FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Řešení problematiky informačního systému střední školy

Studijní program: B0613A140005 – Informační technologie
Studijní obor: Aplikovaná informatika
Autor práce: **Daniel Adámek**
Vedoucí práce: Ing. Lenka Kosková Třísková Ph.D.

Liberec 2024

Tento list nahraďte
originálem zadání.

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.
Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.
Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.
Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.
Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS STAG se shoduje s textem tištěné podoby práce.
Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.
Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

2. 8. 2024 Daniel Adámek

Řešení problematiky informačního systému střední školy

Abstrakt

Tato bakalářská práce se primárně zaměřuje na návrh a implementaci nových komponent pro řešení stávajícího informačního systému střední průmyslové školy, s cílem eliminovat identifikované slabiny a využít potenciál pro zlepšení. Práce vychází z podrobné analýzy současného stavu systému, včetně jeho funkcionalit, toku dat, synchronizačních procesů, API rozhraní, legislativních požadavků a bezpečnostních opatření. Na základě zjištěných výsledků se práce soustředí na návrh a implementaci klíčových funkcí pro efektivnější správu školy a odbourání automatizovatelných a repetetivních úkonů. Důraz je kladen na praktickou aplikovatelnost navrhovaných řešení, jejich integraci do stávajícího systému a možnost dalšího rozvoje.

Klíčová slova: informační systém, návrh, in-house vývoj, školství

Abstract

This bachelor thesis primarily focuses on the design and implementation of new components for the solution of the existing information system of a secondary industrial school, with the aim of eliminating identified weaknesses and exploiting the potential for improvement. The work is based on a detailed analysis of the current state of the system, including its functionalities, data flow, synchronization processes, API interfaces, legislative requirements and security measures. Based on the findings, the thesis focuses on the design and implementation of key features for more efficient school management and the elimination of automated and repetitive tasks. Emphasis is placed on the practical applicability of the proposed solutions, their integration into the existing system and the possibility of further development.

Keywords: information system, design, in-house development, education

Poděkování

Prvně bych rád vyjádřil svou vděčnost Střední průmyslové škole elektrotechnické, Praha 2, Ječná 30, kterou zastupuje ředitel Ing. Bc. et Bc. Ondřej Mandík ING-PAED IGIP. Bez jeho laskavosti a podpory by toto dílo nebylo možné realizovat. Škola mi poskytla nezbytné informace, umožnila mi upravit svůj informační systém.. Tato zkušenost byla pro mě nesmírně cenná a pomohla mi v mnoha studijních i pedagogických aspektech.
Zvláštní poděkování patří všem zaměstnancům školy a mým kolegům pedagogům, kteří mi pomohli kritickým pohledem na stávající systém a při hledání nových, lepších řešení. Jejich nápady a zpětná vazba byly jedním z klíčových aspektů pro vytvoření nového informačního systému, který je efektivní.
Dále bych chtěl poděkovat všem členům mého vedení, rodině a přátelům, kteří mi poskytli cennou pomoc a podporu během procesu vytváření této bakalářské práce. Jejich trpělivost, porozumění a povzbuzování bylo pro mě během celého procesu klíčové.
Nakonec bych chtěl vyjádřit svou vděčnost všem, kteří se na tomto díle podíleli, ať už přímo nebo nepřímo. Bez jejich kolektivního úsilí a podpory by tento projekt nebyl možný. Vaše práce a podpora byla cenná a jsem vám za to hluboce vděčný.
Děkuji všem.

Obsah

Seznam zkratek

FM TUL	Fakulta mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci
SPŠE Ječná	Střední průmyslová škola elektrotechnická, Praha 2, Ječná 30
MŠMT	Ministerstvo školství, mládeže a tělovýchovy
IS	Informační systém
ŠIS	Školní informační systém
OVM	Orgán veřejné moci
NSESSS	Národní standard pro elektronické systémy spisové služby
UCD	User Centered Design

1 Agenda tříd

1.1 Popis problému

Implementace agendy tříd v rozvrhovém API školního systému představuje komplexní úkol, jehož řešení vyžaduje pečlivé zvážení a hluboké pochopení jak administrativních aspektů pro správu školy, tak technických. Tento problém je esenciální pro správné fungování školního systému, který musí být schopen efektivně spravovat nejen aktuální rozvrhy a třídy, ale také uchovávat historická data.

1.2 Úvod do problematiky

Školní rozvrhový systém je srdcem administrativy školy, zajišťující, že učitelé, žáci a učebny jsou správně přiděleni v rámci různých časových bloků během školního roku. Tento systém musí zvládat nejen plánování aktuálních rozvrhů, ale také uchovávání historických dat.

1.3 Požadavky

Pro lepší pochopení problému je důležité nahlédnout do širších souvislostí. Školní systém musí splňovat následující požadavky:

- **Konzistence dat:** Zajištění, že data o třídách, rozvrzích a žácích jsou konzistentní napříč různými časovými obdobími.
- **Historická data:** Uchovávání historických záznamů pro administrativní účely.
- **Flexibilita:** Systém musí být schopen flexibilně reagovat na změny, jako je přechod žáků mezi třídami, změny učitelů a učeben.

