HW2 - Phase 2

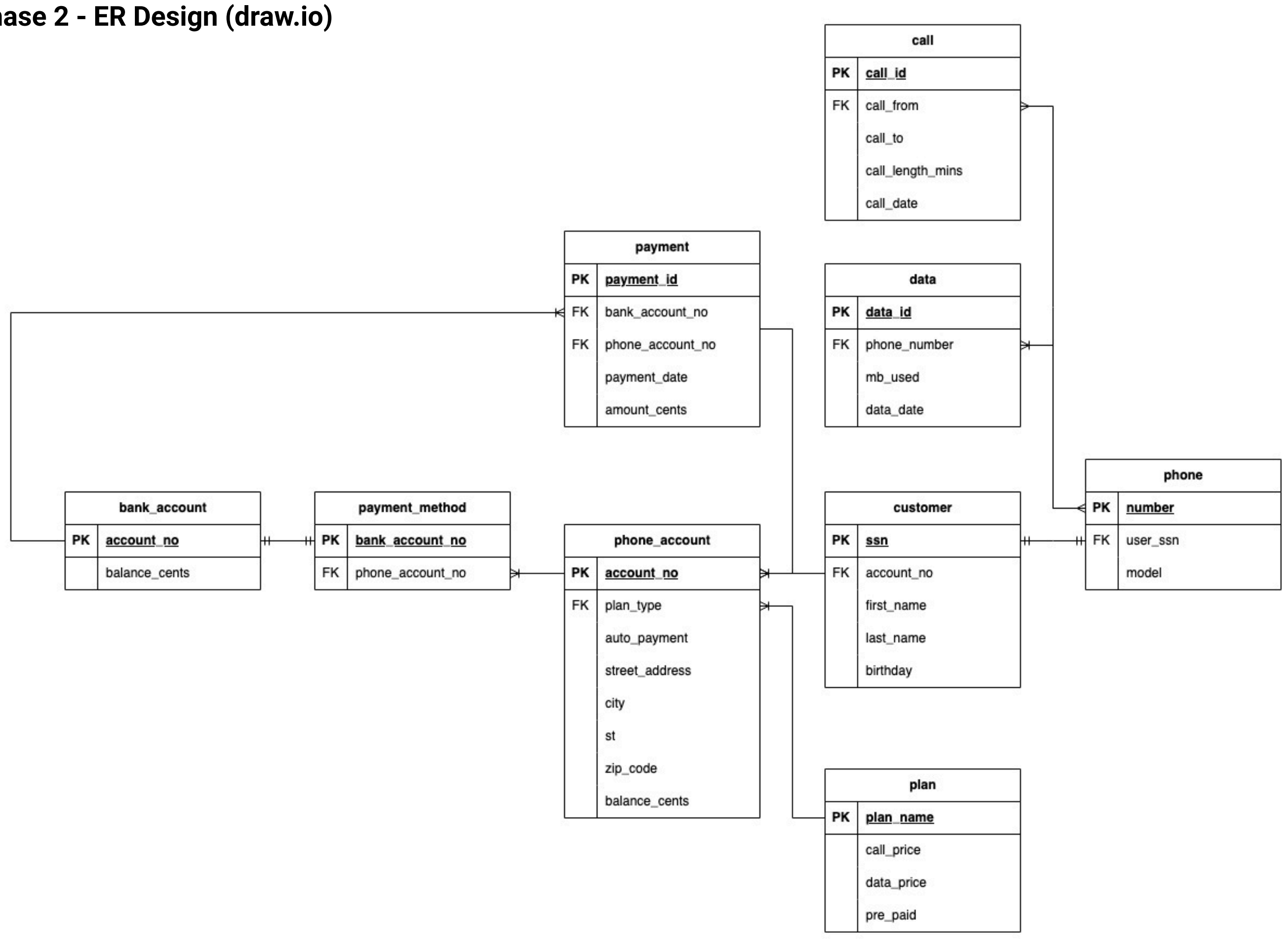
Team 07

Josh Eddie | Quoc Tan Nguyen (Bill)
Christine Ngo | Hassaan Haqqani

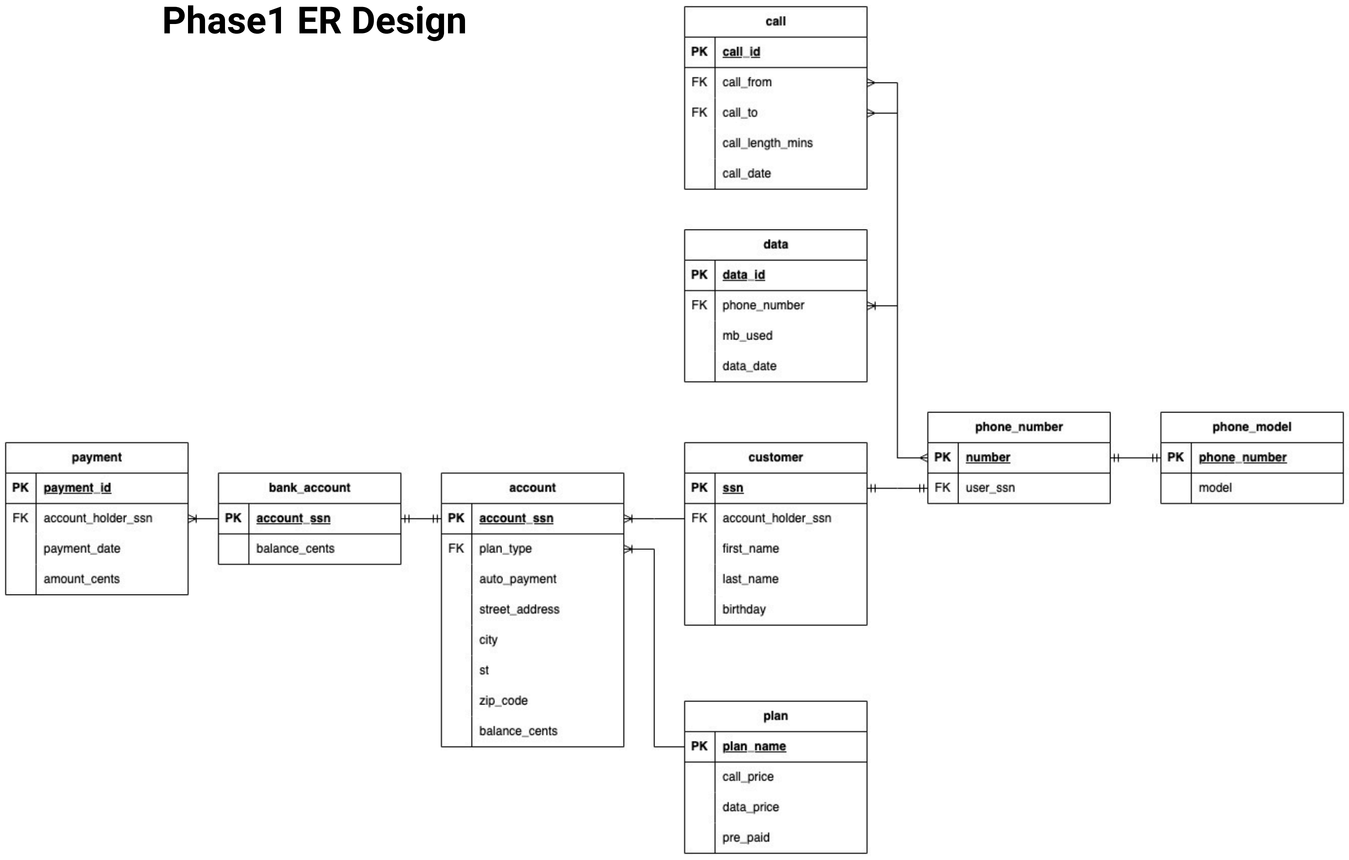
