geo-distributed-billing

Георасределенный биллинг - система для управления балансами и транзакциями в георасределенной инфраструктуре.

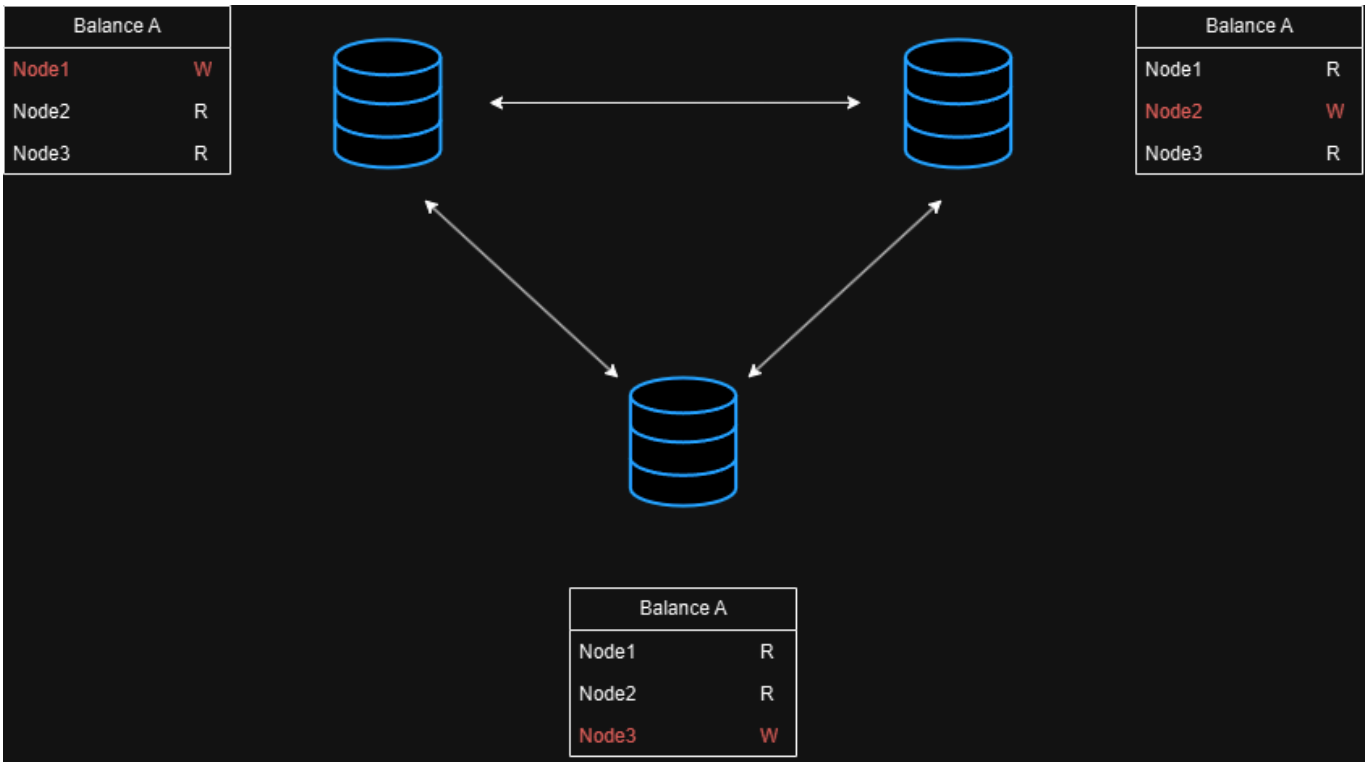
- **Аудитория**: Компании, использующие распределенные датацентры для обслуживания клиентов.
- **Использование**: Учет балансов, проведение транзакций.

Архитектура системы

Баланс пользователя делится на части. В каждом датацентре для каждого баланса хранится локальный баланс, который можно изменять (W), и данные баланса с других нод (R). Баланс пользователя - это суммарный баланс со всех ЦОДов.

При изменении локального баланса на определенной ноде, благодаря репликации баланс изменяется и на других датацентрах.

Такой подход позволяет ускорить время обработки каждого запроса, а в случае отключения датацентра система продолжит работать.



Требования

Функциональные

- Ведение учетных записей.
- Сохранение балансов аккаунтов в разных датацентрах.
- Синхронизация общих данных между датацентрами (информация об аккаунтах, услугах, ценах).
- Обработка Pay In транзакций в асинхронном режиме.

- Обработка Pay Out транзакций, включая синхронные междатацентровые переводы при нехватке средств.
- Ведение истории транзакций.

