System Zarządzania Korporacją X - Dokumentacja

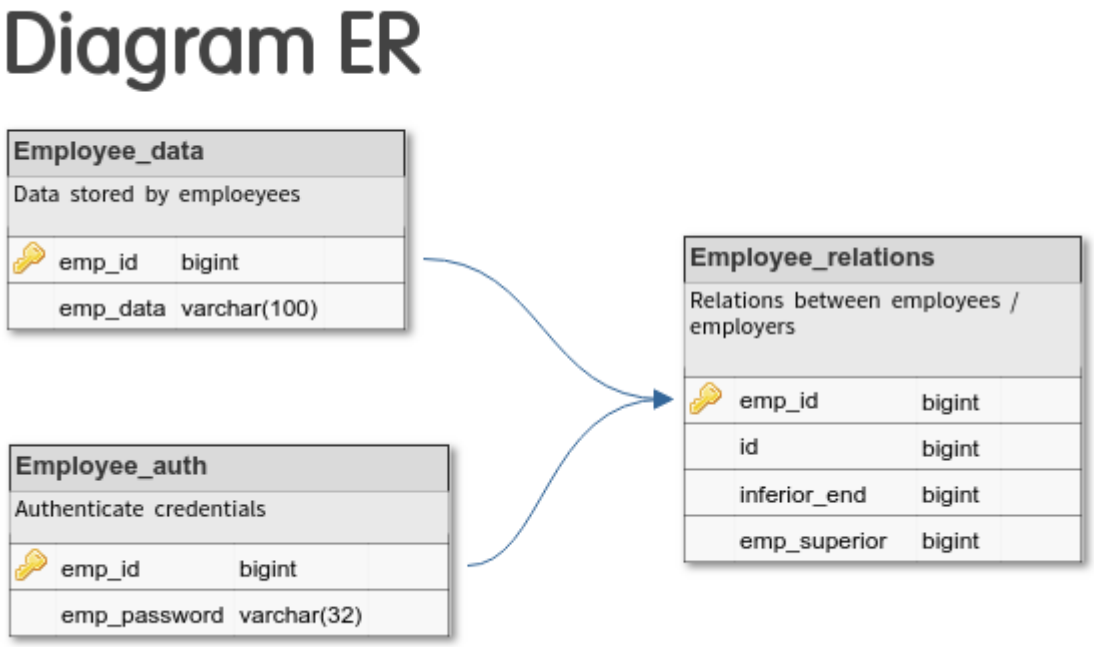
Jan Góra (272472), 15 czerwca 2018r.

grupa 0

Układ Bazy Danych

Tabele

1. **Employee_auth** - zawiera skróty `MD5` haseł pracowników `emp_password`
2. **Employee_data** - przechowuje dane pracowników `emp_data`
3. **Employee_relations** - opisuje relacje pomiędzy pracownikowami i ich przełożonymi
   - `emp_id`: klucz główny, indentyfikator pracownika
   - `emp_superior`: indeksowany identyfikator `emp_id` bezpośredniego przełożonego
   - `id`: indeksowany identyfikator nadawany w trakcie działania algorytmu
   - `inferior_end`: indeksowany identyfikator `id` najdalszego podwładnego
4. **Employee_root** - przechowywany jest identyfikator `emp_id` pracownika `root`



Funkcje

- `insert_employee(emp_id, emp_superior, password, data)` - funkcja zezwalająca na dodanie pracownika do bazy
- `remove_employee(emp_id)` - funkcja zezwalająca na usunięcie pracownika z bazdy
- `update_po(emp_id bigint, id bigint, inferior_end bigint)` - funkcja zezwalająca na zaktualizowanie tabeli `Employee_relations`

