VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Databázové systémy – projekt
Vězeňská pekárna

28. dubna 2019 **Jakub Frejlach** (xfrejl00), **Ondřej Široký** (xsirok09)

Obsah

1 Úvod

Na projektu jsme se od začátku snažili pracovat včas a svědomitě. Práci jsme si rozdělovali po logických celcích. Na první a poslední části jsme pracovali společně, druhou a třetí část dělal vždy pouze jeden a druhý pouze kontroloval výsledek práce.

2 Zadání

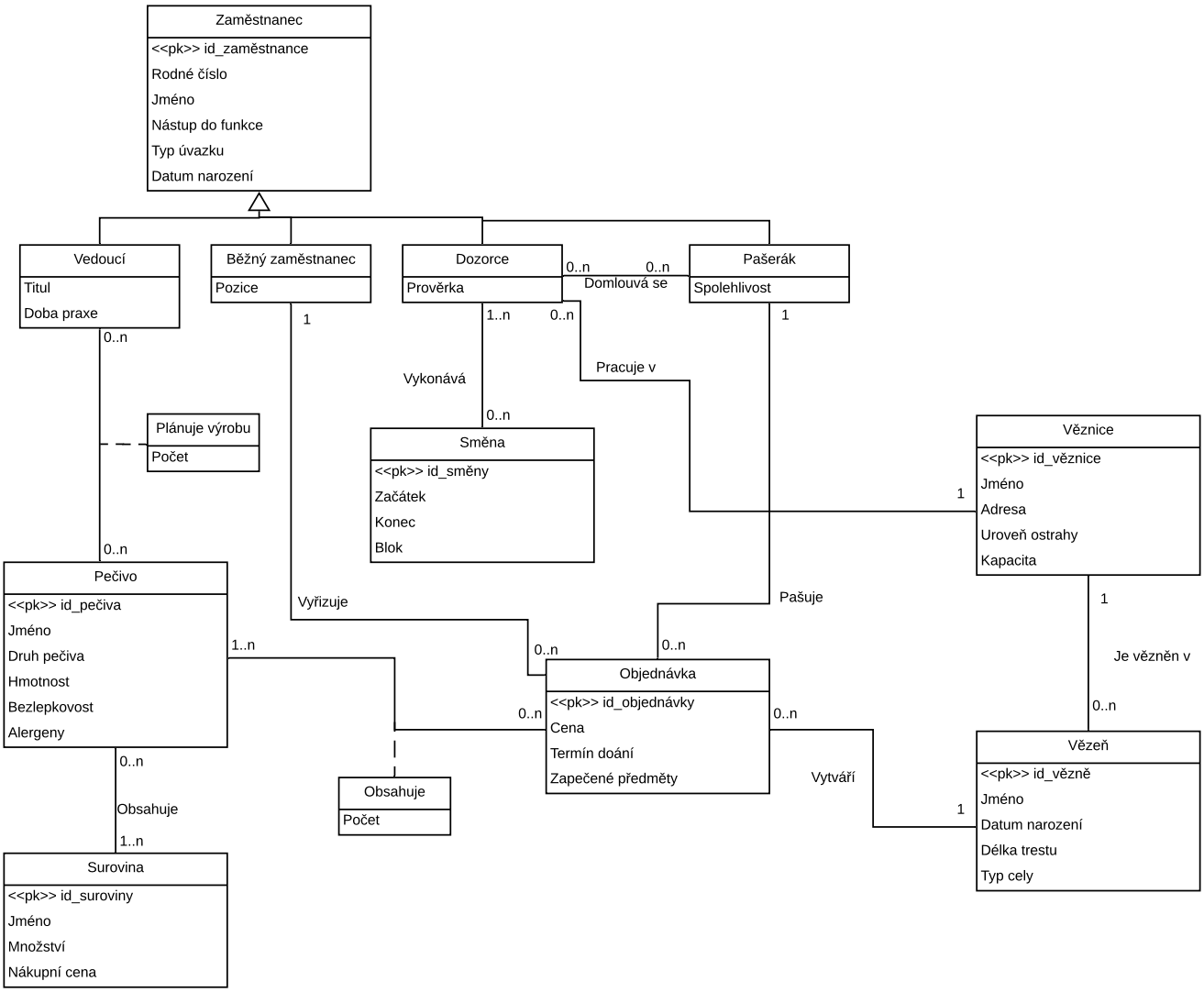
Náš projekt modeluje databázi vězeňské pekárny, jejíž zadání jsme převzali z předmětu Úvod do softwarového inženýrství (IUS). Celé zadání datového modelu:

