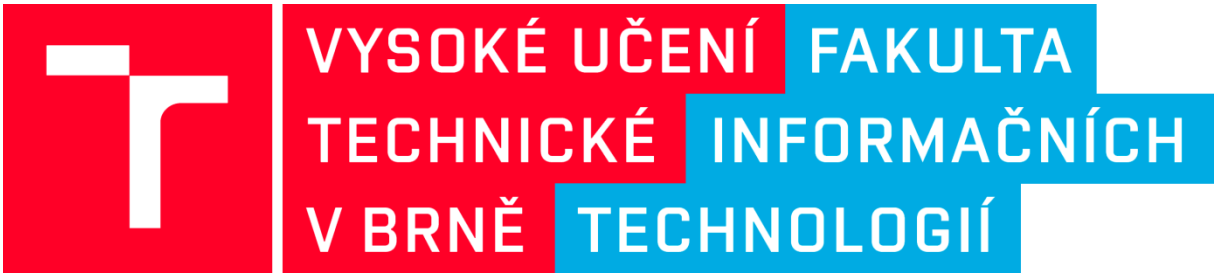
Vysoké učení technické v Brně
Fakulta informačních technologií



Databázové systémy

Dokumentace popisující schéma databáze

Projekt – 5. část

1. května 2016

Nikola Valešová (xvales02)
Ondřej Valeš (xvales03)

Zadání

Knihovna

„Vytvořte informační systém knihovny, který umožňuje čtenářům vyhledávat jednotlivé tituly podle autora/ů, názvu, ISBN, žánru, klíčových slov, atd. Každý titul může být v knihovně ve více výtiscích a může spadat do více žánrů (naučná, historická literatura, beletrie,…). U každého autora je nutno uchovat základní informace (jméno, příjmení, datum narození/úmrtí, jeho literární styl-žánr, ...). Systém bude umožňovat spravovat informace jak o zaměstnancích knihovny, tak o jejich čtenářích. Systém umožňuje čtenářům rezervovat si jednotlivé tituly. Čtenář si může vypůjčit či rezervovat více titulů najednou, přičemž rezervace se provádí na titul, zatímco výpůjčka na konkrétní výtisk.“

Vytvoření tabulek

Námi vytvořený ER diagram jsme podle pravidel prezentovaných na přednášce převedli na schéma databáze v BCNF. Všechny tabulky jsme poté vytvořili pomocí několika příkazů `CREATE TABLE`. V této části skriptu bylo třeba specifikovat primární klíče a zakázat nevyplnění určitých sloupců – především primárních klíčů, ale i dalších esenciálních údajů. Tabulka *titul* navíc obsahuje dvě speciální omezení definovaná příkazem `ADD CONSTRAINT`, a to na hodnoty čísel ISBN a ISSN, která mají přesně daná pravidla dělitelnosti součtu všech cifer. Všechny tabulky jsou vzájemně provázány pomocí cizích klíčů odkazujících primární klíč jiné tabulky.

Vztah generalizace/specializace

Námi vytvořený datový model databáze obsahuje jeden vztah generalizace/specializace. Entitní množina *titul* má dvě specializace, a to *knihu* a *časopis*. S ohledem na to, že je daný vztah generalizace/specializace disjunktní i totální, jsme jako metodu implementace zvolili možnost umístění všech údajů do jedné tabulky *titul*. Tu jsme rozšířili o atribut typ, který určuje, o kterou specializaci se jedná a nabývá hodnot 1 (pro knihu) nebo 2 (pro časopis). Každá specializace má pouze jeden atribut, díky tomu nedochází k přílišné redundanci dat.