Użytkownicy

- `init` - prawo do tworzenia uzytkowników w obrębie bazy danych `CREATEROLE`, brak praw do tworzenia baz danych `NOCREATEDB`
- `app` z prawami:
  - `SELECT` na wszystkich tabelach w bazie danych
  - `UPDATE` na kolumnie `emp_data` w tabeli `Employee_data`
  - wywołania funkcji `insert_employee`, `remove_employee`, `update_po` z uprawnieniami użytkownika `init`

Algorytm post-order

Dzięki zastosowaniu algorytmu przechodzenia drzewa w porządku `post-order` w każdym rekordzie trzymamy zakres pełnego poddrzewa o korzeniu w tym rekordzie. Dzięki czemu możemy bardzo wydajnie odpowiadać na zapytania `ancestor`, `ancestors` oraz `descendants`. Wadą takiego rozwiązania jest to, że po każdym wstawieniu nowego pracownika do bazy danych trzeba przeindeksować całą bazę, jednak zgodnie z treścią zadania, zapytania te będą występować stosunkowo rzadko.

API

Opis funkcji

```
open <database> <login> <password>
// przekazuje dane umożliwiające podłączenie Twojego programu do bazy - nazwę bazy,
login oraz hasło, wywoływane dokładnie jeden raz, w pierwszej linii wejścia
// zwraca status OK/ERROR w zależności od tego czy udało się nawiązać połączenie z
bazą

root <secret> <newpassword> <data> <emp> // tworzy nowego pracownika **o unikalnym
indentyfikatorze <emp>** z hasłem <newpassword>, jest to jedyny pracownik, który
nie ma przełożonego, argument <secret> musi być równy 'qwerty'
// zwraca status OK/ERROR


new <admin> <passwd> <data> <newpasswd> <emp1> <emp> dodawanie nowego pracownika o
identyfikatorze <emp> z danymi <data> i hasłem dostępu <newpasswd>, pracownik <emp>
staje się podwładnym pracownika <emp1>, <admin> musi być pracownikiem o
identyfikatorze <emp1> lub jego bezpośrednim lub pośrednim przełożonym, <passwd> to
hasło pracownika <admin>
// nie zwraca danych

remove <admin> <passwd> <emp> usuwanie pracownika <emp> wraz z wszystkimi
pracownikami, którzy mu (bezpośrednio lub pośrednio) podlegają. <admin> musi być
bezpośrednim lub pośrednim przełożonym pracownika <emp> (zauważ, że oznacza to , że
nie da się usunąć prezesa), <passwd> to hasło pracownika <admin>
// nie zwraca danych

child <admin> <passwd> <emp>  zwraca identyfikatory wszystkich pracowników
bezpośrednio podległych <emp>, <admin> może być dowolnym pracownikiem,, <passwd> to
hasło pracownika <admin>
```

```
// tabela data powinna zawierać kolejne wartości <emp>

parent <admin> <passwd> <emp> zwraca identyfikator bezpośredniego przełożonego
<emp>, <admin> może być dowolnym pracownikiem, <passwd> to hasło pracownika <admin>
// tabela data powinna zawierać dokładnie jedną wartość <emp>

ancestors <admin> <passwd> <emp> zwraca identyfikatory wszystkich pracowników,
którym <emp> pośrednio lub bezpośrednio podlega, <admin> może być dowolnym
pracownikiem, <passwd> to hasło pracownika <admin>
// tabela data powinna zawierać kolejne wartości <emp>

descendants <admin> <passwd> <emp> zwraca identyfikatory wszystkich pracowników
bezpośrednio lub pośrednio podległych <emp>, <admin> może być dowolnym pracownikiem,
<passwd> to hasło pracownika <admin>
// tabela data powinna zawierać kolejne wartości <emp>

ancestor <admin> <passwd> <emp1> <emp2> sprawdza czy <emp1> bezpośrednio lub
pośrednio podlega <emp2>, <admin> może być dowolnym pracownikiem, <passwd> to hasło
pracownika <admin>
// tabela data powinna zawierać dokładnie jedną wartość: true albo false

read <admin> <passwd> <emp> // zwraca dane <data> pracownika <emp>, <admin> musi
być musi być pracownikiem <emp> lub bezpośrednim lub pośrednim przełożonym
pracownika <emp>, <passwd> to hasło pracownika <admin>
// tabela data powinna dokładnie jedną wartość <data>

update <admin> <passwd> <emp> <newdata> // zmienia dane pracownika <emp> na
<newdata>, <admin> musi być pracownikiem <emp> lub bezpośrednim lub pośrednim
przełożonym pracownika <emp>, <passwd> to hasło pracownika <admin>
// nie zwraca danych
```

