Dochádzkový systém

Záverečná správa
Databázy (2)

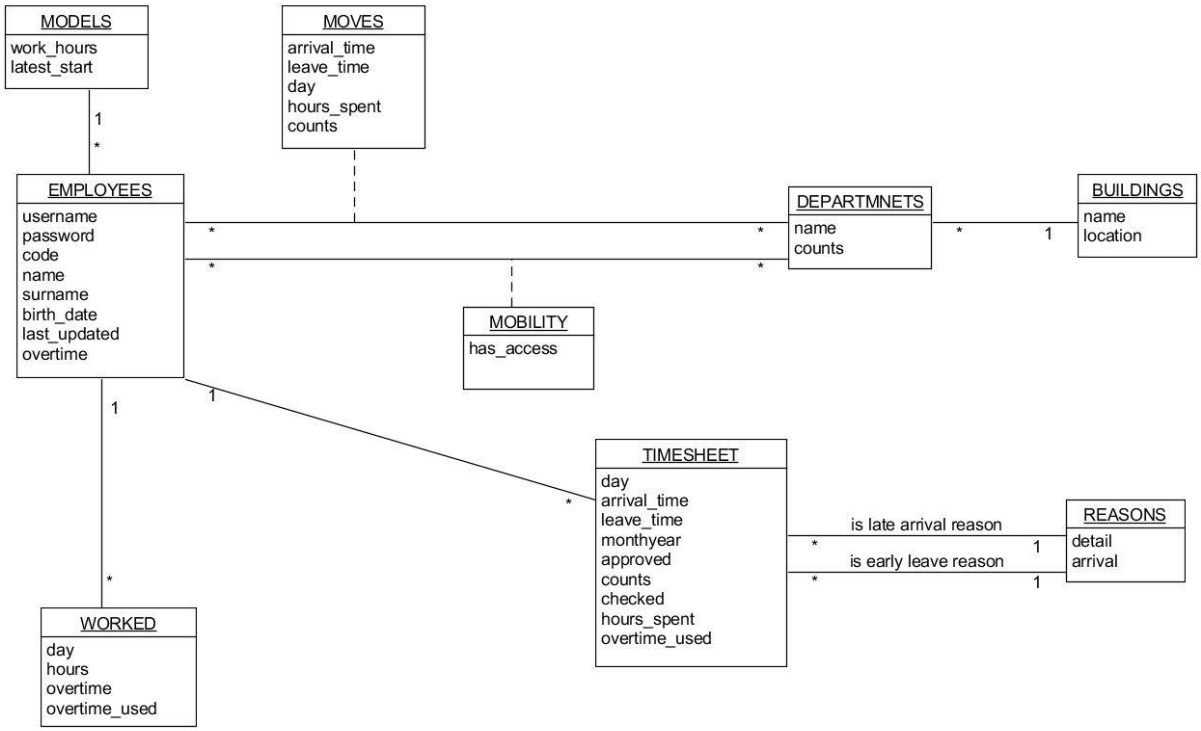
Linda Jurkasová

10.5.2019

1. O dokumente

Dokument slúži ako Záverečná správa, k semestrálnemu projektu k predmetu Databázy (2). Zachytáva logiku dochádzkového systému, evidenciu zamestnancov, ich príchodov a odchodov z pracoviska a pohybmi po oddeleniach ako aj počítaním celkovo odpracovaných hodín, nadrobených hodín aj ich využitie.

2. Dátový model



Obrázok 1: Entitno relačný model dát dochádzkového systému

Daný model dochádzkového systému (Obrázok 1) si uchováva zamestnancov v množine **EMPLOYEES**. Ku každému zamestnancovi sa viaže pracovný model, ktorý určuje najneskorší začiatok pracovnej doby a stanovený počet hodín, ktoré musí odpracovať. Tieto modely sú uchovávané v množine **MODELS**. Pracovisko má viacero budov, ktoré sú uložené v množine **BUILDINGS**. Každá budova obsahuje niekoľko oddelení, ktoré sú uložené v množine **DEPARTMENTS**. Medzi oddeleniami sú aj také, v ktorých pobyt sa neráta do pracovnej doby. Túto skutočnosť určuje parameter *counts*. Zamestnanci majú pre každé oddelenie určenú mobilitu, či na dané oddelenie majú prístup alebo nie. Tento vzťah je definovaný v množine **MOBILITY**. Pohyby zamestnancov po oddeleniach sú zaznamenané v množine **MOVES**, pričom parameter *counts* určuje či sa mu pobyt na danom oddelení (ak sa neráta do pracovnej doby) má odrátavať z odpracovaných hodín za daný deň.

