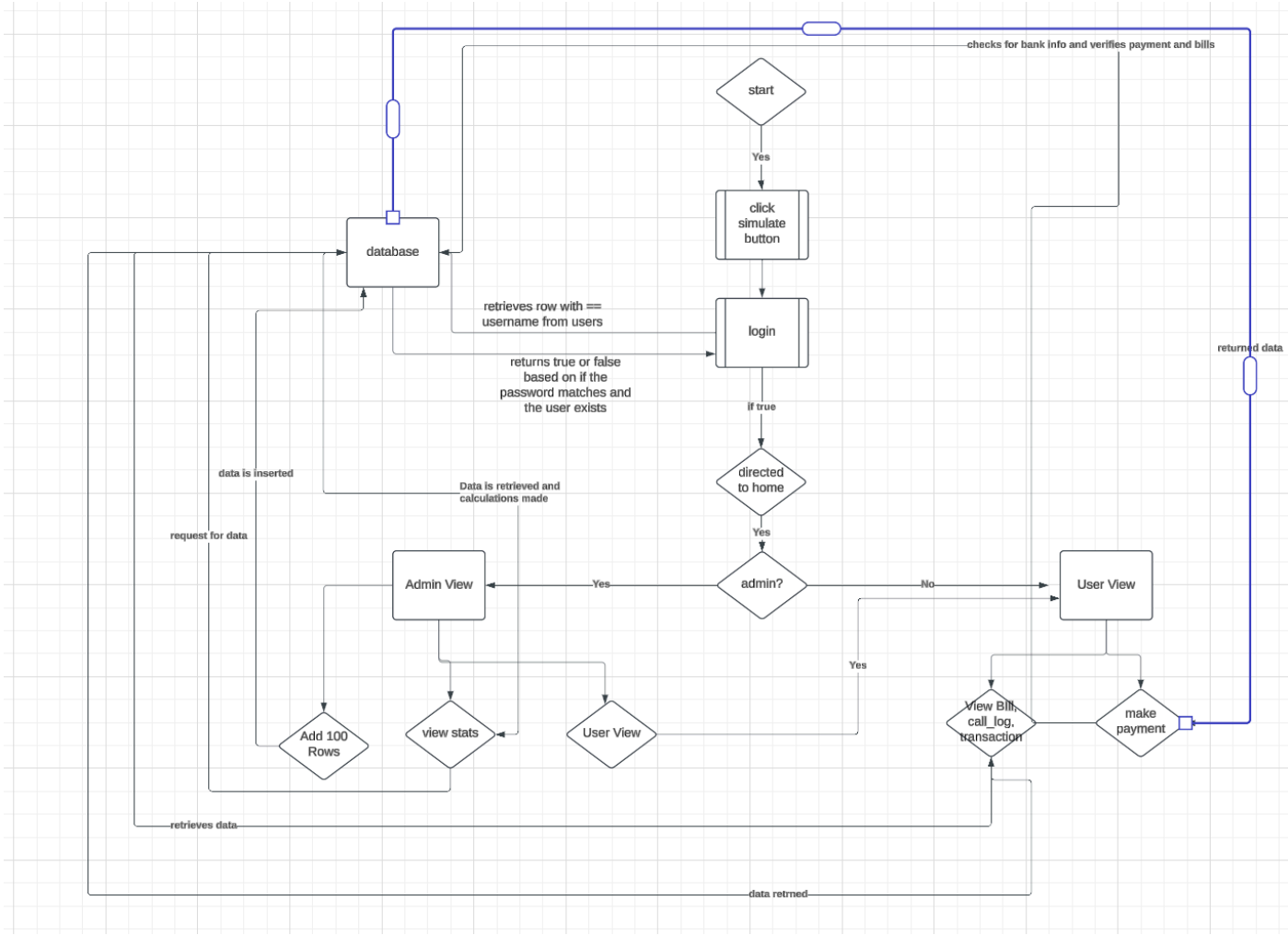
Cool Phone App

Team 05

University of Houston
COSC3320
Francis Generalli - PSID
Erin Bartels - PSID : 2060195
12/4/2024