Нефункциональные

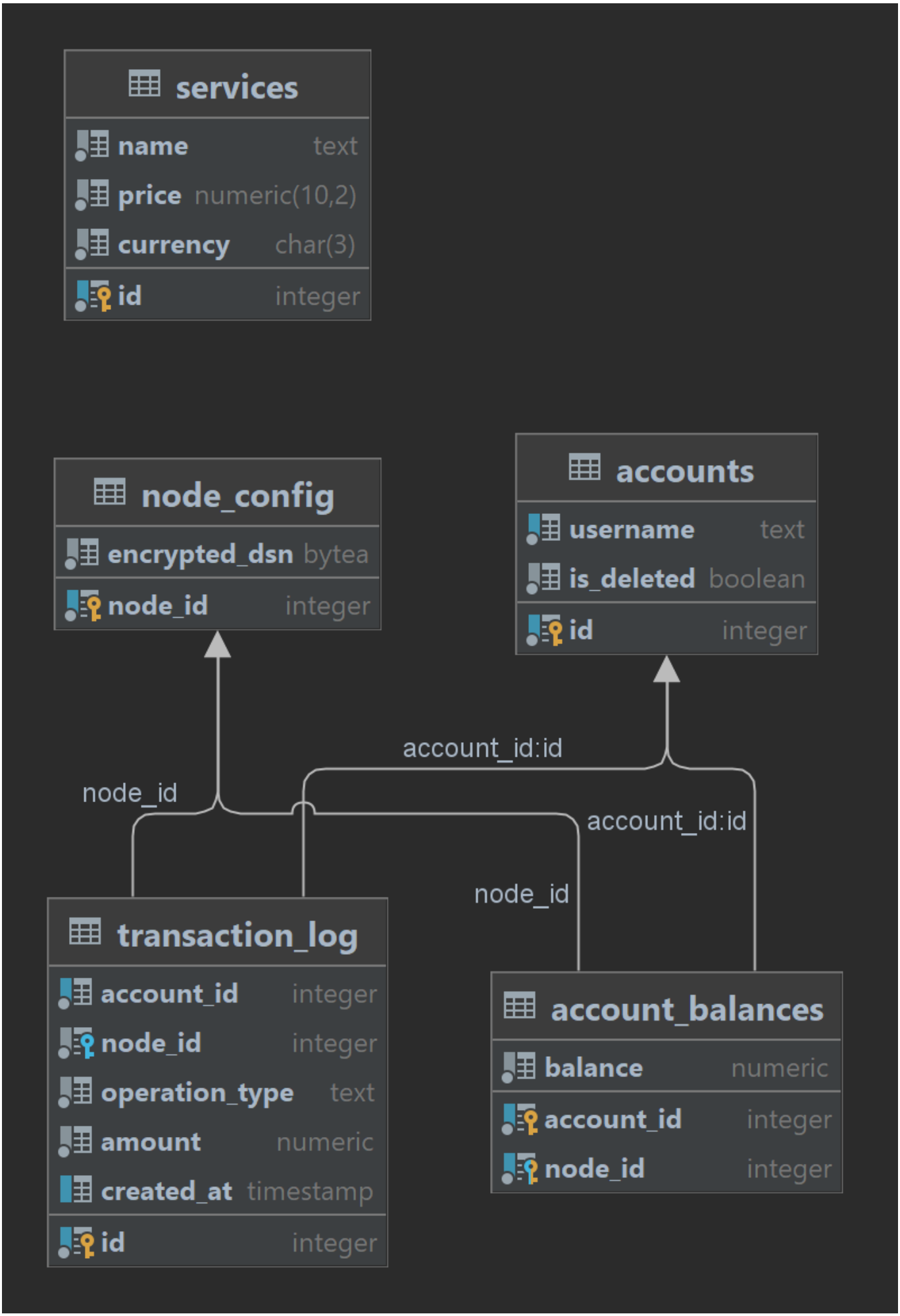
- Максимальная скорость ответа (это важно только для pay-in-транзакций).
- Синхронный контроль баланса (нужен контроль недопустимости отрицательного баланса).

Органичения на данные

- В каждом датацентре должно быть n записей с балансом, где n - число нод.
- Баланс аккаунта в любом датацентре не может быть отрицательным.

Схема БД

Каждый датацентр содержит данную схему БД:


services
name text
price numeric(10,2)
currency char(3)
id integer
node_config
encrypted_dsn bytea
node_id integer
accounts
username text
is_deleted boolean
id integer
account_id:id
node_id
account_id:id
node_id
transaction_log
account_id integer
node_id integer
operation_type text
amount numeric
created_at timestamp
id integer
account_balances
balance numeric
account_id integer
node_id integer

