VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Databázové systémy

Řetězec multikin

29. dubna 2019

Ondřej Studnička, Václav Trampeška

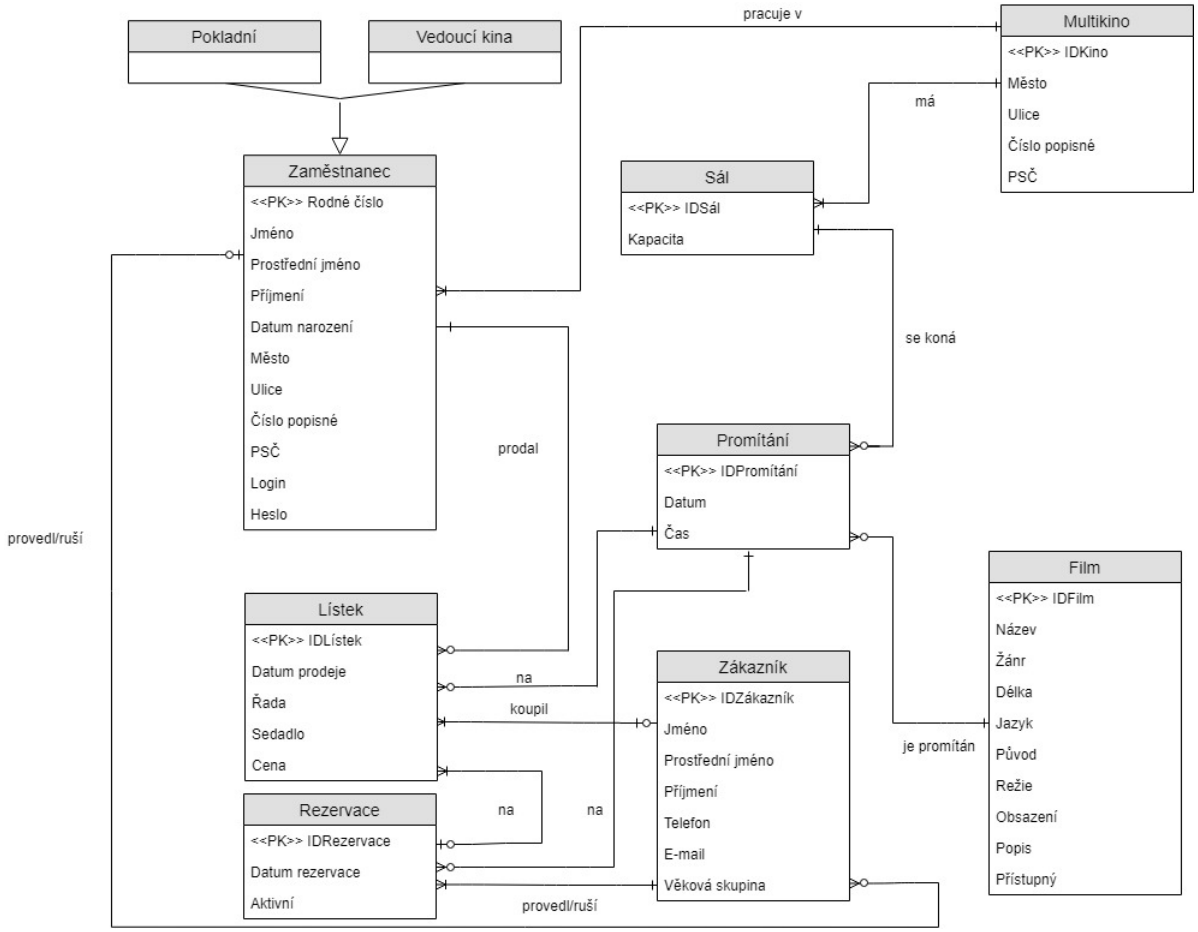
Obsah

1 Úvod

Tato dokumentace se zabývá řešením projektu do předmětu IDS. Jelikož se projekt řeší ve dvojicích, řešili jsme práci na projektu tak, že jsme se společně pravidelně scházeli a projekt vypracovávali společně, díky čemuž jsme byli schopni rychleji opravovat chyby a tím i rychleji zvládnout zadané úkoly.