COSC 3380 - Database Systems
Carlos Ordonez
Nov. 22, 2023

ER Diagrams

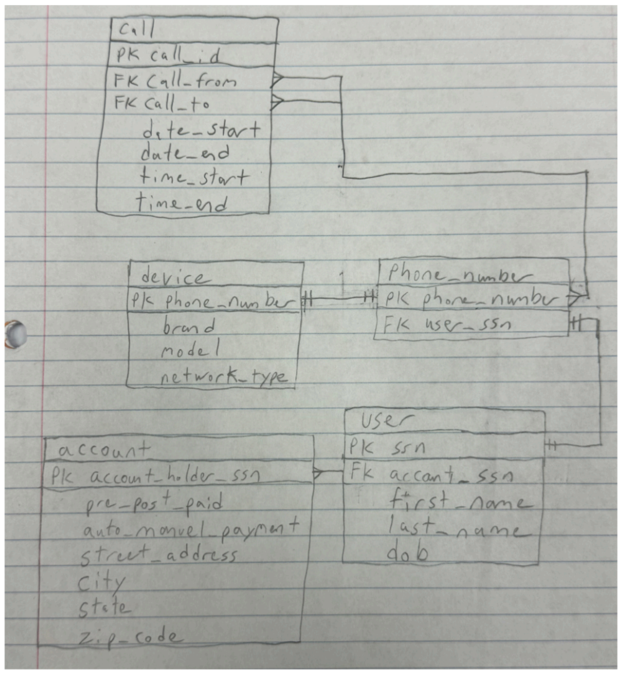
Phase 2 - ER Design (draw.io)