Авторы

- Александров Всеволод
- Павлычев Артемий
- Шалаев Алексей

Docs

SQL

В папке sql есть скрипт run_sql.sh, который запускает пайплайн, состоящий из stages. Каждый stage содержит общие скрипты, которые запускаются на всех датацентрах (папка all), и скрипты, которые запускаются на определенных нодах (папка node{id}, id - номер ноды). Спервa запускаются общие скрипты, а потом для каждой ноды.

```
#!/bin/bash

# Настройки подключения
declare -A nodes=(
    [node1]="postgres1 5432 admin password billing"
    [node2]="postgres2 5432 admin password billing"
    [node3]="postgres3 5432 admin password billing"
)

# Функция для выполнения SQL-файлов
execute_sql() {
    local node=$1
    local dsn=($2)  # Разбиваем строку подключения в массив
    local dir=$3

    local host=${dsn[0]}
    local port=${dsn[1]}
    local user=${dsn[2]}
    local password=${dsn[3]}
    local dbname=${dsn[4]}

    echo "Executing SQL scripts in $dir for $node..."
    for sql_file in "$dir"/*.sql** \; do
        if [[ -f "$sql_file" ]]; then
            echo "Running $sql_file on $node..."
            PGPASSWORD=$password psql -h $host -p $port -U $user -d $dbname -f
"$sql_file" \
                || { echo "Error executing $sql_file on $node. Exiting."; exit 1;
}
        fi
    done
}

for stage in stage1 stage2 stage3; do

    # Выполнение общих скриптов (all) на всех нодах
```

```
    if [[ -d "./$stage/all" ]]; then
        for node in "${!nodes[@]}"; do
            echo "Executing all scripts on $node..."
            execute_sql "$node" "${nodes[$node]}" "./$stage/all"
        done
    fi

    # Выполнение уникальных скриптов для каждой ноды в фиксированном порядке
    for node in node1 node2 node3; do
        if [[ -d "./$stage/$node" ]]; then
            echo "Executing unique scripts on $node..."
            execute_sql "$node" "${nodes[$node]}" "./$stage/$node"
        fi
    done

done

echo "All SQL scripts executed successfully."
```

Stage 1

All

- **00_install_extensions.sql**
  Установка расширений.

```
-- Установим расширения, если они еще не установлены
CREATE EXTENSION IF NOT EXISTS pglogical;
CREATE EXTENSION IF NOT EXISTS dblink;
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

- **01_create_tables.sql**
  Создание таблиц. Таблица services не имеет связей, так как она может использоваться сервисом по определению стоимости услуги.

```
-- Переключение на базу данных billing
\c billing

-- Создание таблицы accounts
CREATE TABLE accounts (
    id SERIAL PRIMARY KEY,
    username TEXT UNIQUE NOT NULL,
    is_deleted BOOLEAN NOT NULL DEFAULT FALSE
);

-- Создание таблицы services
CREATE TABLE IF NOT EXISTS services (
    id SERIAL PRIMARY KEY,                              -- Уникальный
```

```
идентификатор услуги
    name TEXT NOT NULL,                                  -- Название услуги
    price NUMERIC(10, 2) NOT NULL CHECK (price >= 0),    -- Цена услуги
    currency CHAR(3) NOT NULL                            -- Валюта услуги
(например, USD, EUR)
);

CREATE TABLE account_balances (
    account_id INT NOT NULL,            -- ID аккаунта
    node_id INT NOT NULL,               -- ID узла
    balance NUMERIC NOT NULL,           -- Баланс аккаунта
    PRIMARY KEY (account_id, node_id)   -- Составной первичный ключ
);

CREATE TABLE node_config (
    node_id INT PRIMARY KEY,            -- ID ноды
    encrypted_dsn BYTEA NOT NULL        -- Зашифрованная строка подключения
);

CREATE TABLE transaction_log (
    id SERIAL PRIMARY KEY,              -- Уникальный идентификатор записи
    account_id INT NOT NULL,            -- ID аккаунта
    node_id INT NOT NULL,               -- Узел, инициировавший операцию
    -- Тип операции ('deposit' / 'withdraw')
    operation_type TEXT NOT NULL CHECK (operation_type IN ('deposit',
'withdraw')),
    amount NUMERIC NOT NULL,            -- Сумма операции
    created_at TIMESTAMP DEFAULT NOW()  -- Время операции
);

-- Foreign key constraints

ALTER TABLE account_balances
ADD CONSTRAINT fk_account_balances_node
FOREIGN KEY (node_id) REFERENCES node_config (node_id)
ON DELETE RESTRICT;

ALTER TABLE transaction_log
ADD CONSTRAINT fk_transaction_log_node
FOREIGN KEY (node_id) REFERENCES node_config (node_id)
ON DELETE RESTRICT;
```

- **02_get_encryption_key.sql**
  Получение секретного ключа для расшифровки данных.

```
CREATE OR REPLACE FUNCTION get_encryption_key()
RETURNS TEXT AS $$
BEGIN
    RETURN pg_read_file('/pgcrypto_key.txt');
END;
```

```
$$ LANGUAGE plpgsql;
```

- **03_log_transaction.sql**
  Функция для логирования транзакций.

```
CREATE OR REPLACE FUNCTION log_transaction(
    p_account_id INT,
    p_node_id INT,
    p_operation_type TEXT,
    p_amount NUMERIC DEFAULT NULL
)
RETURNS VOID AS $$
BEGIN
    -- Проверка наличия записи о балансе
    IF NOT EXISTS (
        SELECT 1 FROM account_balances
        WHERE account_id = p_account_id AND node_id = p_node_id
    ) THEN
        RAISE EXCEPTION 'No balance record found for account % on node %',
p_account_id, p_node_id;
    END IF;

    -- Запись информации о транзакции в лог
    INSERT INTO transaction_log (account_id, node_id, operation_type, amount)
    VALUES (p_account_id, p_node_id, p_operation_type, p_amount);

    RAISE NOTICE 'Transaction logged: account_id=%, node_id=%, operation_type=%,
amount=%.',
                 p_account_id, p_node_id, p_operation_type, p_amount;
END;
$$ LANGUAGE plpgsql;
```

- **04_node_helpers.sql**
  Набор функций для работы с другими нодами.