V kontextu školního rozvrhového systému se agenda tříd týká správy informací o třídách, které zahrnují aktuální třídy, jejich žáky, rozvrhy, záznamy probrané látky (**??** třídnice) a absence. Systém musí být schopen všechny změny správně zaznamenat a reflektovat. Zmíněný úkol je komplikován několika faktory:

1. **Horizontální přechody žáků:** Žák během studia v průběhu školního roku může změnit třídu, obor nebo třeba školu za speficických podmínek, který stanovuje školský zákon a ředitel školy.

2. **Vertikální přechody žáků:** Žák, který nesplní zákonné požadavky pro postup do dalšího ročníku, tedy neprospěl, může být zařazen v následujícím školním roce do jiné třídy a ročník opakovat. Žák též může vzdělávání přerušit, či školu opustit.

3. **Přerušení studia:** Žák může během probíhajícího studia požádat ředitele školy o přerušení vzdělávání. Maximální doba trvání přerušení dle zákona § 66 odst. 5 zákona č. 561/2004 Sb. školský zákon - znění od 01.01.2024 [1] až na dobu dvou let s výjimkou mateřství [2], které se řídí jiným procesem.

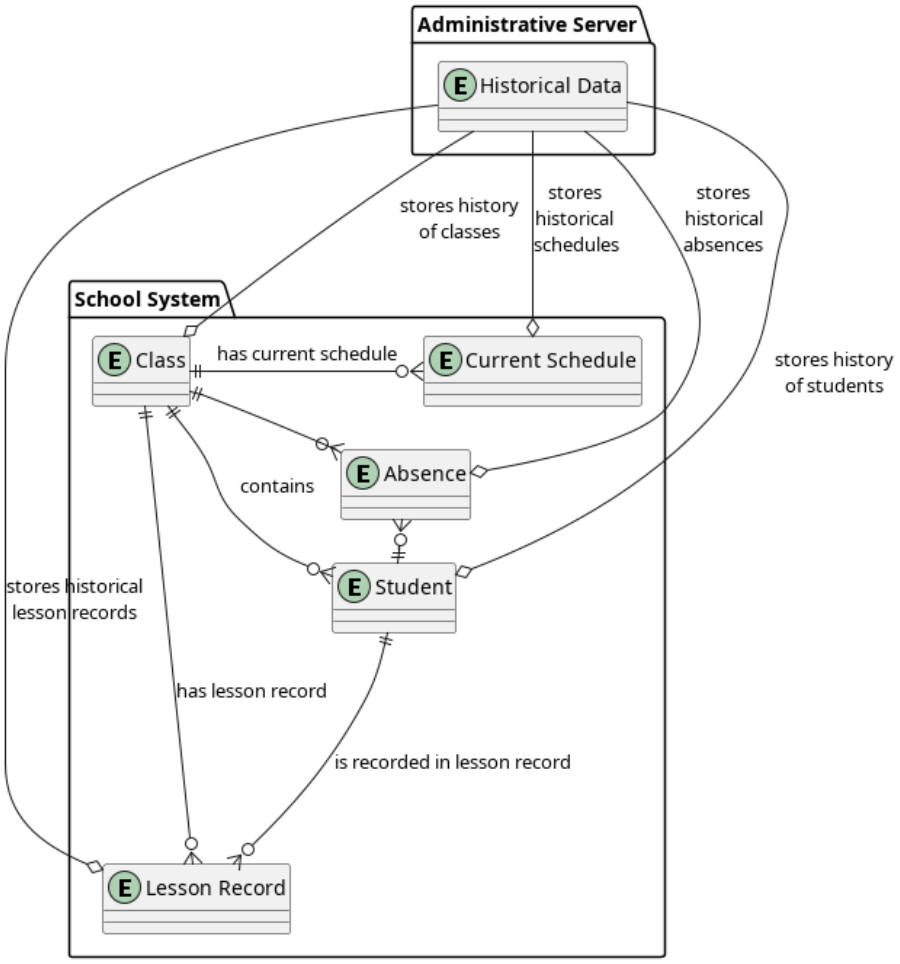
4. **Administrativní a kontrolní požadavky:** Uchovávání dat pro administrativní, právní a legislativní důvody, kterým mohou být inspekce státními orgány nebo například soudy.

Z těchto důvodů je implementace agendy tříd ve školním rozvrhovém systému složitá a vyžaduje pečlivé zvážení různých přístupů. Je důležité navrhnout systém, který bude nejen efektivní a flexibilní, ale také bude splňovat všechny uvedené požadavky. V následujících částech této kapitoly se budeme podrobněji zabývat jednotlivými návrhy implementace a jejich dopady na celkovou konzistenci a funkčnost školního systému.

1.4 Návrhy implementace

Na základě výše uvedených požadavků a kontextu jsou navrženy tři hlavní přístupy k implementaci agendy tříd v rozvrhovém API:

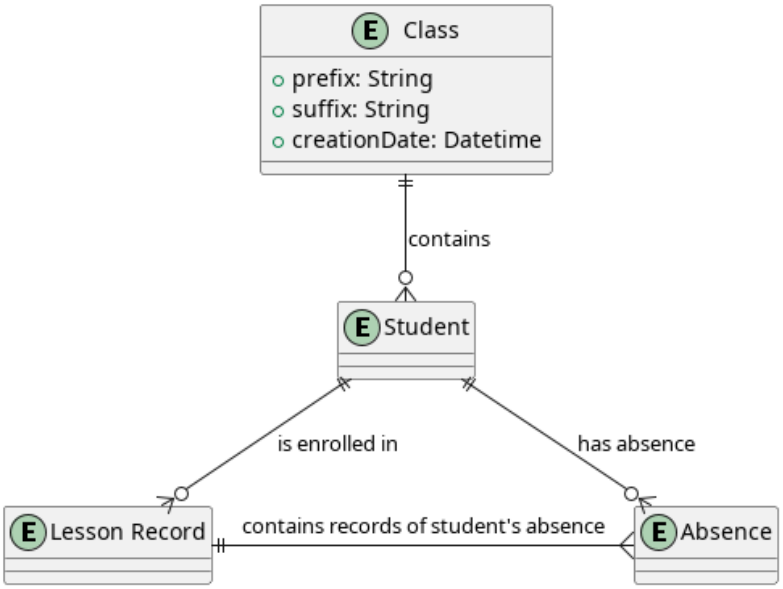
1. **Ukládání pouze aktuálního ročníku a aktuálních rozvrhů:** Tento přístup minimalizuje datovou zátěž na serveru tím, že uchovává pouze informace o aktuálním ročníku a rozvrzích. Historická data jsou uložena na odděleném administrativním serveru.



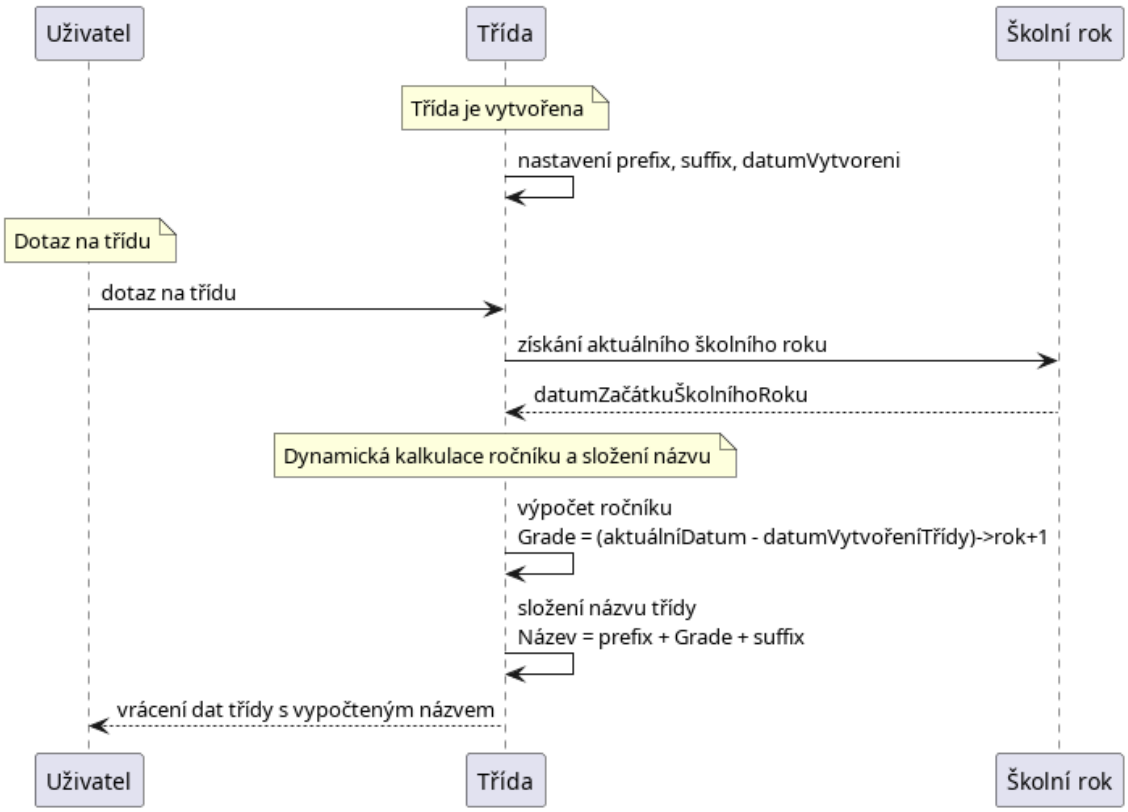
Obrázek 1.1: Diagram způsobu ukládání pouze aktuálního ročníku a aktuálních rozvrhů

Před začátkem každého školního roku by bylo tedy potřeba sjednat synchronizační proces pro archivování dat na administrativní server, smazání z rozvrhového serveru a nahrání nových dat. Tento synchronizačí mechanismus by mohl být obtížný pro správný chod. Předpokládáme, že oba servery spravuje jiný zaměstnanec. V případě chyby v programu způsobené programátory by bylo možné nenávratně přijít o důležitá data.

2. **Třída definována prefixem, suffixem, datem vytvoření a kalkulovaným ročníkem:** Tento přístup ukládá informace o třídách pomocí kombinace prefixu, suffixu a data vytvoření třídy. Ročník je kalkulován jako rozdíl mezi datem vytvoření a začátkem aktuálního školního roku.