The Cool Phone App

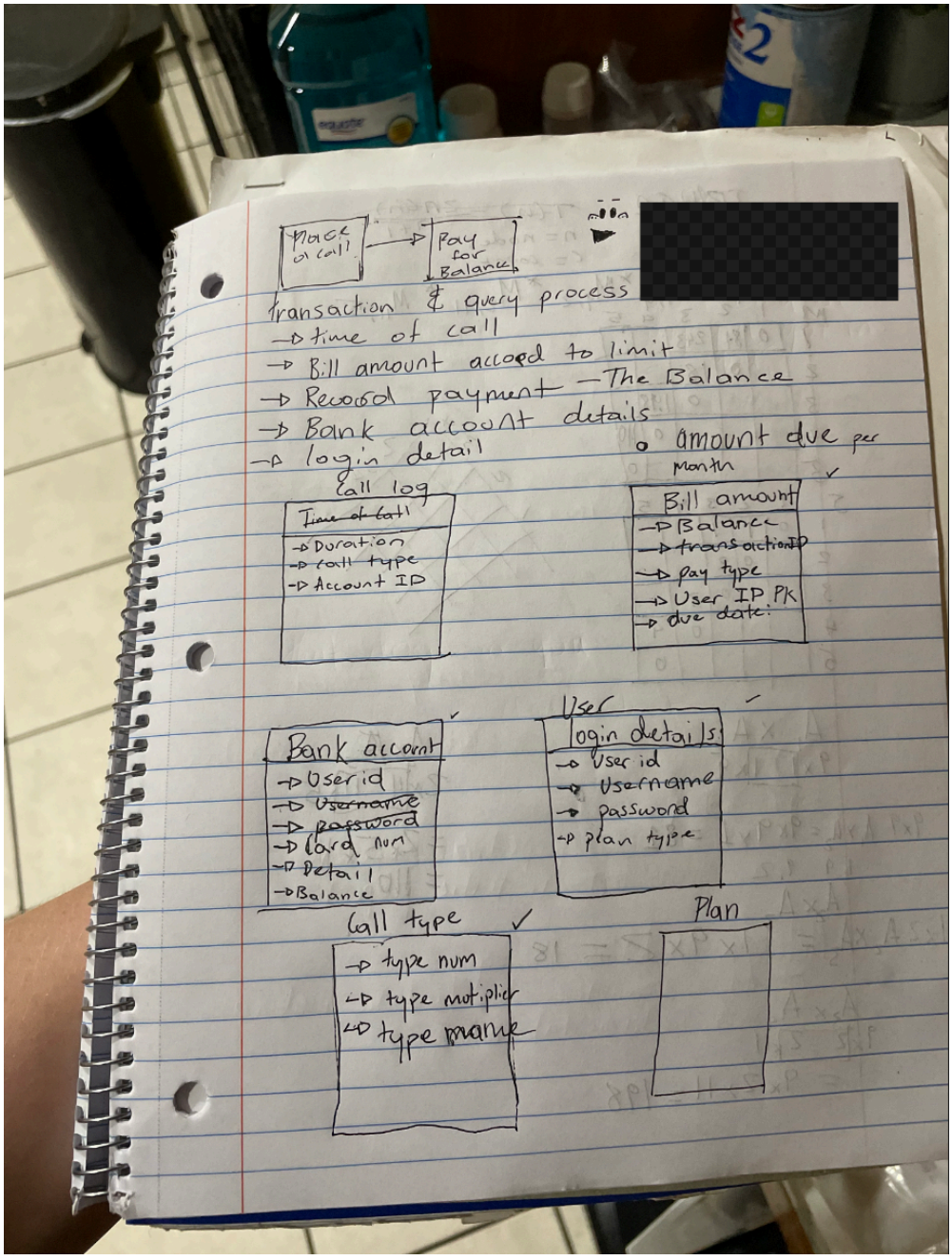
Flow Diagram



1. The user is taken to the login page where they should click the simulate button then login.
2. The login will be authenticated
3. If the User is an admin the will be sent to the regular admin view
4. An admin can add users and phone calls
5. View calculated stats on the customer base
6. Go to User View
7. In User view a person can see their call log, their data plan, their transaction history, and their due bills.
8. The user can click pay bills which will authenticate payment and update the database.