```
-- Эта функция получает расшифрованную строку подключения для заданной ноды
CREATE OR REPLACE FUNCTION get_decrypted_dsn(p_node_id INT)
RETURNS TEXT AS $$
BEGIN
    RETURN (
        SELECT pgp_sym_decrypt(encrypted_dsn::bytea, get_encryption_key())
        FROM node_config
        WHERE node_id = p_node_id
    );
END;
$$ LANGUAGE plpgsql;
```

```
-- Эта функция устанавливает соединение с указанной нодой и возвращает имя
соединения
CREATE OR REPLACE FUNCTION connect_to_node(p_node_id INT)
RETURNS TEXT AS $$
DECLARE
    conn_name TEXT := 'conn_' || p_node_id;
    dsn TEXT;
BEGIN
    dsn := get_decrypted_dsn(p_node_id);

    IF dsn IS NULL OR TRIM(dsn) = '' THEN
        RAISE EXCEPTION 'Decrypted DSN for node % is invalid.', p_node_id;
    END IF;

    -- Отключение существующего соединения с таким же именем, если оно есть
    BEGIN
        PERFORM dblink_disconnect(conn_name);
    EXCEPTION WHEN OTHERS THEN
        -- Игнорировать ошибку, если соединение не существует
    END;

    PERFORM dblink_connect(conn_name, dsn);
    RAISE NOTICE 'Connected to node % with connection name %.', p_node_id,
conn_name;
    RETURN conn_name;
EXCEPTION WHEN OTHERS THEN
    RAISE EXCEPTION 'Failed to connect to node %. Details: %', p_node_id, SQLERRM;
END;
$$ LANGUAGE plpgsql;

-- Эта функция отключает соединение с заданной нодой
CREATE OR REPLACE FUNCTION disconnect_from_node(p_conn_name TEXT)
RETURNS VOID AS $$
BEGIN
    PERFORM dblink_disconnect(p_conn_name);
    RAISE NOTICE 'Disconnected from connection %.', p_conn_name;
EXCEPTION WHEN OTHERS THEN
    RAISE NOTICE 'Failed to disconnect from connection %.', p_conn_name;
END;
$$ LANGUAGE plpgsql;

-- Эта функция выполняет заданный SQL-запрос на удалённой ноде через dblink
CREATE OR REPLACE FUNCTION execute_remote_query(p_conn_name TEXT, p_query TEXT)
RETURNS VOID AS $$
BEGIN
    PERFORM dblink_exec(p_conn_name, p_query);
EXCEPTION WHEN OTHERS THEN
    RAISE EXCEPTION 'Failed to execute remote query on connection %. Details: %',
p_conn_name, SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

- **05_add_account.sql**
  Функция создания аккаунта.

```
CREATE OR REPLACE FUNCTION add_account(
    p_username TEXT            -- Имя пользователя
)
RETURNS INT AS $$
DECLARE
    new_account_id INT;
BEGIN
    -- Проверка имени пользователя
    IF p_username IS NULL OR TRIM(p_username) = '' THEN
        RAISE EXCEPTION 'Username cannot be null or empty.';
    END IF;

    -- Создание нового аккаунта на локальном узле
    INSERT INTO accounts (username)
    VALUES (p_username)
    RETURNING id INTO new_account_id;

    -- Проверка, что аккаунт был создан
    IF NOT FOUND THEN
        RAISE EXCEPTION 'Account with username % already exists.', p_username;
    END IF;

    RAISE NOTICE 'Account was added successfully with ID %.',
                 new_account_id;

    RETURN new_account_id;

EXCEPTION WHEN OTHERS THEN
    RAISE EXCEPTION 'Transaction failed: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

- **06_add_account_balance.sql**
  Создание баланса пользователя. Для каждой ноды создается соединение и начинается транзакция, которая создает баланс, в случае ошибок все транзакции откатываются. Баланс делится на целые части.

```
CREATE OR REPLACE FUNCTION add_account_balance(
    account_id INT,
    total_balance NUMERIC,
    local_node_id INT
)
RETURNS VOID AS $$
DECLARE
    node RECORD;                -- Для итерации по узлам
```

```
    node_count INT;              -- Общее количество нод
    distributed_balance NUMERIC; -- Равномерный баланс для каждой ноды
    remainder NUMERIC;           -- Остаток, который будет добавлен одной ноде
    is_remainder_distributed BOOLEAN := FALSE; -- Флаг, указывает, был ли
распределён остаток
    decrypted_dsn TEXT;          -- Расшифрованная строка подключения
    connections TEXT[] := ARRAY[]::TEXT[]; -- Список активных подключений
BEGIN
    -- Проверка наличия аккаунта с указанным account_id
    IF NOT EXISTS (
        SELECT 1 FROM accounts WHERE id = account_id
    ) THEN
        RAISE EXCEPTION 'Account with ID % does not exist.', account_id;
    END IF;