Obrázek 1.2: Relační schéma varianty s kalkulovaným ročníkem



Obrázek 1.3: Schéma průběhu dotazu na jméno třídy

Problémem v tomto případě je samotná kalkulace. Pro dodržení aktuálnosti dat, musí být ročník vždy vypočítán, protože jakákoliv forma uchovávání

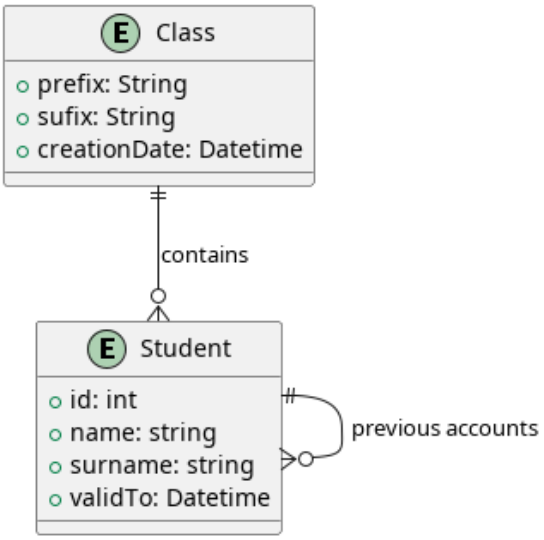
mezivýsledku by mohla vést k nekonzistenci v čase. Pokud dorazí požadavek k získání aktuálních tříd, musí server vypočíst všechny ročníky a spojit celé jméno třídy.

Dalším problémem tohoto řešení jsou propadlí nebo přecházející žáci. Jakmile žák propadne, nastávají 2 varianty řešení:

- odebrat žáka z aktuální třídy a přiřadit jej do nové třídy, ale ztratí minulé záznamy, protože třída je jedním z atributů žáka.

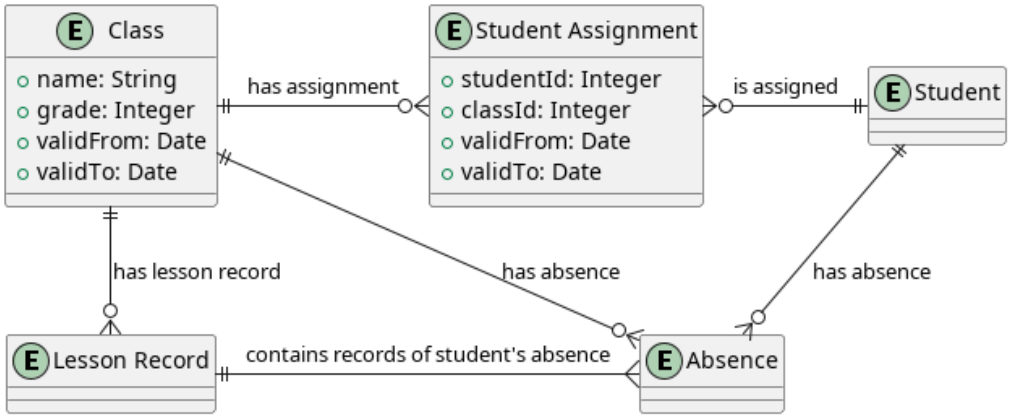
  Řešením může být např. psané záznamy o změnách, avšak nepokryjí všechny požadavky pro snadnou manipulaci s daty.
- vytvoření nového žáka, který začíná studovat v pokročilém ročníku. Tím by byla zajištěna konzistence dat, avšak duplicitní účty pro jednoho žáka.

  Řešením může být propojení minulých účtů žáka s novými (relací 1:N), kde žák by měl uložené své minulé účty. Stále je problém, že to není řešení příčiny, ale důsledku.