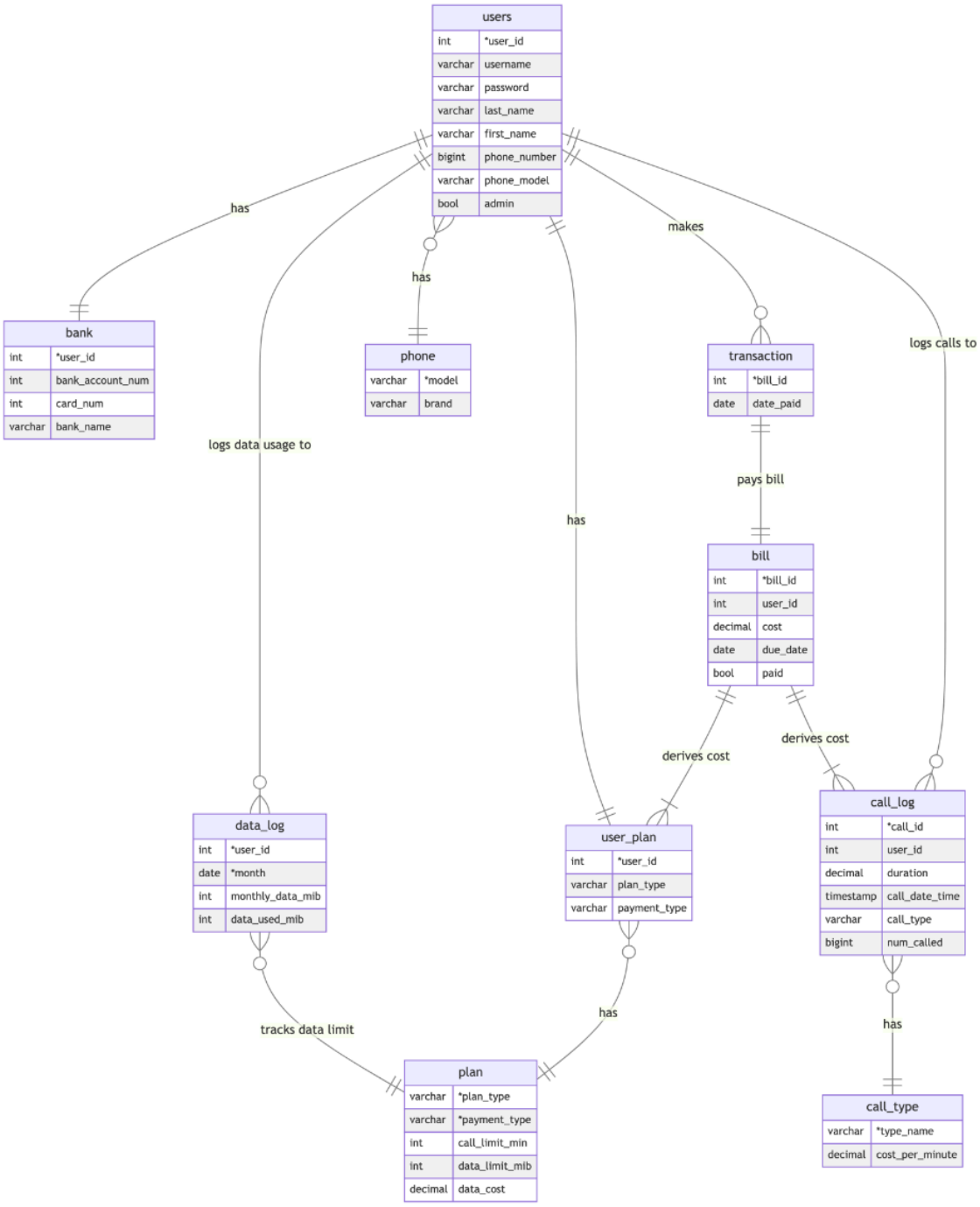
The Design

The Cool Phone App



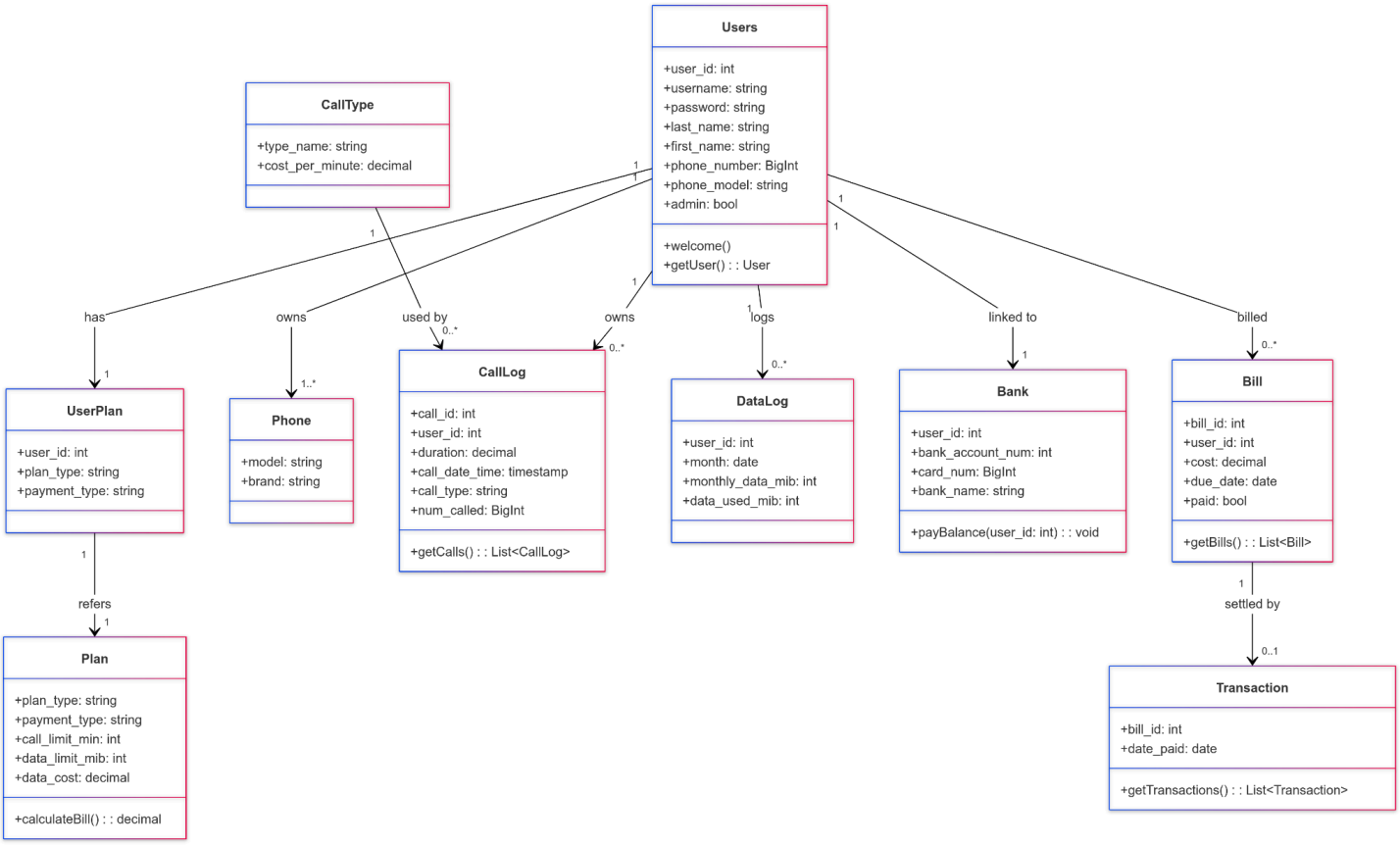
During this phase of planning we had mapped out five basic tables to meet the requirements for phase one. Five tables quickly proved to not be enough tables. As we started implementing this design we had to greatly re-adjust our tables. This diagram is rough and doesn't contain information about functions or relationships.

The Cool Phone App



This is the UML diagram after phase 1 was completed. As you can see above, when we finished Phase 1, we better understood how our program would function. Later, we decided that the data_log and plan did not have a direct relationship as there was no way to join them directly. We put the relationships on this diagram, but the diagram does not display cardinality or functions.

The Cool Phone App



For the final diagram, we added the functions and changed the way we displayed the relationships to display cardinality.

SQL

Bill log

```
BEGIN;



            SELECT
                b.bill_id AS bill_id,
                b.cost AS cost,