    -- Подсчитать количество нод
    SELECT COUNT(*) INTO node_count FROM node_config;

    IF node_count = 0 THEN
        RAISE EXCEPTION 'No nodes found in node_config.';
    END IF;

    -- Равномерное распределение баланса
    distributed_balance := FLOOR(total_balance / node_count);
    remainder := total_balance - (distributed_balance * node_count);

    -- Отладочный вывод: количество нод
    RAISE NOTICE 'Number of nodes: %', node_count;

    -- Начинаем транзакцию на всех нодах
    FOR node IN SELECT node_id, pgp_sym_decrypt(encrypted_dsn,
get_encryption_key()) AS decrypted_dsn FROM node_config LOOP
        BEGIN
            -- Вывод информации о текущей ноде
            RAISE NOTICE 'Connecting to node %', node.node_id;

            -- Устанавливаем соединение через dblink
            PERFORM dblink_connect('conn_' || node.node_id, node.decrypted_dsn);
            connections := array_append(connections, 'conn_' || node.node_id);

            -- Открываем транзакцию
            PERFORM dblink_exec('conn_' || node.node_id, 'BEGIN');
        EXCEPTION WHEN OTHERS THEN
            RAISE EXCEPTION 'Failed to connect or start transaction on node %.',
node.node_id;
        END;
    END LOOP;

    -- Вставляем данные
    FOR node IN SELECT node_id, pgp_sym_decrypt(encrypted_dsn,
get_encryption_key()) AS decrypted_dsn FROM node_config LOOP
        BEGIN
            IF node.node_id = local_node_id THEN
                -- Добавляем запись на локальную ноду
```

```
                INSERT INTO account_balances (account_id, node_id, balance)
                VALUES (
                    account_id,
                    node.node_id,
                    distributed_balance + CASE
                        WHEN NOT is_remainder_distributed THEN remainder
                        ELSE 0
                    END
                );

                -- Отмечаем, что остаток распределён
                is_remainder_distributed := TRUE;
            ELSE
                -- Добавляем запись на удалённую ноду через dblink
                PERFORM dblink_exec(
                    'conn_' || node.node_id,
                    'INSERT INTO account_balances (account_id, node_id, balance) '
||
                    'VALUES (' || account_id || ', ' || node.node_id || ', ' ||
                    distributed_balance + CASE
                        WHEN NOT is_remainder_distributed THEN remainder
                        ELSE 0
                    END || ');'
                );

                -- Отмечаем, что остаток распределён
                is_remainder_distributed := TRUE;
            END IF;
        EXCEPTION WHEN OTHERS THEN
            -- В случае ошибки откатываем транзакцию
            FOR i IN 1..array_length(connections, 1) LOOP
                PERFORM dblink_exec(connections[i], 'ROLLBACK');
                PERFORM dblink_disconnect(connections[i]);
            END LOOP;
            RAISE EXCEPTION 'Failed to insert account on node %. Rolling back all
changes.', node.node_id;
        END;
    END LOOP;

    -- Завершаем транзакцию на всех нодах
    FOR i IN 1..array_length(connections, 1) LOOP
        BEGIN
            PERFORM dblink_exec(connections[i], 'COMMIT');
            PERFORM dblink_disconnect(connections[i]);
        EXCEPTION WHEN OTHERS THEN
            RAISE NOTICE 'Failed to commit or disconnect on connection %.',
connections[i];
        END;
    END LOOP;

    RAISE NOTICE 'Account % successfully added with total balance %.', account_id,
total_balance;

EXCEPTION WHEN OTHERS THEN
```

```
    -- Откат транзакции на всех нодах в случае общей ошибки
    FOR i IN 1..array_length(connections, 1) LOOP
        PERFORM dblink_exec(connections[i], 'ROLLBACK');
        PERFORM dblink_disconnect(connections[i]);
    END LOOP;
    RAISE EXCEPTION 'Transaction failed: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

- **07_delete_account.sql**
  Мягкое удаление аккаунта.

```
CREATE OR REPLACE FUNCTION delete_account(p_account_id INT)
RETURNS VOID AS $$
BEGIN
    -- Проверяем, что аккаунт существует и не удалён
    IF EXISTS (
        SELECT 1 FROM accounts
        WHERE id = p_account_id AND is_deleted = FALSE
    ) THEN
        -- Помечаем аккаунт как удалённый
        UPDATE accounts
        SET is_deleted = TRUE
        WHERE id = p_account_id;

        RAISE NOTICE 'Account % has been marked as deleted.', p_account_id;
    ELSE
        RAISE EXCEPTION 'Account % does not exist or is already deleted.',
p_account_id;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

- **08_deposit_balance.sql**
  Пополнение баланса. Когда пользователь делает запрос на пополнение, его обрабатывает ближайший датацентр. После изменения локального баланса на ноде, изменения доедут в асинхронном режиме до других датацентров.