> Navrhněte jednoduchý informační systém vězeňské pekárny, která dodává pečivo do okolních věznic. Systém umožňuje spravovat informace o nabízeném pečivu, věznicích, jejich vězních a umožňuje zákazníkům (vězňům) vytvářet objednávky pečiva. Objednávka musí obsahovat informaci o tom kolik a jakého pečiva je objednáváno a za jakou cenu, termín dodání, způsob dodání atd. Vězeňská pekárna navíc dle zákazníkova přání do každého pečiva zapeče vybraný předmět (pilník, šroubovák, .). Dále bude systém evidovat suroviny, které se pro výrobu pečiva používají včetně jejich aktuálního množství skladem a nákupní ceny. Musí také evidovat kolik a jakých surovin je pro výrobu daného pečiva potřeba, zda se jedná o bezlepkové pečivo, jeho druh (celozrnné, běžné, atd.), seznam alergenů a hmotnost (bez zapečeného předmětu). Do jednotlivých věznic dodávají pečivo pašeráci. Pašeráci jsou domluveni vždy s konkrétním dozorcem, díky kterému můžou do věznice pečivo bezpečně dodat. V systému tak musí být i informace o směnách, při kterých dozorci dohlíží na pořádek ve věznici. Na každé směně je více dozorců, směna má čas nástupu a čas ukončení. Vězeňská pekárna spolupracuje s několika pašeráky, přičemž jeden z nich může dodávat pečivo do několika věznic. Systém musí umožnit vedení pekárny plánovat produkci v závislosti na objednávkách a evidovat zákazníky. Pašeráci si mohou vypsat své rozvozy podle oblastí nebo podle zákazníků. Navíc, aby zaměstnanci pekárny věděli, zda budou vůbec schopni pečivo zákazníkovi dodat, musí mít v systému informaci o tom, ve které cele se zákazník nachází a o jaký typ cely se jedná (zda není zákazník na samotce, kam by se pečivo pašovalo stěží).