Implementacja

- **new**, **root**

```
SELECT insert_employee(...)
```

- **remove**

```
SELECT remove_employee(...)
```

- **descendants**

```
WITH sup AS (
    SELECT id, inferior_end
    FROM "Employee_relations"
    WHERE emp_id = $emp
)
SELECT emp_id FROM "Employee_relations"
WHERE id BETWEEN (SELECT id FROM sup) + 1 and (SELECT inferior_end FROM sup) ORDER
BY id
```

- **ancestors**

```
WITH inf AS (
    SELECT id
    FROM "Employee_relations"
    WHERE emp_id = $emp
)
SELECT emp_id FROM "Employee_relations"
WHERE (SELECT id FROM inf) BETWEEN id + 1 and inferior_end ORDER BY id
```

- **ancestor**

```
WITH inf AS (
    SELECT id
    FROM "Employee_relations"
    WHERE emp_id = $emp1
    LIMIT 1
)
SELECT 1 FROM "Employee_relations"
WHERE emp_id = $emp2 AND (SELECT id FROM inf) BETWEEN id + 1 and inferior_end LIMIT
1
    ```

- **`update`**

```sql
SELECT update_po(...)
```

- **parent**

```
SELECT emp_superior FROM "Employee_relations" WHERE emp_id = $emp
```

- **child**

```
SELECT emp_id FROM "Employee_relations"
    WHERE emp_superior = $emp ORDER BY emp_superior
```

- **read**

```
SELECT emp_data FROM "Employee_data"
    WHERE emp_id = $emp
```

Uwagi techniczne

Uruchomienie

Program należy uruchomić poleceniem `python3 app.py [--init] [> STDOUT] < STDIN`, w razie braku zainstalowanego modułu `psycopg2` należy zainstalować owy moduł, używając przykładowo polecenia `pip3 install psycopg2`. Pełny model fizyczny oraz przykładowa konfiguracja bazy znajdują się odpowiednio w plikach `tables.sql` oraz `database.sql`

Format pliku wejściowego

Każda linia pliku wejściowego powinna zawierać obiekt `JSON`. Każdy z obiektów opisuje wywołanie jednej funkcji API wraz z argumentami.

W pierwszej linii wejścia znajduje się wywołanie funkcji `open` z argumentami umożliwiającymi nawiązanie połączenia z bazą danych.

W przypadku uruchomienia z flagą `--init` w drugiej linii musi znajdować poprawnie opisana funkcja `root`, a w kolejnych wierszach jedynie funkcje `new`.

W przeciwnym razie w kolejnych wierszach mogą pojawić się wszystkie funkcje z wyjątkiem `root` oraz `open`.

Przykładowe wejście

```
{"open": {"password": "qwerty", "login": "app", "database": "XCorp"}}
{"remove": {"admin": 2, "emp": 99, "passwd": "qwerty2"}}
{"child": {"admin": 675, "emp": 9, "passwd": "qwerty675"}}
```

Format pliku wyjściowego

Dla każdego wywołania program wypisuje w osobnej linii obiekt JSON zawierający `status` wykonania funkcji `OK/ERROR`, tabelę `data` zawierającą `wynik działania` tej funkcji wg specyfikacji oraz ewentualnie dodatkowe dane `debug`.

Przykładowe wyjście

```
{
  "status":"OK",
  "data": ["v1", "v2"],
  "debug": "..."
}
```