Obrázek 1.4: Schéma vazby na předchozí účty

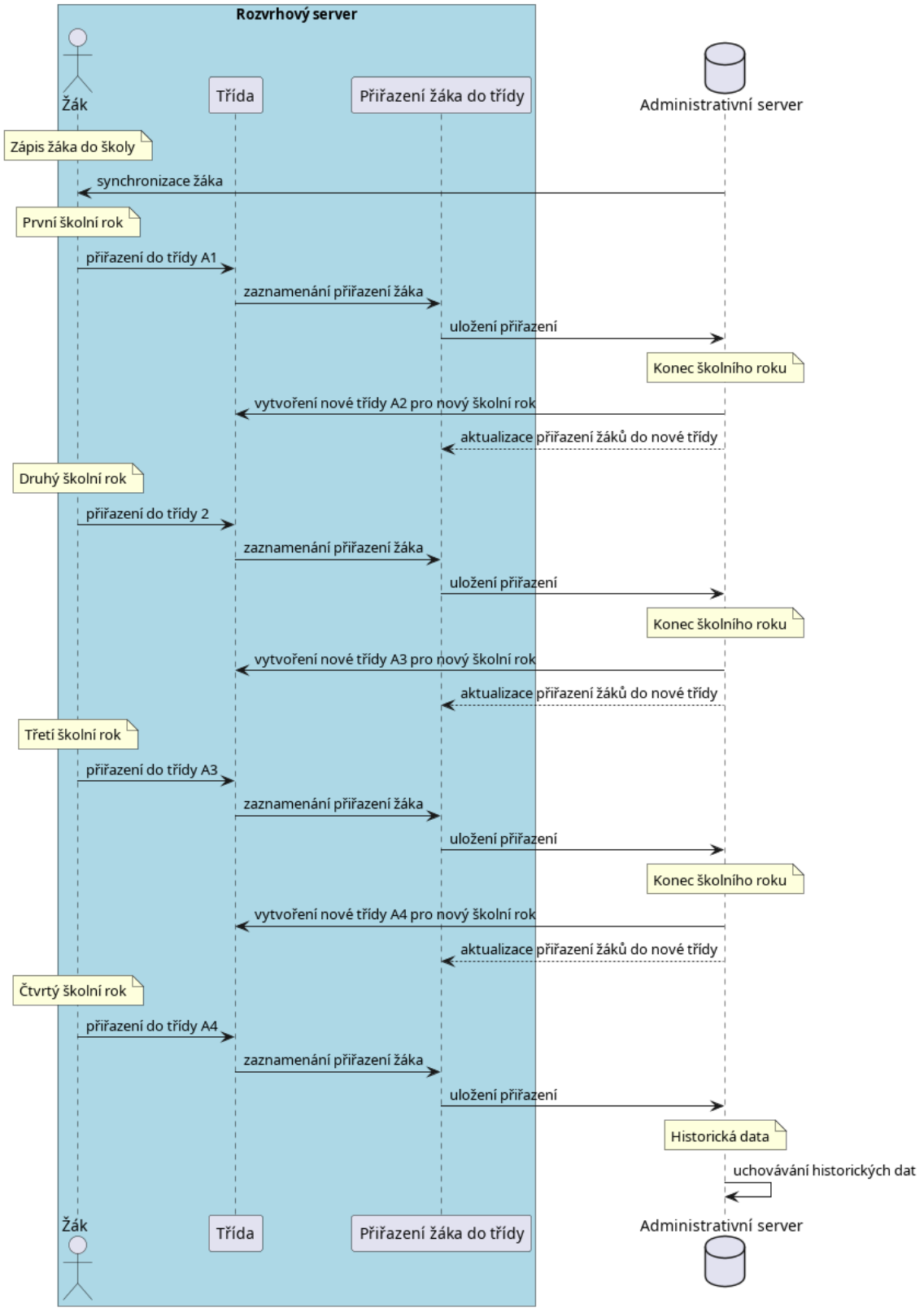
3. **Třída definována celým názvem a datem validity od - do:** Tento přístup zahrnuje kompletní informace o třídě včetně ročníku a platnosti od-do. Každý rok se vytváří nové třídy a vztahy mezi žáky a třídami jsou zaznamenány pomocí M:N tabulky.



Obrázek 1.5: Schéma vazby M:N pro přiřazení třídy v návrhu od - do

Hlavní výhodou tohoto řešení je konzistence dat i při přesunu žáků v třídách. V tom případě stačí ukončit validitu dotčených tříd, vytvořit nové se stejným názvem a přiřazení nových žáků ve správném pořadí.

Nevýhodou je vyšší náročnost dotazování na třídu konkrétního žáka. Databáze musí projít všechny přiřazené třídy žáka, vybrat tu aktuální dle dat rozsahu dat validity a následně třídu vybrat.



Obrázek 1.6: Schéma průchodu žáka školním procesem z podlehu rozvrhů v návrhu validity od - do

1.5 Vyhodnocení vhodného řešení

Přístup	**Výhody (+)**	**Nevýhody (-)**
Ukládání pouze aktuálního ročníku a aktuálních rozvrhů	1. Minimalizuje datovou zátěž na serveru 2. Jednoduché dotazování na aktuální data	1. Nutnost synchronizačního mechanismu 2. Riziko ztráty dat při chybě programu 3. Vyžaduje správu od různých zaměstnanců
Třída definována prefixem, suffixem, datem vytvoření a kalkulovaným ročníkem	1. Snížení počtu záznamů v databázi 2. Flexibilita v pojmenování tříd	1. Komplexita kalkulace ročníků 2. Potenciální nekonzistence dat 3. Obtížná manipulace s propadlými nebo přecházejícími žáky
Třída definována celým názvem a datem validity od - do	1. Konzistence dat při přesunech žáků 2. Jednoduchá správa historických záznamů 3. M:N vazby umožňují flexibilní přiřazení žáků	1. Vyšší náročnost dotazování na aktuální třídu 2. Zvýšená komplexita databázových dotazů

Tabulka 1.1: Porovnání přístupů

1.6 Vyhodnocení vhodného řešení

Přístup	Výhody (+)	Nevýhody (-)
Ukládání pouze aktuálního ročníku a aktuálních rozvrhů	1. Minimalizuje datovou zátěž na serveru 2. Jednoduché dotazování na aktuální data	1. Nutnost synchronizačního mechanismu 2. Riziko ztráty dat při chybě programu 3. Vyžaduje správu od různých zaměstnanců
Třída definována prefixem, suffixem, datem vytvoření a kalkulovaným ročníkem	1. Snížení počtu záznamů v databázi 2. Flexibilita v pojmenování tříd	1. Komplexita kalkulace ročníků 2. Potenciální nekonzistence dat 3. Obtížná manipulace s propadlými nebo přecházejícími žáky
Třída definována celým názvem a datem validity od - do	1. Konzistence dat při přesunech žáků 2. Jednoduchá správa historických záznamů 3. M:N vazby umožňují flexibilní přiřazení žáků	1. Vyšší náročnost dotazování na aktuální třídu 2. Zvýšená komplexita databázových dotazů

Tabulka 1.2: Porovnání přístupů

Na základě porovnání v tabulce 1.2 je nejlepší variantou: 3. **Třída definována celým názvem a datem validity od - do**. Tento přístup má více výhod (+) než nevýhod (-), a hlavní výhodou je jednoduchá správa historických záznamů a flexibilní vazba přiřazení žáků ke třídám a skupinám.