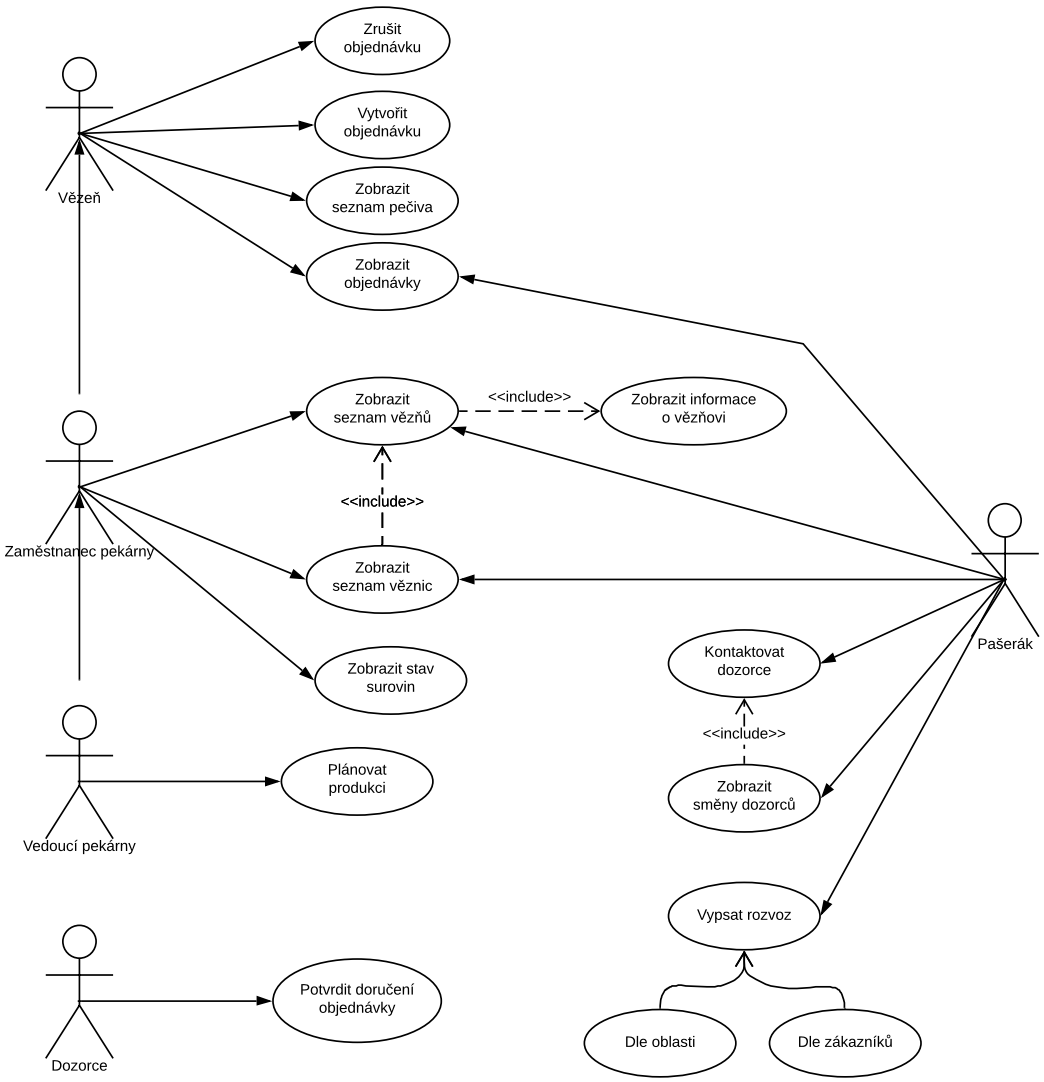
3 Datový návrh

ER diagram i model případů užití byly převzaty z předmětu Úvod do softwarového inženýrství (IUS), nicméně byly z větší části přepracovány, zejména kvůli relativně nízkému hodnocení ve výše zmíněném předmětu. Nový ER diagram i model případů užití jsem tvořili pomocí online nástroje `LucidChart`. ER diagram je poměrně rozsáhlý, protože musí pokrýt několik typů zaměstnanců a mnoho tabulek je několiknásobně provázaných. Třeba tabulka objednávek až čtyřikrát. Vyskytuje se zde také větší množství M:N relací. Vztah generalizace/specializace jsme realizovali na zaměstnancích pekárny/věznice, kdy jsme vedoucí, běžné zaměstnance, dozorce a pašeráky sdružili pod zaměstnance.

3.1 ER diagram



3.2 Model případů užití



4 Vytvoření základních objektů schématu databáze

Pomocí SQL skriptu bylo potřeba vytvořit následující tabulky:

- **Základní tabulky databáze**
  - pecivo
  - surovina
  - smena
  - objednavka
  - veznice
  - vezen
- **Tabulky pro generalizaci/specializaci**
  - zamestnanec
  - vedouci
  - bezny_zamestnanec
  - dozorce
  - paserak
- **Tabulky pro vymodelování M:N relací**
  - objednavka_obsahuje_pecivo
  - vedouci_planuje_vyrobu
  - pecivo_obsahuje_surovinu
  - dozorce_vykonava_smenu
  - paserak_domlouva_dozorce

Tabulky jsme následně naplnili ukázkovými daty. Většina údajů je vymyšlená, protože jsme neměli k dispozici reálná data, jelikož data o provozu věznic a pekáren většinou nebyla nikde k dohledání. Z reality vycházejí pouze jména věznic a údaje o pečivu.

5 Výběr dat – SQL dotazy

V rámci projektu bylo třeba vytvořit několik specifických dotazů v jazyce SQL. Zde se můžete podívat na jejich ukázky.

- **Spojení dvou tabulek (2 dotazy)**
  - Dotaz spojí tabulky objednávek a vězňů, vybere z toho spojení jen relevantní údaje k objednávce a vypíše všechny objednávky objednané vezněm Jiřím Zlámalem. (Upravením jména lze zjistit informace o objednávkách dalších vězňů.

```
SELECT id_objednavky, cena, termin_dodani, zapecene_predmety,
jmeno AS vezen
FROM objednavka NATURAL JOIN vezen
WHERE jmeno = 'Jiri Zlamal';
```

  - Dotaz spojí tabulku věznic s tabulkou vězňů a vybere pouze ty věznice, kde se nacházejí vězni na samotkách.

```
SELECT DISTINCT v.id_veznice, v.jmeno
FROM veznice v JOIN vezen k ON k.id_veznice = v.id_veznice
WHERE k.typ_cely = 'samotka';
```