Phase1 ER Design



Preliminary ER Design



Second ER Design

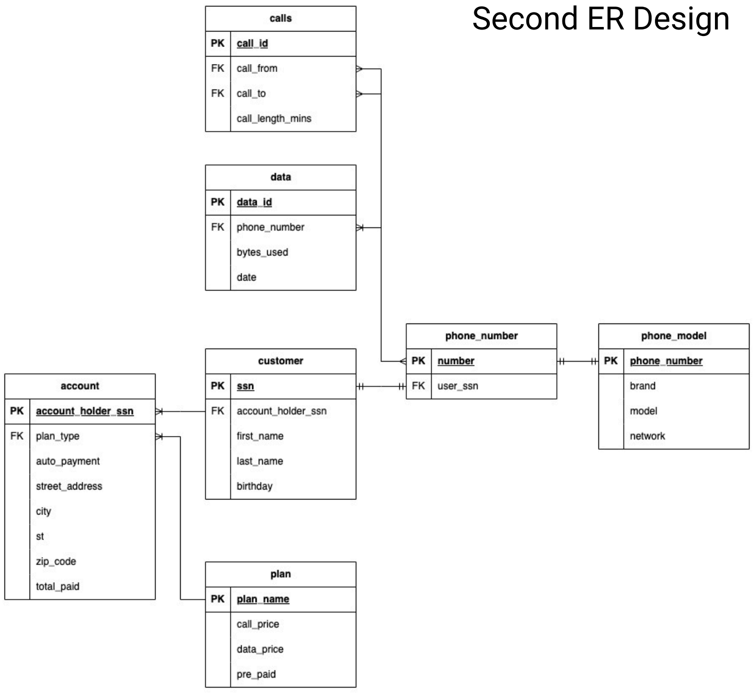


Table Description & Normalization

phone_account

Holds the primary information related to a phone account, address, plan type and account balance. Closely linked with customer. Normalized to 2NF, as zip code would be dependent on a composite of city/state breaking 3NF normalization. Maintaining a table of cities, states, and zip codes though is not desirable for this application.

customer

Holds detailed information on each customer, ssn, name, birthday and associated phone_account. Normalized to BCNF. Although, it is possible with a minimal dataset example set that there could be partial dependencies with name, and birthday. However, we would not expect name to be a unique identifiable with a large enough dataset.

phone

Holds the phone number and phone model of a customer. Linked via ssn. Normalized to 2NF. Phone model has a dependency on ssn violating 3NF. Originally, we maintained BCNF by having a separate table phone_model (as seen in phase1 ER diagram), but given that for each phone model there is only one phone number and one ssn these two tables would always have been of the same size and we would have always been duplicating phone number between two tables. This would have cost us time and space to maintain these two tables.

plan

Plan is the smallest table in our database. It holds the call and data prices of our different plan offerings. It is not modified at all by the application and is purely a lookup table. Normalized to BCNF.

call

Call holds the necessary information about a phone call. Call from, call to, call length, and a timestamp. Technically timestamp will likely cause the table to violate 3NF as it has a call id for a primary key and timestamp will likely be unique. Found multiple places online stating that using a timestamp as a primary key was a poor idea. Additionally, having a separate table comprised of just call id and timestamp would also be a bad idea, we chose to violate 3NF.

data

The counterpart to call, but for data. The same explanations on normalization apply to this table as well.

bank_account

A simple table holding an account number and a balance. Normalized to BCNF.

payment_method

Table to link bank account and phone account. This table would also allow for phone account to have multiple payment methods (although this ended up not getting implemented into the GUI). Normalized to BCNF.

payment

Holds information on payments made between bank and phone accounts. The timestamp presents the same problem here as on call and data technically making this table only normalized to 2NF.