1.7 Implementace

V předchozí kapitole jsme diskutovali matematickou úvahu o třídách a jejich skupinách (SubClasses), kde jsme stanovili tři klíčové podmínky: úplnost, disjunkci a podmnožinovou strukturu. V této kapitole se zaměříme na implementaci těchto matematických principů do třídního a databázového modelu. Zároveň se pokusíme najít nejefektivnější řešení, které zajistí plnou konzistenci dat v databázi.

1.7.1 Třídní Model

Třídní model zahrnuje několik hlavních entit: `Class`, `SubClass`, `Student` a `StudentAssignment`. Model `Class` reprezentuje třídu jako celek, model `SubClass` reprezentuje skupiny v rámci třídy, model `Student` reprezentuje jednotlivé žáky a model `StudentAssignment` zaznamenává přiřazení žáků k třídám a skupinám.

Class

`Class` obsahuje informace o třídě, jako jsou identifikátor třídy, označení třídy, prefix, data platnosti, kmenovou učebnu a třídního učitele.

V této implementaci není speciálně řešena změna třídního učitele nebo kmenové učebny během školního roku. Při změně je třeba ukončit validitu staré třídy a založit novou se stejným názvem. Dojte tedy k duplikování všech vazeb žáků, tříd, skupin a rozvrhů. Tyto případy jsou zcela výjimečné a tedy není třeba situaci řešit komplikovaněji.

SubClass

`SubClass` reprezentuje skupiny žáků v rámci třídy. Každá skupina má své vlastní identifikační číslo, název a odkaz na třídu, ke které patří. SubClass může existovat pouze v kontextu třídy, což znamená, že každá SubClass musí být vždy přiřazena konkrétní třídě.

Student

`Student` obsahuje informace o jednotlivých žácích, jako jsou identifikační číslo, uživatelské jméno, jméno a příjmení.

StudentAssignment

`StudentAssignment` zaznamenává přiřazení žáků k třídám a skupinám. Obsahuje odkazy na žáky, třídy a skupiny, ke kterým jsou přiřazeni. Klíčovým prvkem tohoto modelu je zajištění trvanlivosti přiřazení, tedy schopnost zaznamenávat změny přiřazení žáků v průběhu roku.

V průběhu školního roku mohou nastat různé změny, které vyžadují aktualizaci přiřazení žáků. Tyto změny mohou zahrnovat mnoho faktorů a ty nejčastější z nich

jsou vyjmenovány níže. Trvanlivost přiřazení umožňuje udržovat historický záznam o všech těchto změnách, což je nezbytné pro správu.

Problémy a důvody pro trvanlivost:

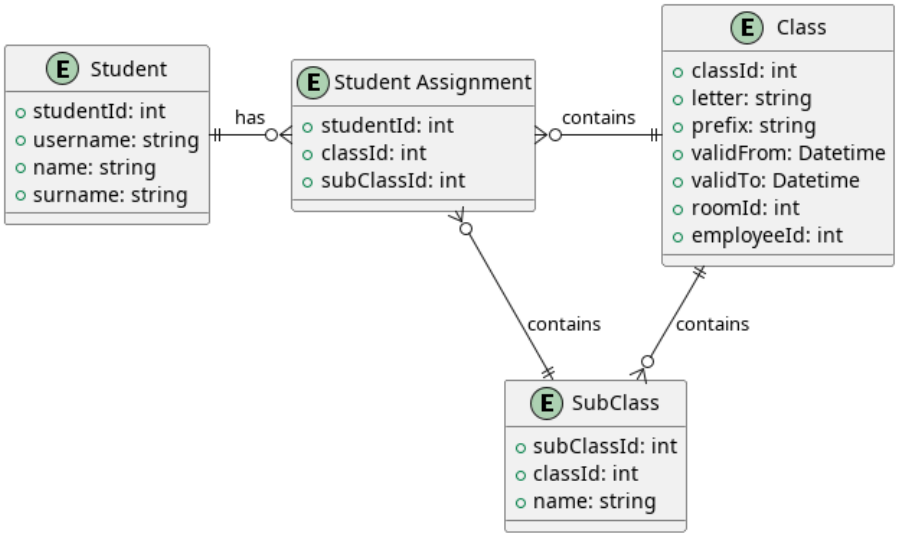
- **Přechody mezi třídami**: Žáci mohou být během školního roku přeřazeni do jiné třídy z různých důvodů, jako jsou změny nálad v třídním kolektivu nebo přestup žáka mezi obory.
- **Změny ve složení skupin**: V závislosti na specifických potřebách výuky mohou být žáci přeřazeni mezi různými skupinami. Například při výuce jazyků může být potřeba změnit složení skupin kvůli homogenizaci, nebo naopak heterogenizaci skupiny, čímž v závislosti na potřebách jednotlivých žáků lze docílit zefektivnění výuky.

Model `StudentAssignment` musí být schopen zaznamenávat všechny tyto změny, aby bylo možné udržovat konzistentní a aktuální přehled o přiřazení žáků. Tento model tedy umožňuje sledování přiřazení žáků k třídám a skupinám v určitém časovém období a zajišťuje, že historické změny jsou správně zaznamenány.