Príchody a odchody z pracoviska sú zaznamenané v množine **TIMESHEET**, kde sa uchováva presný čas, kedy zamestnanec prešiel cez hlavný vchod. Na základe príchodu a odchodu sa narátava *hours_spent*, čo uľahčuje počítanie odpracovaných hodín. Takisto pri neskorom príchode alebo odchode z pracoviska ak nemá splnenú pracovnú dobu sú v množine **TIMESHEET** zaznamenané dôvody. Tieto

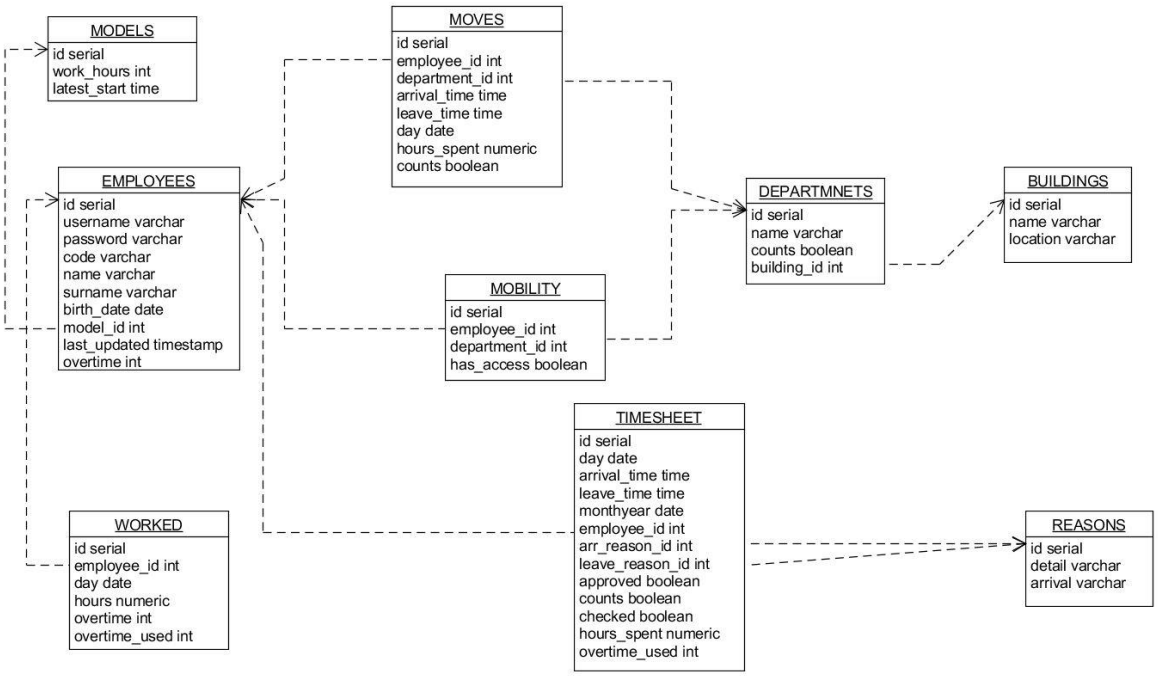
dôvody sú zadefinované v množine **REASONS**. Niektoré dôvody odchodu ukončujú odpracovanú dobu za daný deň, iné (ako napr. lekár, rokovanie, obedná prestávka) vyžadujú aby sa zamestnanec po určitej dobe vrátil na pracovisko. Pre tieto dôvody sa do množiny **TIMESHEET** zaznamenávajú extra riadky, ktoré určujú čas, ktorý strávil zamestnanec mimo pracovisko. Tento prístup nám umožňuje, jednoduchšie narátavanie odpracovaných hodín a flexibilitu pri schvaľovaní neskontrolovaných dôvodov. Niektoré dôvody vyžadujú schválenie kompetentnou osobou aby sa mohli zarátavať do fondu odpracovaných hodín. Či bol dôvod schválený určuje parameter *approved* a či už bol daný dôvod kompetentnou osobou skontrolovaný určuje *checked*. Parameter *counts* opäť určuje či sa daný záznam ráta do pracovnej doby (napr. príchod z nepracovných dôvodov sa neráta).

Celkový počet odpracovaných hodín, nadrobených hodín a využitia nadrobených hodín pre daného zamestnanca pre daný deň je uchovávané v množine **WORKED**, ktorá sa aktualizuje na konci dňa, podľa záznamov v množine **TIMESHEET**.

3. Zmeny voči zadaniu

Do zložitejších doménových operácií som pridala odchod z oddelenia, nakoľko som potrebovala údaj o tom či dané oddelenie už opustil.

4. Relačná databáza



Obrázok 2: Relačný model dát dochádzkového systému

Transformáciou entitno relačného modelu z Obrázku 1 dostanem daný relačný model zobrazený na Obrázku 2.

