0.005 sec)

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;

Benjamin Harris

StorageID	ServiceID	services	cost
1	98765	Amazon Storage Service	50000
2	87654	Amazon Glacier	2000
3	76543	Amazon Elastic File System	10000
4	42423	Amazon Elastic Block Store	30000
5	65432	Amazon Storage Gateway	1500
6	14231	Amazon Glacier	2000
7	21424	Amazon Elastic File System	10000
8	31423	Amazon Storage Gateway	1500
9	45382	Amazon Elastic Block Store	30000

9 rows in set (0.000 sec)

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;

ClientID	Computing Cost	Storage Cost	Repository Storage Cost	Total Due
11111	0.00	2000.00	50.00	2050.00
12345	40000.00	50000.00	80.00	90080.00
22222	10000.00	10000.00	125.00	20125.00
23456	25000.00	2000.00	60.00	27060.00
33333	15000.00	1500.00	211.50	16711.50
34567	0.00	10000.00	NULL	10000.00
44444	30000.00	30000.00	370.00	60370.00
45678	30000.00	1500.00	NULL	31500.00
55555	15000.00	0.00	NULL	15000.00
56789	0.00	0.00	37.50	37.50
67891	0.00	30000.00	NULL	30000.00

11 rows in set (0.005 sec)

Benjamin Harris