Schopnosti modelu:

- **Referenční integrita**: Každý záznam o přiřazení musí odkazovat na platné záznamy ve třídách a skupinách.
- **Flexibilita přiřazení**: Umožňuje přiřazení žáka buď přímo do třídy nebo do specifické skupiny.
- **Trvanlivost**: Udržuje historické záznamy o přiřazení, které umožňují sledovat změny v průběhu školního roku.

Tímto způsobem model `StudentAssignment` nejen zajišťuje aktuálnost a konzistenci dat, ale také poskytuje cenné historické informace, které mohou být využity pro analýzu a zlepšení vzdělávacího procesu.



Obrázek 1.7: Model tříd žákovských skupin

1.7.2 Databázový Model

Databázový model musí zajistit, že všechny podmínky úplnosti, disjunkce a podmnožinové struktury jsou dodrženy. Toho lze dosáhnout pomocí správného nastavení cizích klíčů, unikátních omezení a triggerů.

Úplnost

Úplnost zajišťuje, že každá třída je pokryta skupinami tak, aby žádný žák nezůstal nezařazen v dané kategorii (tzn. dělení na poloviny, třetiny, dívky a chlapce). V backendové části zajistíme, že všechny skupiny dohromady pokrývají celou třídu. To můžeme kontrolovat při každém přiřazení žáka do skupiny. Je nezbytné, aby všechny skupiny dané kategorie třídy byly disjunktní a jejich sjednocení tvořilo celou třídu.

Disjunkce

Disjunkci zajistíme na úrovni databáze pomocí unikátních omezení. Každý žák může patřit pouze do jedné skupiny určitého typu v dané třídě. To lze implementovat pomocí unikátních indexů nebo pomocí aplikační logiky, která před přiřazením žáka do skupiny provede potřebné kontroly.

Podmnožinová struktura

Podmnožinovou strukturu zajistíme pomocí cizích klíčů a referenční integrity. Každá skupina musí odkazovat na existující třídu a každé přiřazení žáka do skupiny musí odkazovat na existujícího žáka, třídu a skupinu.

1.7.3 Validace dat v aplikační logice

Validace dat je důležitým aspektem jakéhokoliv robustního informačního systému. V tomto návrhu systému zajišťuje validace dat konzistenci a integritu uložených informací. Níže jsou popsány validátory použité v aplikační logice.

Úvod do validace dat

Validace dat je proces ověřování, zda vstupní data splňují stanovené požadavky a pravidla. Tato pravidla mohou zahrnovat různé aspekty, jako je formát, rozsah, konzistence a vztahy mezi jednotlivými datovými poli. Ve školním systému je validace dat implementována na úrovni aplikační logiky. To zajišťuje, že pouze správně formátovaná a platná data jsou ukládána do databáze.

V této aplikaci je použito několik validátorů, které kontrolují různé aspekty vstupních dat. Každý validátor je navržen tak, aby ověřil konkrétní sadu pravidel a zajistil, že data jsou konzistentní a správná.

Validátor dat třídy (`validateClassDates`)

Validátor `validateClassDates` je zodpovědný za ověření, zda datum začátku (`validFrom`) třídy je dříve než datum konce (`validTo`). Zajišťuje, že žádná třída nemůže mít začátek po svém konci, což by bylo nelogické a mohlo by vést k chybám při dalších operacích.

```
function validateClassDates(instance: Class) {
    if (new Date(instance.validFrom) > new Date(instance.validTo)) {
        throw new Error('validFrom must be less than validTo');
    }
}
```

Kód validátoru dat třídy

Validátor unikátnosti názvu třídy (`validateClassName`)

Validátor `validateClassName` ověřuje, že jméno třídy je unikátní v daném období platnosti. Tím je zajištěno, že nemůže existovat více tříd se stejným jménem v tomtéž období. Tím validátor předchází duplicitě názvů tříd, což by mohlo vést ke zmatkům a chybám v rozvrhu a přiřazení studentů.

```
async function validateClassName(instance: Class) {
    const existingClass = await Class.findOne({
        where: {
            name: instance.name,
            validFrom: { [Op.lte]: instance.validTo },
            validTo: { [Op.gte]: instance.validFrom }
        }
    });

    if (existingClass) {
        throw new Error('Class name already exists in the given period'
           );
    }
}
```

Kód validátoru unikátnosti názvu třídy

Validátor časových intervalů tříd (`validateClassInterval`)

Validátor `validateClassInterval` zajišťuje, že nově vytvářená nebo aktualizovaná třída nemá časový interval, který by se překrýval s jinou třídou se stejným jménem, místností nebo učitelem. Tím zajišťuje, že třídy nebudou mít překrývající se časové intervaly, což by mohlo způsobit problémy s plánováním.

```
async function validateClassInterval(instance: Class) {
    const existingClass = await Class.findOne({
        where: {
          name: instance.name,
          roomId: instance.roomId,
          employeeId: instance.employeeId,
          validFrom: { [Op.lte]: instance.validTo },
          validTo: { [Op.gte]: instance.validFrom },
          classId: { [Op.ne]: instance.classId }
        }
      });

      if (existingClass) {
        throw new Error('Class interval is overlapping with another
            class');
      }
}
```

Kód validátoru časových intervalů tříd

Validátor existence učitele (`validateTeacherExistence`)