5. Organizácia kódu

Semestrálny projekt je naprogramovaný v objektovo orientovanom jazyku Java. Pri prístupe do databázy som využila aplikačné rozhranie JDBC, ktoré je priamo implementované v Jave. Štruktúru tried som riešila pomocou vzoru Row Data Gateway a zložitejšie doménové operácie využívajú vzor

Transaction Scriptu. Kvôli organizácii kódu som zdrojový kód celého projektu rozdelila do nasledovných balíkov

app

Obsahuje hlavnú triedu Main, ktorá tvorí základ projektu. Pomocou nej sa celý projekt spúšťa. Takisto obsahuje triedu DBContext, navrhnutú ako singleton a to z toho dôvodu, aby sa udržiavalo iba jedno spojenie na databázový systém.

app.rdg

Tento balík mapuje každú tabuľku v databáze na príslušnú triedu vytvorenú podľa vzoru Row Data Gateway. Pre niektoré tabuľky (napr. Timesheet, Worked) boli vytvorené triedy, ktoré reprezentujú jeden riadok tabuľky (TimsheetRow, WorkedRow) ale aj pomocné triedy (Timesheet, Worked), ktoré asistujú pri tvorbe riadkov do danej tabuľky aby sa to nemuselo riešiť mimo danej triedy, keďže vytváranie daných riadkov je o čosi komplikovanejšie.

Pre účely vypisovania riadkov je definovaná abstraktná trieda Row, ktorá má predpísané metódy na vypísanie riadka (print) a hlavičky (printHeader). Väčšina tried, ktoré predstavujú jeden riadok tabuľky a sú v danej aplikácii vypisované dedia od triedy Row.

app.ts

Obsahuje triedy pre zložitejšie doménové operácie, realizované vzorom Transaction Script. Triedy využívajú viacero tabuliek, aby sa zistilo či má na vstup/odchod oprávnenie preto sú v daných triedach pomocné funkcie, ktoré to kontrolujú. Je tu definovaná aj TSException, ktorá slúži na oznamovanie chýb užívateľovi.

app.statistics

Tento balík obsahuje triedy, ktoré vykonávajú štatistiky, ako aj triedy na reprezentovanie jedného riadku danej štatistiky. Štatistika pre nadrobené hodiny využíva abstraktnú triedu reprezentujúcu jeden riadok štatistiky OvertimeStatisticsRow, ktorá obsahuje metódu na vypisovanie riadku, keďže obsah medzi danými dvomi štatistikami pre nadrobené hodiny a využitie nadrobených hodín je v podstate rovnaký. Líšia sa jedine hlavičkou výpisu, ktorá je definovaná v dedených triedach OvertimeRow, OvertimeUsedRow.

app.finders

Obsahuje Findery pre triedy v balíku app.rdg

app.filters

Tento balík obsahuje pomocné triedy, ktoré som si nazvala filters. Pri niektorých operáciach bolo výhodnejšie vybrať viacero riadkov (napr. pre celý deň) a na základe daného zamestnanca vyfiltrovať riadky, alebo vypočítať všetky odrobené hodiny za daný deň. V konštruktore daných tried sa priamo volá triede určený finder ktorým naplním zoznam, ktorý si udržujem ako členskú premennú. Dané funkcie sú realizované pomocou Streamov pre jednoduchosť kódu.

app.ui

Tento balík obsahuje triedy slúžiace na interakciu s používateľom. Používateľské rozhranie je ovládané príkazovým riadkom, pričom sa pokaždé zobrazí MENU s funkciami aplikácie a používateľ si vyberie jednu z nich zadaním príslušného čísla. Obsahuje dva Scrollery, ktoré zabezpečujú stránkovanie pričom opakujúci sa kód je vyčlenený do triedy AbstractScroller, od ktorého obidva dedia.

Obsahuje aj triedu Checker, ktorá slúži na vypisovanie riadku a podľa vstupu užívateľa ho schváli alebo neschváli. Vypisovanie riadkov je realizované triedou Printer, ktorá využíva abstraktnú triedu Row, ktorá bola vyššie spomenutá.

Kvôli organizácii kódu, tento balík obsahuje pomocnú triedu DateTime v ktorej sú uložené všetky funkcie pre prácu s dátumom a časom.