```
CREATE OR REPLACE FUNCTION deposit_balance(p_account_id INT, p_amount NUMERIC,
p_node_id INT)
RETURNS VOID AS $$
BEGIN
    -- Проверяем, что запись о балансе существует
    IF NOT EXISTS (
        SELECT 1 FROM account_balances
        WHERE account_id = p_account_id AND node_id = p_node_id
    ) THEN
        RAISE EXCEPTION 'No balance record found for account % on node %',
p_account_id, p_node_id;
```

```
    END IF;

    -- Проверяем, что сумма положительна
    IF p_amount <= 0 THEN
        RAISE EXCEPTION 'Deposit amount must be positive';
    END IF;

    -- Увеличиваем баланс для текущей ноды
    UPDATE account_balances
    SET balance = balance + p_amount
    WHERE account_id = p_account_id
      AND node_id = p_node_id;

    -- Логируем транзакцию
    PERFORM log_transaction(p_account_id, p_node_id, 'deposit', p_amount);

    -- Репликация выполнится автоматически через pglogical
    RAISE NOTICE 'Balance updated locally on node % and changes will replicate
asynchronously.', p_node_id;
END;
$$ LANGUAGE plpgsql;
```

- **09_withdraw_balance.sql**
  Уменьшение баланса. В рамках конкретного датацентра мы просто контролируем уровень локального баланса, чтобы он не ушел в минус. Если в датацентре на локальном балансе хватает денежных средств, то с локального баланса списываются средства (информация об измении доедет до других датацентров с помощью асинхронной репликации). Если средств не хватает, то суммируем все средста. Если сумма не позволяет провести транзакцию, то отклоняем запрос, иначе обнуляем запрос на данном датацентре и снимаем в синхронном режиме средвста с других нод.

```
CREATE OR REPLACE FUNCTION withdraw_balance(p_account_id INT, p_amount NUMERIC,
p_node_id INT)
RETURNS VOID AS $$
DECLARE
    local_balance NUMERIC;                -- Локальный баланс
    total_balance NUMERIC;                -- Суммарный баланс по всем нодам
    needed_balance NUMERIC;               -- Необходимая сумма для снятия
    node_balances RECORD;                 -- Запись для итерации по другим узлам
    remote_dsn TEXT;                      -- Расшифрованная строка подключения к
узлу
BEGIN
    -- Проверяем, что запись о балансе существует
    IF NOT EXISTS (
        SELECT 1 FROM account_balances
        WHERE account_id = p_account_id AND node_id = p_node_id
    ) THEN
        RAISE EXCEPTION 'No balance record found for account % on node %',
p_account_id, p_node_id;
    END IF;
```

```
    -- Проверяем, что сумма положительна
    IF p_amount <= 0 THEN
        RAISE EXCEPTION 'Withdrawal amount must be positive';
    END IF;

    -- Получаем локальный баланс
    SELECT balance INTO local_balance
    FROM account_balances
    WHERE account_balances.account_id = p_account_id
      AND account_balances.node_id = p_node_id;

    -- Проверяем локальный баланс
    IF local_balance >= p_amount THEN
        -- Если хватает, снимаем локально
        UPDATE account_balances
        SET balance = balance - p_amount
        WHERE account_balances.account_id = p_account_id
          AND account_balances.node_id = p_node_id;

        -- Логируем транзакцию
        PERFORM log_transaction(p_account_id, p_node_id, 'withdraw', p_amount);

        RAISE NOTICE 'Withdrawn % from local balance on node %. Replication will
sync changes.', p_amount, p_node_id;
        RETURN;
    END IF;

    -- Если не хватает локально, проверяем суммарный баланс
    SELECT SUM(balance) INTO total_balance
    FROM account_balances
    WHERE account_balances.account_id = p_account_id;

    -- Если суммарного баланса недостаточно
    IF total_balance < p_amount THEN
        RAISE EXCEPTION 'Insufficient funds';
    END IF;

    -- Определяем необходимую сумму для снятия
    needed_balance := p_amount - local_balance;

    -- Обнуляем локальный баланс
    UPDATE account_balances
    SET balance = 0
    WHERE account_balances.account_id = p_account_id
      AND account_balances.node_id = p_node_id;

    -- Итерация по другим узлам и отправка запросов
    FOR node_balances IN
        SELECT node_id, balance
        FROM account_balances
        WHERE account_balances.account_id = p_account_id
          AND account_balances.node_id <> p_node_id
    LOOP
```

```
        -- Расшифровываем строку подключения для узла
        SELECT pgp_sym_decrypt(node_config.encrypted_dsn::bytea,
get_encryption_key()) INTO remote_dsn
        FROM node_config
        WHERE node_config.node_id = node_balances.node_id;

        -- Рассчитываем долю для снятия
        PERFORM dblink_exec(
            remote_dsn,
            'UPDATE account_balances ' ||
            'SET balance = balance - ' || needed_balance * (node_balances.balance
/ (total_balance - local_balance)) ||
            ' WHERE account_id = ' || p_account_id || ' AND node_id = ' ||
node_balances.node_id || ';'
        );

        RAISE NOTICE 'Sent withdraw request to node %.', node_balances.node_id;
    END LOOP;

    -- Логируем транзакцию после завершения
    PERFORM log_transaction(p_account_id, p_node_id, 'withdraw', p_amount);

    RAISE NOTICE 'Withdrawn %. Requests sent to other nodes.', p_amount;

END;
$$ LANGUAGE plpgsql;
```

- **10_create_triggers.sql**
  Триггеры.