2 Zadání

Vytvořte návrh informačního systému pro řetězec multikin. Řetězec vlastní několik multikin a každé multikino obsahuje několik promítacích sálů. V sálech probíhají projekce filmů. Systém musí umožnit klientům přípojeným přes webové rozhraní vyhledávání projekcí podle názvu filmu, žánru, kina, apod. Systém umožní klientovi zarezervovat si na zvolenou projekci konkrétní sedadla (jeden klient si může zarezervovat i více sedadel)(není třeba uvažovat překročení kapacity sedadel promítacího sálu), či dříve zadanou rezervaci zrušit. Cena za vstupenku se liší podle toho, zda je divák dítě, mládež, dospělý, či důchodce, zda se jedná o dopolední, odpolední, či večerní představení, atd. Pokladní musí mít možnost prodat vstupenku, ať už při koupi na místě, tak i na základě webové rezervace. Není třeba pamatovat si, který prodavač prodal kterou vstupenku. Vedoucí pracovníci jednotlivých poboček mají možnost zjistit tržby jednotlivých multikin, filmů, apod. Zákazníci si mohou přes webové rozhraní prohlížet program kin.



Obrázek 1: ER diagram

3 Řešení

3.1 Převedení diagramu na relační databázi

V modelování entity jsme postupovali podle upraveného ER diagramu z prvního odevzdání, který vycházel z návrhu v předmětu IUS z minulého ročníku. Při převádění jsme narazili na několik úskalí, které nás donutily relační databázi oproti ER diagramu mírně upravit. Například jsme smazali atribut `tržba` v tabulce `promítání`, neboť tržbu jsme schopni vypočítat z cen lístků, které jsou na promítání pomocí vazby napojené. Jelikož je při vytváření rezervace zaměstnancem nutné k rezervaci přidat i konkrétního zákazníka, museli jsme také zrušit vztah mezi tabulkou `zaměstnanec` a `rezervace` a nahradit ho vztahem mezi tabulkami `zaměstnanec` a `zákazník` . Další změnou bylo přidání atributu `aktivní` do tabulky `rezervace`, který se nastavuje podle toho, zda je rezervace momentálně platná, nebo je zrušená, či vypršená.

Generalizaci jsme v našem případě vyřešili tak, že jsme zaměstnance rozdělili podle zadání na dva druhy – pokladního a vedoucího. V relační databázi je tato generalizace řešena tak, že je v tabulce `zákazník` přidán atribut `vedoucí`, který je v případě zaměstnance nastaven na `null` a v případě vedoucího na hodnotu `'v'`. Jelikož generalizací vzniklé entity nemají vlastní atributy, jevilo se nám toto řešení jako optimální.

Převod zbytku ER diagramu už probíhal bez problému a díky tomu, že se v něm nevyskytují žádné M:N vztahy nebylo ani potřeba vytvářet další tabulky pro tyto vztahy. Finální ER diagram najdete výše v obrázku 1.

3.2 Triggery

Naše implementace obsahuje dva triggery. První zajišťuje automatické generování primárního klíče u tabulky `promítání`. Trigger využívá sekvenci a pomocí inkrementace získává s každým zavoláním nový indetifikátor. Druhý trigger zajišťuje, že do databáze nebude přidáno promítání, které časově koliduje již s jiným promítáním ve stejném sálu. V triggeru tedy najdeme všechny promítání v sále, k začátku promítaní přičteme trvání promítaného filmu a zjišťujeme případnou kolizi.