                b.due_date AS due_date,
                b.paid AS paid,
                COALESCE(SUM(c.duration), 0) AS total_minutes,
                COALESCE(SUM(d.data_used_mib), 0) AS data_used_mib
            FROM bill b
            LEFT JOIN call_log c ON c.user_id = b.user_id
                AND EXTRACT(YEAR FROM c.call_date_time) = EXTRACT(YEAR
FROM (b.due_date - INTERVAL '1 month'))
```

```
                AND EXTRACT(MONTH FROM c.call_date_time) = EXTRACT(MONTH
FROM (b.due_date - INTERVAL '1 month'))
            LEFT JOIN data_log d ON d.user_id = b.user_id
                AND EXTRACT(YEAR FROM d.month) = EXTRACT(YEAR FROM
(b.due_date - INTERVAL '1 month'))
                AND EXTRACT(MONTH FROM d.month) = EXTRACT(MONTH FROM
(b.due_date - INTERVAL '1 month'))
            WHERE b.user_id = 1
            GROUP BY b.bill_id
            ORDER BY b.bill_id DESC
            LIMIT 100
        ;

SELECT SUM(cost) AS total_due FROM bill WHERE user_id = 1 AND paid =
false;

COMMIT;
```

This query calculates the bill for the month by joining bill, data_log and call log.