```
-- Запрет на отрицательный баланс
CREATE OR REPLACE FUNCTION prevent_negative_balance()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.balance < 0 THEN
        RAISE EXCEPTION 'Balance cannot be negative for account % on node %',
NEW.account_id, NEW.node_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_prevent_negative_balance
BEFORE INSERT OR UPDATE ON account_balances
FOR EACH ROW EXECUTE FUNCTION prevent_negative_balance();

-- Запрет на прямое удаление аккаунта
CREATE OR REPLACE FUNCTION prevent_account_deletion()
RETURNS TRIGGER AS $$
BEGIN
    RAISE EXCEPTION 'Direct deletion from accounts is not allowed. Use the
```

```
delete_account function.';
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_prevent_account_deletion
BEFORE DELETE ON accounts
FOR EACH ROW
EXECUTE FUNCTION prevent_account_deletion();

-- Запрет на операции с удалёнными аккаунтами
CREATE OR REPLACE FUNCTION prevent_operations_on_deleted_accounts()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM accounts
        WHERE id = NEW.account_id AND is_deleted = TRUE
    ) THEN
        RAISE EXCEPTION 'Cannot perform operation on deleted account %.',
NEW.account_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_prevent_operations_on_deleted_accounts_account_balances
BEFORE INSERT OR UPDATE ON account_balances
FOR EACH ROW
EXECUTE FUNCTION prevent_operations_on_deleted_accounts();

CREATE TRIGGER trg_prevent_operations_on_deleted_accounts_transaction_log
BEFORE INSERT OR UPDATE ON transaction_log
FOR EACH ROW
EXECUTE FUNCTION prevent_operations_on_deleted_accounts();
```

- **11_create_indexes.sql**
  Индексы.

```
-- Таблица account_balances
CREATE INDEX idx_account_balances_account_id ON account_balances (account_id);

-- Таблица transaction_log
CREATE INDEX idx_transaction_log_account_id ON transaction_log (account_id);
CREATE INDEX idx_transaction_log_created_at ON transaction_log (created_at);
```

- **12_insert_nodes.sql**
  Добавление зашифрованной информации о датацентрах.

```
INSERT INTO node_config (node_id, encrypted_dsn)
VALUES
    (1, pgp_sym_encrypt('host=postgres1 dbname=billing user=admin
password=password', get_encryption_key())),
    (2, pgp_sym_encrypt('host=postgres2 dbname=billing user=admin
password=password', get_encryption_key())),
    (3, pgp_sym_encrypt('host=postgres3 dbname=billing user=admin
password=password', get_encryption_key()));
```

Node 1

- **00_setup_replication.sql**
  Настройка репликации на ноде 1.

```
-- Создадим узел логической репликации
SELECT pglogical.create_node(
    node_name := 'node1',
    dsn := 'host=postgres1 dbname=billing user=admin password=password'
);

-- Настроим публикацию для общих таблиц
SELECT pglogical.create_replication_set(
    set_name := 'node1_set',
    replicate_insert := true,
    replicate_update := true,
    replicate_delete := true,
    replicate_truncate := false
);

SELECT pglogical.replication_set_add_table(
    set_name := 'node1_set',
    relation := 'account_balances',
    row_filter := 'node_id = 1' -- Только строки с node_id = 1
);

SELECT pglogical.replication_set_add_table(
    set_name := 'node1_set',
    relation := 'node_config'
);

SELECT pglogical.replication_set_add_table(
    set_name := 'node1_set',
    relation := 'services'
);

SELECT pglogical.replication_set_add_table(
    set_name := 'node1_set',
    relation := 'accounts'
);
```

Node 2

- **00_setup_replication.sql**
  Настройка репликации на ноде 2.

```
-- Создадим узел логической репликации
SELECT pglogical.create_node(
    node_name := 'node2',
    dsn := 'host=postgres2 dbname=billing user=admin password=password'
);

-- Настроим публикацию для общих таблиц
SELECT pglogical.create_replication_set(
    set_name := 'node2_set',
    replicate_insert := true,
    replicate_update := true,
    replicate_delete := true,
    replicate_truncate := false
);

SELECT pglogical.replication_set_add_table(
    set_name := 'node2_set',
    relation := 'account_balances',
    row_filter := 'node_id = 2'
);
```

Node 3

- **00_setup_replication.sql**
  Настройка репликации на ноде 3.

```
-- Создадим узел логической репликации
SELECT pglogical.create_node(
    node_name := 'node3',
    dsn := 'host=postgres3 dbname=billing user=admin password=password'
);

-- Настроим публикацию для общих таблиц
SELECT pglogical.create_replication_set(
    set_name := 'node3_set',
    replicate_insert := true,
    replicate_update := true,
    replicate_delete := true,
    replicate_truncate := false
);

SELECT pglogical.replication_set_add_table(
    set_name := 'node3_set',
    relation := 'account_balances',
    row_filter := 'node_id = 3'
```

```
);
```

Stage 2

Node 1

- **00_setup_subscription.sql**
  Настройка подписок ноды 1.