3.3 Procedury

V projektu jsou implementovány dvě procedury:

- Procedura `vekove_skupiny_zakazniku_procentualne()` vyhledá všechny zákazníky v databázi, spočítá zastoupení jednotlivých věkových kategorií a vypíše na výstup jejich procentuální zastoupení. Tato procedura využívá jeden kurzor, ošetření výjimek a použití proměnné s datovým typem odkazujícím se na řádek či typ sloupce tabulky.

- Procedura `celkova_trzba_jednotlivych_filmu()` spočítá a vypíše celkovou tržbu všech filmů v databázi ve všech kinech. Za pomocí tří kurzorů procedura nejprve projde každý film, u každého filmu každé jeho promítání a u každého promítaní všechny lístky, kterých cenu sčítá do proměnné, kterou následně společně s názvem příslušícího filmu vypíše. Procedura také používá proměnnou s datovým typem odkazujícím se na řádek či typ sloupce tabulky

3.4 Explain plan a Index

Pomocí příkazu `EXPLAIN PLAN` můžeme nechat databázi vypsat informace o zpracování dotazu a vyvolat optimalizátor. Optimalizátor implemetujeme pomocí příkazu `INDEX`. V našem podání vypadá takto:

```
CREATE INDEX promitani_index ON promitani ( id_promitani, datum, cas );
```

Pro demonstraci příkazu `EXPLAIN PLAN` používáme tento výběr:

```
EXPLAIN PLAN FOR
    SELECT promitani.id_promitani AS ID,
        promitani.datum AS DATUM,
        TO_CHAR(promitani.cas, 'hh24:mi') AS CAS,
        AVG(listek.cena) AS prumerna_cena_na_osobu

    FROM promitani, listek

    WHERE listek.id_promitani = promitani.id_promitani

    GROUP BY promitani.id_promitani, promitani.datum,
        TO_CHAR(promitani.cas, 'hh24:mi');

SELECT * FROM TABLE(dbms_xplan.display());
```

V tomto výběru často využíváme tabulku `promítání` , je tedy výhodné vytvořit index právě pro tuto tabulku.
Při první zavolání příkazu `EXPLAIN PLAN`, tedy ještě před zavedením optimalizátoru, jsou vypsána tato data:

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		4	244	7 (15)	00:00:01
1	HASH GROUP BY		4	244	7 (15)	00:00:01
* 2	HASH JOIN		4	244	6 (0)	00:00:01
3	TABLE ACCESS FULL	LISTEK	4	104	3 (0)	00:00:01
4	TABLE ACCESS FULL	PROMITANI	19	665	3 (0)	00:00:01

Po zavedení optimalizátoru a opětovném zalování `EXPLAIN PLAN` je na první pohled znatelná změna a optimalizátor tedy byl využit.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		4	244	5 (20)	00:00:01
1	HASH GROUP BY		4	244	5 (20)	00:00:01
* 2	HASH JOIN		4	244	4 (0)	00:00:01
3	TABLE ACCESS FULL	LISTEK	4	104	3 (0)	00:00:01
4	INDEX FULL SCAN	PROMITANI_INDEX	19	665	1 (0)	00:00:01

Při porovnání obou tabulek si můžeme všimnout toho, že ceny jednotlivých operací se snížily a příkaz tedy byl optimalizován. Na druhou stranu si ale databáze vyžádala větší výkon procesoru pro vykonání operace.

3.5 Materializovaný pohled

Materializovaný pohled slouží na uložení často využívaného pohledu lokálně na disk pro rychlý přístup při opakovaném žádání o tento pohled. Implementovali jsme pohled patřící druhému členu týmu, který využívá tabulky nadefinované prvním členem.

Používáme materializovaný pohled pro zobrazení počtů multikin v jednotlivých městech. Místo nutnosti vyhledávat ve vzdálené tabulce se využívá právě pohled.

4 Závěr

Projekt jsme vytvářeli a testovali na serverech Oracle pomocí programu SQL Developer. K vypracování jsme využili znalosti z přednášek a demonstračních cvičení předmětu IDS a dokumentaci Oracle SQL.