Application Overview

The application is divided into two main sections, Network Admin and User Interface.

Network Admin

Upon starting the application for the first time, the user will be taken here. The only action that will be available is a button to initialize database. After this action is performed Network Admin allows a user to view the various tables in the database, delete data from them. Additionally, the entire database can be reinitialized or have all tables dropped.

Finally, there are some various reports that a user can access such as the high and low customer revenue months, highest grossing plans, most popular plan and phone models, and where our customers are located.

User Interface

User interface is the core of the application. Upon subsequent entries into the application it will be the starting point.

Account Search

Search bar that allows to search for an existing customer via either First Name, Last Name, Phone Number, or Account No. Clicking on a search result will take you into the account for that user.

Create Account

Create account is the secondary action that can be performed from the initial user interface page. Clicking on this button will take you to a form to fill in all necessary account details. This form will populate errors if data is missing or violates the database parameters. Upon finalizing the form and clicking create account the application will take you into the account page.

Account Overview

The account page has a brief info bar at the top showing the account number, address, and plan type. As well as links for Billing, Lines, Calls, Data Use, Edit Info and Sign Out.

Billing

Billing is the first page upon entering the account. It shows the account balance, bank account balance as well as a table with past payments made. User can also make a payment by typing in a value and clicking make payment button. This will immediately reflect in the two balances shown as well as an entry into the payment table.

Lines

Shows information for the lines/customers associated with this account. Clicking on an existing line allows information for that user to be edited. Additionally, a line can be added via the add line button.

Calls & Data Use

Calls & Data Use will show a table of calls and data uses made for each line on the account. There is a drop down menu that allows to filter between the various lines available on each account. There is also a button to simulate a phone call or data use that will be reflected in these tables as well as in the billing window.

Edit Info

Edit info allows for account details such as address and plan type to be changed.

Sign Out

Sign out will prompt a warning asking a user to confirm that they want to sign out.

Transaction SQL

Payment Transaction

```
BEGIN;

UPDATE phone_account
    SET balance_cents = balance_cents + 493
    WHERE account_no = '10000000'
    RETURNING (balance_cents / 100.00)::MONEY as balancedollar;

UPDATE bank_account
    SET balance_cents = balance_cents - 493
    WHERE account_no = '1234578'
    RETURNING (balance_cents / 100.00)::MONEY as balancedollar;

INSERT INTO payment (bank_account_no, phone_account_no, payment_date, amount_cents)
    VALUES ('1234578', '10000000', TIMESTAMP 'NOW()', -493);

COMMIT;

END;
```

Create Account Transaction

```
BEGIN;

INSERT INTO phone_account VALUES
    (nextval('phone_account_no_sequence'), 'Talker', 'false', '1234 Test Ln', 'Testing', 'TX', '98765')
    RETURNING account_no;

INSERT INTO bank_account VALUES
    (nextval('bank_account_no_sequence'), 45228);

INSERT INTO customer VALUES
    ('999999999', currval('phone_account_no_sequence'), 'Josh', 'Eddie', '11/20/2000');

INSERT INTO phone VALUES
    ('8121880814', '999999999', 'iPhone 15 Pro');

INSERT INTO payment_method VALUES
    (currval('bank_account_no_sequence'), currval('phone_account_no_sequence'));

INSERT INTO payment (bank_account_no, phone_account_no, payment_date, amount_cents)
    VALUES (currval('bank_account_no_sequence'), currval('phone_account_no_sequence'), NOW(), -3500);

COMMIT;

END;
```

Edit Account Transaction

```
BEGIN;

UPDATE phone_account
    SET street_address = '2858 Hillcrest Dr',
    city = 'Orange',
    st = 'TX',
    zip_code = '77976',
    plan_type = 'Want it All'
    WHERE account_no = '10000000';

COMMIT;

END;
```

Make Phone Call Transaction

```
BEGIN;

INSERT INTO call (call_from, call_to, call_length_mins, call_date)
    VALUES ('8367765983', '2508141538', 1, NOW());

UPDATE phone_account
    SET balance_cents = balance_cents - 1
    WHERE account_no = '10000000'
    RETURNING balance_cents;

COMMIT;

END;
```