```
SELECT pglogical.create_subscription(
    subscription_name := 'subscription_from_node2_to_node1',
    provider_dsn := 'host=postgres2 dbname=billing user=admin password=password',
    replication_sets := ARRAY['node2_set']
);

SELECT pglogical.create_subscription(
    subscription_name := 'subscription_from_node3_to_node1',
    provider_dsn := 'host=postgres3 dbname=billing user=admin password=password',
    replication_sets := ARRAY['node3_set']
);
```

Node 2

- **00_setup_subscription.sql**
  Настройка подписок ноды 2.

```
SELECT pglogical.create_subscription(
    subscription_name := 'subscription_from_node1_to_node2',
    provider_dsn := 'host=postgres1 dbname=billing user=admin password=password',
    replication_sets := ARRAY['node1_set']
);

SELECT pglogical.create_subscription(
    subscription_name := 'subscription_from_node3_to_node2',
    provider_dsn := 'host=postgres3 dbname=billing user=admin password=password',
    replication_sets := ARRAY['node3_set']
);
```

Node 3

- **00_setup_subscription.sql**
  Настройка подписок ноды 3.

```
SELECT pglogical.create_subscription(
    subscription_name := 'subscription_from_node1_to_node3',
```

```
    provider_dsn := 'host=postgres1 dbname=billing user=admin password=password',
    replication_sets := ARRAY['node1_set']
);

SELECT pglogical.create_subscription(
    subscription_name := 'subscription_from_node2_to_node3',
    provider_dsn := 'host=postgres2 dbname=billing user=admin password=password',
    replication_sets := ARRAY['node2_set']
);
```

Stage 3

Node 1

- **00_insert_services.sql**
  Вставка данных на ноду 1. На остальных нодах данные появятся благодаря логической репликации.

```
-- Переключение на базу данных billing
\c billing

-- Пример вставки данных в таблицу
INSERT INTO services (name, price, currency)
VALUES
    ('Консультация специалиста', 50.00, 'USD'),
    ('Аренда оборудования', 100.00, 'EUR'),
    ('Обучение персонала', 200.00, 'USD');
```

- **01_insert_accounts.sql**
  Вставка данных на ноду 1. На остальных нодах данные появятся благодаря логической репликации.

```
-- Переключение на базу данных billing
\c billing

SELECT add_account_balance(add_account('Aleksandrov'), 1000, 1);
SELECT add_account_balance(add_account('Pavlichev'), 2000, 1);
SELECT add_account_balance(add_account('Shalaev'), 3000, 1);
```

Docker

Данный docker-compose.yml является примеров системы из 3 датацентров.

```
services:
  psql-client:
    image: postgres:17
```

```
    container_name: psql-client
    networks:
      - postgres_net
    working_dir: /sql
    entrypoint: ["./run_sql.sh"]
    volumes:
      - ./sql:/sql:ro
    depends_on:
      postgres1:
        condition: service_healthy
      postgres2:
        condition: service_healthy
      postgres3:
        condition: service_healthy

  postgres1:
    image: postgres:17
    container_name: postgres1
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password
      POSTGRES_DB: billing
      DB_HOST: postgres1
    volumes:
      - ./pgcrypto_key.txt:/pgcrypto_key.txt:ro
    #  - ./data/postgres1:/var/lib/postgresql/data
    ports:
      - "15432:5432"
    networks:
      - postgres_net
    command: >
      bash -c "apt-get update && apt-get install -y postgresql-17-pglogical &&
               docker-entrypoint.sh postgres -c shared_preload_libraries=pglogical
-c wal_level=logical"
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready", "-d", "billing"]
      interval: 10s
      timeout: 3s
      retries: 3

  postgres2:
    image: postgres:17
    container_name: postgres2
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password
      POSTGRES_DB: billing
      DB_HOST: postgres2
    volumes:
      - ./pgcrypto_key.txt:/pgcrypto_key.txt:ro
    #  - ./data/postgres2:/var/lib/postgresql/data
    ports:
      - "15433:5432"
    networks:
```

```
      - postgres_net
    command: >
      bash -c "apt-get update && apt-get install -y postgresql-17-pglogical &&
               docker-entrypoint.sh postgres -c shared_preload_libraries=pglogical
-c wal_level=logical"
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready", "-d", "billing"]
      interval: 10s
      timeout: 3s
      retries: 3

  postgres3:
    image: postgres:17
    container_name: postgres3
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password
      POSTGRES_DB: billing
      DB_HOST: postgres3
    volumes:
      - ./pgcrypto_key.txt:/pgcrypto_key.txt:ro
    #  - ./data/postgres3:/var/lib/postgresql/data
    ports:
      - "15434:5432"
    networks:
      - postgres_net
    command: >
      bash -c "apt-get update && apt-get install -y postgresql-17-pglogical &&
               docker-entrypoint.sh postgres -c shared_preload_libraries=pglogical
-c wal_level=logical"
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready", "-d", "billing"]
      interval: 10s
      timeout: 3s
      retries: 3

networks:
  postgres_net:
    driver: bridge
```