The bill cost

```
BEGIN;



            SELECT
                t.bill_id AS bill_id,
                b.cost AS cost,
                b.due_date AS due_date,
                t.date_paid AS date_paid
            FROM transaction t
            JOIN bill b ON b.bill_id = t.bill_id
            WHERE b.user_id = 1
            ORDER BY t.bill_id DESC
            LIMIT 100
        ;

COMMIT;

BEGIN;

This SQL shows the bill history
```

The Cool Phone App

```
The SQL to retrieve the

            SELECT
                cl.call_id AS call_id,
                cl.call_date_time AS date,
                cl.duration AS duration,
                cl.call_type AS call_type,
                (cl.duration * ct.cost_per_minute) AS cost
            FROM call_log cl
            JOIN call_type ct ON ct.type_name = cl.call_type
            WHERE cl.user_id = 1
            ORDER BY cl.call_date_time DESC
            LIMIT 100
        ;

COMMIT;
```

The revenue

```
SELECT SUM(cost) AS total_revenue FROM bill WHERE paid = true
```

The average revenue per user

```
 WITH
                revenue_per_user AS (
                    SELECT user_id, SUM(cost) AS revenue
                    FROM bill
                    WHERE paid = true
                    GROUP BY user_id
                )
            SELECT AVG(revenue) AS avg_revenue
            FROM revenue_per_user
```

Finds the total money owed

```
SELECT SUM(cost) AS outstanding_bills FROM bill WHERE paid = false
```

Calculate percentage of local calls

```
SELECT SUM(duration) AS minutes FROM call_log
  WITH
                total_minutes AS (
                    SELECT SUM(duration) AS minutes FROM call_log
                ),
```

```
                local_minutes AS (
                    SELECT SUM(duration) AS minutes FROM call_log WHERE
call_type = 'Local'
                )
            SELECT (l.minutes / t.minutes * 100) AS percent
            FROM total_minutes t, local_minutes l
```

Calculate percentage of international calls

```
 WITH
                total_minutes AS (
                    SELECT SUM(duration) AS minutes FROM call_log
                ),
                national_minutes AS (
                    SELECT SUM(duration) AS minutes FROM call_log WHERE
call_type = 'National'
                )
            SELECT (n.minutes / t.minutes * 100) AS percent
            FROM total_minutes t, national_minutes n
        `;
```

Normalization

1. First Normal Form

- It contains only atomic (indivisible) values.
- Each record is unique.
- Each field contains only one value (no repeating groups or arrays).

  Therefore our project is in 1NF

2. Second Normal Form

- It is in 1NF.
- All non-key attributes are fully functionally dependent on the primary key (no partial dependency).

  Therefore the project is in 2NF

3. Third Normal Form

- It is in 2NF.

- It has no transitive dependencies (i.e., non-key attributes depend on other non-key attributes).

  Therefore it is in 3NF

4. Boyce-Codd Normal Form

- It is in 3NF.
- Every determinant is a candidate key (i.e., any attribute that determines another attribute must be a primary key or part of a candidate key).

  Therefore the project is in BCNF

5. Fourth Normal Form

- It is in BCNF.
- It has no multi-valued dependencies (i.e., a single record should not contain more than one fact).

  Therefore the project is in 5NF

Citations

CodeShack. (n.d.). Basic Login System in Node.js, Express, and MySQL. Retrieved from https://codeshack.io/basic-login-system-nodejs-express-mysql
Mermaid. (n.d.). Entity Relationship Diagram Syntax. Retrieved from https://mermaid.js.org/syntax/entityRelationshipDiagram.html#relationship-syntax
Google Docs. (2024). Node.js Project Documentation. Retrieved from https://docs.google.com/document/d/12geT2lJtT1K7FIhqqReMkTOxkihLHvHmaVOvZWgjitU/edit?tab=t.0#heading=h.70ypq3ivbm6q