6. Optimalizácia SQL

Pri veľkom množstve dát boli niektoré SQL dopyty v generate_scripte veľmi neefektívne. Pôvodný kód, v ktorom som využívala subselecty, ktoré sa museli vyhodnocovať pre každý riadok som nahradila WITH konštrukciou.

```
UPDATE employees AS e
SET    overtime = (SELECT Sum(w.overtime)
                   FROM   worked w
                   WHERE  w.employee_id = e.id) - (SELECT Sum(w.overtime_used)
                                                   FROM   worked w
                                                   WHERE  w.employee_id = e.id);
```

Obrázok 3: Pôvodný kód

```
WITH sumovertime(id, summ)
     AS (SELECT w.employee_id,
                Sum(w.overtime) - Sum(w.overtime_used)
         FROM   worked w
         GROUP  BY w.employee_id)
UPDATE employees AS e
SET    overtime = s.summ
FROM   sumovertime s
WHERE  e.id = s.id;
```

Obrázok 4: obrázok 3 po optimalizácií

```
UPDATE worked w
SET    overtime_used = (SELECT Sum(t.overtime_used)
                        FROM   timesheet AS t
                        WHERE  t.counts = true
                               AND t.employee_id = w.employee_id
                               AND w.day = t.day);
```

Obrázok 5: Pôvodný kód

```sql
WITH overtimeused(day, emp, used)
     AS (SELECT t.day,
                t.employee_id,
                Sum(t.overtime_used)
         FROM   timesheet t
         WHERE  t.counts = true
         GROUP  BY t.day,
                   t.employee_id)
UPDATE worked w
SET    overtime_used = o.used
FROM   overtimeused o
WHERE  w.day = o.day
       AND w.employee_id = o.emp;
```

Obrázok 6: obrázok 5 po optimalizácií

Kód	*Veľkosť tabuľky WORKED (v riadkoch)*	*Veľkosť tabuľky TIMESHEET (v riadkoch)*	*Execution Time (v ms)*	
			Pred optimalizáciou	Po optimalizácii
Obrázok 3	400 000	800 000	15 109.400 ms	1 201.833
Obrázok 5	400 000	800 000	834 88.592 ms	161 619.281

Kód som optimalizovala aj použitím indexov:

```sql
CREATE INDEX emp_index
  ON mobility (employee_id);

CREATE INDEX dep_index
  ON mobility (department_id);

CREATE INDEX day_index
  ON timesheet(day);
```

Väčšina prehľadávaní je na základe primárneho kľúča alebo stĺpcov, ktoré majú UNIQUE obmedzenie, preto som tieto INDEXY už vytvárať nemusela.

7. Vybraný riešený problém

Pôvodne som uvažovala že tabuľku TIMESHEET budem riešiť tak, že pri vstupe aj odchode sa vloží jeden riadok s časom. To by ale spôsobilo, že by sa nedalo zistiť či je to príchod alebo odchod, bez toho aby sa spočítali všetky riadky pre daného človeka za deň. Započítavanie hodín by sa muselo riešiť zložitými operáciami, keďže pri každom riadku by sa musel nájsť predchádzajúci a časy by sa museli odčítať.

Riešením bolo zmeniť štruktúru riadku tak aby sa zadal čas príchodu a potom sa aktualizoval čas odchodu. Tento spôsob mi umožnil vytvoriť stĺpec, v ktorom sa hneď hodiny započítavajú. To uľahčilo započítavanie odpracovaných hodín a tvorbu štatistík.

Ďalší problém ale nastal, keď som si uvedomila, že niektoré dôvody sa musia schvaľovať a po ich schválení sa započítavajú do pracovnej doby a že niektoré odchody z práce sa rátajú do pracovnej doby. Pôvodne som to chcela riešiť pomocou Javy kde by sa našiel predošlý riadok a odčítali by sa časy a následne na základe podmienok by sa tento čas pripočítal do pracovnej doby. Nastal ale problém v tom, že pri udaní dôvodu príchodu a odchodu by sa museli kontrolovať obidva a musela by som duplikovať stĺpce approved a checked, ktoré určujú či je daný záznam skontrolovaný a schválený. Muselo by sa to duplikovať pre príchod a odchod.

Na vyriešenie tohto problému som zmenila logiku pridávania riadkov nasledovne

- Ak zamestnanec príde v daný deň prvý krát do práce a meška z dôvodu návštevy lekára, systém do tabuľky pridá riadok navyše, ktorý bude reprezentovať čas u lekára. Teda príchod bude jeho najneskorší príchod do práce podľa jeho modelu a odchod bude čas kedy reálne prišiel do práce. Tomuto riadku sa nastaví checked, approved a counts na false. Kým nebude tento riadok schválený zamestnancovi sa neráta do pracovnej doby. Po jeho skontrolovaní sa aktualizujú tieto stĺpce a na základe toho či je schváleny sa podľa stĺpca counts započíta do pracovnej doby
- Ak zamestnanec odchádza z práce skôr a udá dôvod odchodu ako lekár, rokovanie a obed, do tabuľky sa mu pridá navyše riadok ktorý opäť reprezentuje čas mimo práce. Pri príchode do práce sa mu automaticky aktualizuje daný riadok a vloží sa nový pre reálny príchod do práce. Pri rokovaní sa automaticky riadok ráta do pracovnej doby, pri obede sa neráta a pri lekárovi záleží na tom či je schválený alebo nie.