- **Spojení tří tabulek (1 dotaz)**
  - Dotaz spojí tabulku zaměstnanců s tabulkou objednávek a s tabulkou vězňů a vybere všechny zaměstnance s plným, úvazkem, kteří vyřizovali objednávky pro vězně Jiřího Zlámala a zobrazí také předměty, které byly do objednávek zapečeny. Změnou jména vězně lze zjistit tyto informace i ostatních vězních.

```
SELECT b.id_zamestnance, z.jmeno,
o.zapecene_predmety, o.id_objednavky
FROM (bezny_zamestnanec b JOIN zamestnanec z
ON z.id_zamestnance = b.id_zamestnance)
JOIN objednavka o ON z.id_zamestnance = o.id_zamestnance
JOIN vezen v ON v.id_vezne = o.id_vezne
WHERE v.jmeno = 'Jiri Zlamal' and z.typ_uvazku = 'Full-time';
```

- **Klauzule GROUP BY (2 dotazy)**
  - Dotaz spojí tabulky pečiva a surovin a spočítá počet výskytů jednotlivých surovin přes agregační funkci COUNT a klauzuli GROUP BY. Počty výskytů poté seřadí pomocí ORDER BY.

```
SELECT COUNT(surovina.jmeno) as pocet_vyskytu, surovina.jmeno
FROM pecivo, pecivo_obsahuje_surovinu, surovina
WHERE pecivo.id_peciva = pecivo_obsahuje_surovinu.id_peciva
and surovina.id_suroviny = pecivo_obsahuje_surovinu.id_suroviny
GROUP BY surovina.jmeno
ORDER BY pocet_vyskytu;
```

- Dotaz spojí tabulku vězňů a věznic a poté k nim pomocí LEFT OUTER JOIN připojí ještě tabulku objednávek. Pomocí klauzule GROUP BY a agregační funkce COUNT je poté spočítáno, kolik mají jednotliví vězni objednávek. Dotaz ukáže i vězně, kteří nemají žádné objednávky.

```
SELECT COUNT(o.id_vezne) as pocet_objednavek, k.jmeno,
v.jmeno AS nazev_veznice
FROM veznice v JOIN vezen k ON v.id_veznice = k.id_veznice
LEFT OUTER JOIN objednavka o ON o.id_vezne = k.id_vezne
GROUP BY k.jmeno, v.jmeno
ORDER BY pocet_objednavek DESC;
```

- **Predikát EXISTS (1 dotaz)**
  - Dotaz vybere pomocí predikátu EXISTS pouze takové věznice, kde nejsou vězni ubytováni na čtyřlůžkách.

```
SELECT veznice.jmeno
FROM veznice
WHERE NOT EXISTS(
SELECT vezen.jmeno
FROM vezen
WHERE veznice.id_veznice = vezen.id_veznice
AND vezen.typ_cely = 'ctyrluzko');
```

- **Predikát IN (1 dotaz)**
  - Dotaz pomocí predikátu IN vybere všechny vězně z vnořeného dotazu, který spojí tabulku vězňů a objednávek a vybere všechny vezně, jejichž objednávky mají být doručeny v březnu 2019. Časový interval lze libolně měnit

```
SELECT vezen.jmeno
FROM vezen
WHERE vezen.id_vezne IN(
SELECT id_vezne
FROM vezen NATURAL JOIN objednavka
WHERE termin_dodani BETWEEN TO_DATE('1.3.2019', 'dd.mm.yyyy')
AND TO_DATE('31.3.2019', 'dd.mm.yyyy'));
```

6 Pokročilé objekty databáze

V našem projektu jsme vytvořili dva triggery. Jeden na indexování a jeden na kontrolu dat. Dále užitečné procedury pro případné uživatele systému, materializovaný pohled a detailní plán dotazu.

6.1 Triggery

- **check_rodne_cislo** – Tento trigger vždy zkontroluje rodné číslo, při vkládání nového zaměstnance do databáze. Zkontrolován je základní formát rodného čísla pomocí REGEXP, poté jsou zkontrolovány měsíce a dny a dále je zkontrolována dělitelnost číslem 11.
- **seq_id_zamestnance** – Tento trigger čísluje ID při vkládání jednotlivých zaměstnanců.