Query SQL

Get Account Lines with calls and data use summed

```
WITH calls AS (
    SELECT number, model, first_name, last_name, TO_CHAR(SUM(COALESCE(call_length_mins, 0)), 'fm999G999') as
minutes
    FROM customer
    JOIN phone ON customer.ssn = phone.user_ssn
    LEFT JOIN call ON phone.number = call.call_from OR phone.number = call.call_to
    WHERE account_no = '10000000'
    GROUP BY number, first_name, last_name
),
dataUsed AS (
    SELECT phone_number, TO_CHAR(SUM(mb_used), 'fm999G999') as data_used
    FROM data
    GROUP BY phone_number
)
SELECT number, model, first_name, last_name, minutes, COALESCE(data_used, '0') as data_used
FROM calls
LEFT JOIN dataUsed ON calls.number = dataUsed.phone_number;
```

Get Phone and Bank Balances Query

```
SELECT (phone_account.balance_cents / 100.00)::MONEY as pbd, (bank_account.balance_cents / 100.00)::MONEY as bbd
    FROM phone_account
    JOIN payment_method ON phone_account.account_no = payment_method.phone_account_no
    JOIN bank_account ON payment_method.bank_account_no = bank_account.account_no
    WHERE phone_account.account_no = '10000000'
    GROUP BY pbd, bbd;
```

Gets Call for Particular number

```
SELECT call_from, call_to, call_length_mins as minutes, TO_CHAR(call_date,'MM-DD-YYYY') as Date, TO_CHAR(call_
date,'HH24:MI:SS') as Time
    FROM call
    JOIN phone ON phone.number = call.call_from OR phone.number = call.call_to
    WHERE phone.number = '8367765983'
    GROUP BY call_from, call_to, minutes, Date, Time
    ORDER BY date DESC, time DESC;
```

Customer Monthly Revenue

```
SELECT
    MAX(amount / 100.00)::MONEY AS customer_max_monthly_spend,
    MIN(amount / 100.00)::MONEY AS customer_min_monthly_spend,
    AVG(amount / 100.00)::MONEY AS customer_avg_monthly_spend
FROM (
    SELECT phone_account_no AS account_no, EXTRACT(MONTH FROM payment_date) AS month, EXTRACT(YEAR FROM payment_
date) AS year, SUM(amount_cents * -1) as amount
    FROM payment
    GROUP BY account_no, month, year
    ORDER BY amount DESC
) AS month_totals;
```

Plan Revenue

```
WITH revenue AS (
    SELECT plan_type, SUM(amount_cents * -1 / 100.00) AS total_revenue
    FROM payment
    JOIN phone_account ON payment.phone_account_no = phone_account.account_no
    GROUP BY plan_type
),
customer_count AS (
    SELECT plan_type, count(*) AS line_count
    FROM customer
    JOIN phone_account ON customer.account_no = phone_account.account_no
    GROUP BY plan_type
)
SELECT revenue.plan_type, revenue.total_revenue::MONEY, SUM(revenue.total_revenue / line_count)::MONEY AS avg_
per_customer
    FROM revenue
    JOIN customer_count ON revenue.plan_type = customer_count.plan_type
    GROUP BY revenue.plan_type, revenue.total_revenue
    ORDER BY total_revenue DESC;
```

Most Popular Plan

```
SELECT plan_type, COUNT(*) AS plan_count
    FROM phone_account
    GROUP BY plan_type
    ORDER BY plan_count DESC;
```

Sources

Create entity relationship diagrams with draw.io:

https://www.youtube.com/watch?v=JYZPdU5F2iMdraw.io

AT&T

https://www.att.com/support/

Billing and Payments: AT&T's system includes a billing and payment support section, where customers can learn how to view and pay their bills online, change their service, and more. In our Billing section, which details account balances, bank account balances, and a history of payments made.

React:

https://react.dev/

React Official Documentation: This is the best place to start for learning React. It covers all the basics, advanced topics, hooks, and more.

PostgreSQL:

https://node-postgres.com/

Node-postgres Documentation: This is the official documentation for using PostgreSQL in Node.js, which provides guides, API references, and examples.

Example Databases:

Pagila - https://github.com/devrimgunduz/pagila/

Chinook - https://github.com/lerocha/chinook-database/blob/master/ChinookDatabase/DataSources/Chinook_PostgreSql.sql

Credit

Josh Eddie

Web Application (react)

Server Application (API calls and transactions)

Quoc Tan Nguyen (Bill)

SQL Queries

References

Testing

Christine Ngo

SQL Queries

Testing

Hassaan Haqqani

ER Diagrams

SQL Queries

Testing