Validátor `validateTeacherExistence` kontroluje, zda zaměstnanec, který je přiřazen k třídě, skutečně existuje v databázi a je učitel. Tím je zajištěno, že nejsou přiřazováni neexistující učitelé nebo nepedagogičtí zaměstnanci.

```
async function validateTeacherExistence(instance: Class) {
    const teacher = await Employee.findOne({
        where: { employeeId: instance.employeeId, isTeacher: true }
      });

      if (!teacher) {
        throw new Error('Teacher does not exist');
      }
}
```

Kód validátoru existence učitele

Validátor rozvrhu učitele (`validateEmployeeSchedule`)

Validátor `validateEmployeeSchedule` zajišťuje, že učitel nemá přiřazen více tříd ve stejném časovém období. Tím zajišťuje konflikty v rozvrzích učitelů.

```
async function validateEmployeeSchedule(instance: Class) {
    const existingTeacher = await Class.findOne({
        where: {
            employeeId: instance.employeeId,
            validFrom: { [Op.lte]: instance.validTo },
            validTo: { [Op.gte]: instance.validFrom },
            classId: { [Op.ne]: instance.classId }
        }
    });

    if (conflictingClass) {
        throw new Error('Teacher is already assigned to another class
           in this period');
    }
}
```

Kód validátoru rozvrhu učitele

Validátor existence místnosti (`validateRoomExistence`)

Podobně jako validátor existence učitele, validátor `validateRoomExistence` ověřuje, že místnost přiřazená k třídě existuje v databázi a je učebnou. Zajišťuje, že místnost není například kancelář.

```
async function validateRoomExistence(instance: Class) {
    const room = await Room.findOne({
        where: {
            roomId: instance.roomId,
            type: 'classroom'
        }
    });

    if (!room) {
        throw new Error('Room does not exist');
    }
}
```

Kód validátoru existence místnosti

Validátor rozvrhu místnosti (`validateRoomSchedule`)

Validátor `validateRoomSchedule` ověřuje, že místnost není přiřazena více třídám ve stejném časovém období. Tím je zajištěno, že místnost může být kmenovou učebnou a místnost není přiřazena k více třídám zároveň.

```
async function validateRoomSchedule(instance: Class) {
    const existingRoom = await Class.findOne({
        where: {
          roomId: instance.roomId,
          validFrom: { [Op.lte]: instance.validTo },
          validTo: { [Op.gte]: instance.validFrom },
          classId: { [Op.ne]: instance.classId }
        }
    });

    if (existingRoom) {
        throw new Error(
          'Room is already assigned to another class within the
             validity period'
        );
    }
}
```

Kód validátoru rozvrhu místností

1.7.4 Diskuse o efektivitě řešení

Pro zajištění úplné konzistence dat v databázi je nutné implementovat následující mechanismy.

Validace na úrovni aplikace

Při každém přiřazení žáka do skupiny zkontrolujeme, zda přiřazení splňuje podmínky úplnosti a disjunkce. Pokud podmínky nejsou splněny, přiřazení se neprovede a uživatel obdrží příslušnou chybovou zprávu. Aplikační logika bude muset zohlednit také případy, kdy se celá třída učí společně a není třeba přiřazovat žáky do skupin.

Unikátní omezení

Na úrovni databáze zavedeme unikátní omezení, která zajistí, že každý žák může patřit pouze do jedné skupiny určitého typu v dané třídě. Tím zajistíme disjunkci dat.

Referenční integrita

Pomocí cizích klíčů zajistíme, že každá skupina je podmnožinou existující třídy a každé přiřazení žáka odkazuje na existujícího žáka, třídu a skupinu. Tím zajistíme podmnožinovou strukturu.

Implementace výše uvedených mechanismů v kombinaci s aplikační logikou poskytuje robustní řešení pro správu tříd a jejich skupin. Validace na úrovni aplikace umožňuje flexibilitu a zajišťuje, že jsou splněny všechny kladené podmínky. Unikátní omezení a referenční integrita na úrovni databáze zajišťují konzistenci dat a minimalizují riziko chyb.

Tímto způsobem zajišťujeme:

- **Úplnost**: Každá třída je kompletně pokryta skupinami a žádný žák nezůstane nezařazen.
- **Disjunkce**: Každý žák může být přiřazen pouze do jedné skupiny určitého typu v dané třídě.
- **Podmnožinová struktura**: Každá skupina je vždy podmnožinou konkrétní třídy, ke které patří.

2 Rozvrh

2.1 Popis problému

2.2 Úvod do problematiky

2.3 Požadavky

2.4 Návrh implementace

3 Reference

[1] *ČESKO. § 31 odst. 1 zákona č. 561/2004 Sb., o předškolním, základním, středním, vyšším odborném a jiném vzdělávání (školský zákon) - znění od 1. 1. 2024*. Revidováno: 17.06.2024. URL: https://www.zakonyprolidi.cz/cs/2004-561#p66-5.

[2] *ČESKO. § 31 odst. 1 zákona č. 561/2004 Sb., o předškolním, základním, středním, vyšším odborném a jiném vzdělávání (školský zákon) - znění od 1. 1. 2024*. Revidováno: 17.06.2024. URL: https://www.zakonyprolidi.cz/cs/2004-561#p66-6.

4 Přílohy

- Analýza pro proces modernizace školního informačního systému pro střední školu, Daniel Adámek