6.2 Procedury

- **procenta_veznu_ve_veznici** – Tato procedura přijímá jako argument ID věznice, jejíž procentuální podíl na celkovém počtu vězňů má být spočítán. Pomocí kurzoru jsou vybráni všichni vězni. Čítač počtu všech vězňů je inkrementován vždy, zatímco čítač vězňů ve věznici je inkrementován jen tehdy, když se ID věznice, ve které je vezeň vězněn, shoduje s požadovaným ID věznice. Obě hodnoty jsou poté poděleny a výsledek je získán v procentech. V případě, že nejsou žádní vězni ve věznicích je ošetřena vyjímka, při které by došlo k dělení nulou.
- **zkontroluj_vezne** – Tato procedura přijímá jako argumenty limit objednávek a dva datumy, které tvoří datové rozmezí. Pomocí dvou kurzorů jsou vybrány všechny objednávky a všichni vězni a pro každého vězně se projdou všechny objednávky a v případě, že se shoduje příjemce objednávky, právě vybraný vězeň a datum dodání spadá do vybraného intervalu, je inkrementován čítač objednávek vězně. Pokud tento čítač převýšší stanovený limit, je tento vězeň přesunut na samotku.

6.3 Udělení práv

Pro možnost splnění úkolu s materializovaným pohledem bylo nutno, aby jeden z dvojice vytvořil databázi a udělil druhému práva na přístup do databáze pomocí příkazu GRANT.

6.4 EXPLAIN PLAN

Na demonstraci EXPLAIN PLAN jsme použili následující dotaz, který sečte ceny za jednotlivé objednávky pro jednotlivé vězně.

```
SELECT jmeno, SUM(cena) AS cena_celkem
FROM vezen JOIN objednavka ON vezen.id_vezne = objednavka.id_vezne
GROUP BY jmeno;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                      | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |             |     5 |   120 |     7  (29)| 00:00:01 |
|   1 |  HASH GROUP BY                 |             |     5 |   120 |     7  (29)| 00:00:01 |
|   2 |   MERGE JOIN                   |             |     5 |   120 |     6  (17)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| VEZEN       |     6 |   102 |     2   (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN            | PK_VEZEN    |     6 |       |     1   (0)| 00:00:01 |
|*  5 |    SORT JOIN                   |             |     5 |    35 |     4  (25)| 00:00:01 |
|   6 |     TABLE ACCESS FULL          | OBJEDNAVKA  |     5 |    35 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

V tomto výpisu z PLAN EXPLAIN můžeme vidět, že do celé tabulky objednávek se muselo přistupovat, nicméně potřebujeme z ní pouze dva údaje. Tento přístup lze vylepšit zavedením indexu do tabulky objednávek a to konkrétně na sloupce cena a id vězně. Index je definován následujícím příkazem.

```
CREATE INDEX index_objednavka_vezen ON objednavka(cena, id_vezne);
```

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                       |     5 |   120 |     5  (40)| 00:00:01 |
|   1 |  HASH GROUP BY                |                       |     5 |   120 |     5  (40)| 00:00:01 |
|   2 |   MERGE JOIN                  |                       |     5 |   120 |     4  (25)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| VEZEN                 |     6 |   102 |     2   (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN           | PK_VEZEN              |     6 |       |     1   (0)| 00:00:01 |
|*  5 |    SORT JOIN                  |                       |     5 |    35 |     2  (50)| 00:00:01 |
|   6 |     INDEX FULL SCAN           | INDEX_OBJEDNAVKA_VEZEN |     5 |    35 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------
```

Druhý výpis PLAN EXPLAIN již ukazuje změnu ve způsobu přístupu do tabulky objednávky. Nepřistupuje do celé, ale pouze na indexy a tím celý proces zrychluje. Zrychlení není samozřejmě při takto malém množství dat nijak patrné a projevilo by se ve větších databázích.

6.5 Materializovaný pohled

Slouží pro rychlý přístup při častém žádání o pohled. V našem příkladu jsme zvolili pohled na dotaz, který zjišťuje počet zaměstnanců pro jednotlivé typy úvazku. Materializovanému pohledu jsme nastavili tyto atributy:

- **CACHE** – optimalizuje čtení pohledu
- **BUILD IMMEDIATE** – pohled se naplní hned po vytvoření
- **REFRESH FAST ON COMMIT** – aktualizuje pohled dle logu master tabulek
- **ENABLE QUERY REWRITE** – bude používán optimalizátor

7 Závěr

Vypracování projektu až na ojedinělé případy nebylo složité a jeho vypracování nám dalo cenné zkušenosti s prací s databázemi. Ve výsledku máme funkční model databáze, který by ovšem ještě pro reálně nasazení musel být doplněn o další omezení na datech. Bonusové rozšíření ve formě klinstské aplikace jsme zvažovali, nicméně s docházejícím časem jsme se nakonec rozhodli tuto část vypustit.