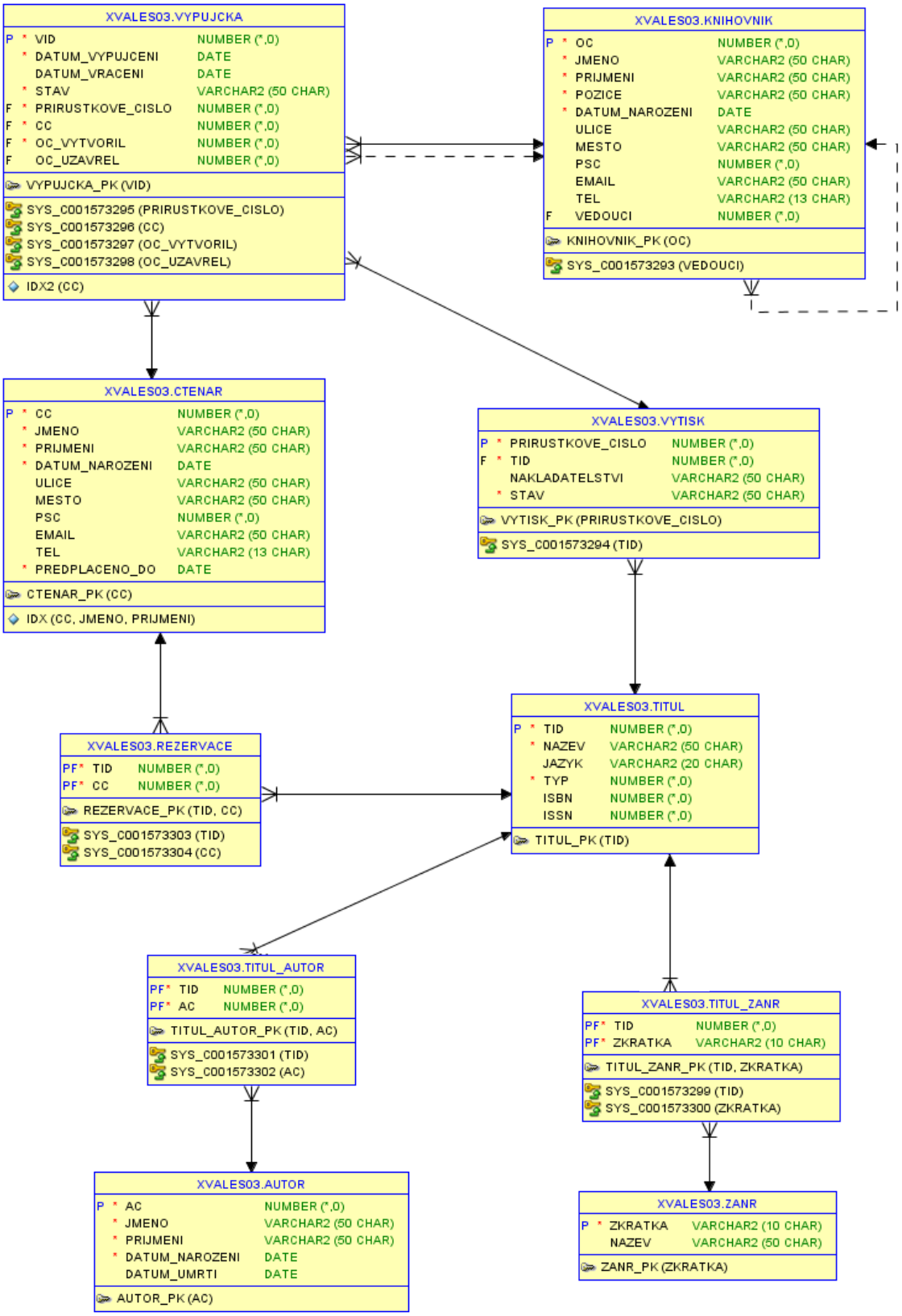
Vytvoření sekvencí

V další části skriptu jsme s využitím příkazu `CREATE SEQUENCE` vytvořili sekvence pro každou tabulku, která má celočíselný primární klíč a je u něj vhodné využít možnosti automatické inkrementace. Jako počáteční hodnotu jsme nastavili číslo 1, maximální pak 1025, jelikož pro naše účely je tento rozsah více než dostačující.

Vložení dat

V dalším kroku jsme vytvořené tabulky naplnili ukázkovými daty za pomoci příkazu `INSERT`. Pro využití automatického doplnění primárního klíče pomocí sekvence jsme využili příkaz `nextval`, pro zadání data narození nebo data vytvoření výpůjčky bylo potřeba použít funkci `TO_DATE` doplněnou o formát zadávaného údaje.

Schéma databáze



Databázové triggery

Dalším úkolem bylo vytvořit databázové triggery. Náš skript obsahuje celkem dva triggery, `zmen_stav_datum` a `nastav_prirustkove_cislo`. První z nich slouží k uzavření výpůjčky. V tabulce *vypujcka* změní její stav na „vraceno“ a do data uzavření výpůjčky vloží dnešní datum. Tento trigger je vyvolán před aktualizací hodnoty ve sloupci *OC_uzavrel*, tedy před vložením identifikačního čísla knihovníka, který danou výpůjčku uzavírá. Příklad použití tohoto triggeru spočívá v aktualizaci výpůjčky s identifikačním číslem 1 – nastavení identifikačního čísla knihovníka, který danou výpůjčku uzavřel, na 1.

	VID	DATUM_VYPUJCENI	DATUM_VRACENI	STAV	PRIRUSTKOVE_CISLO	CC	OC_VYTVORIL	OC_UZAVREL
1	1	28.02.16	(null)	vypujceno	1	2	2	(null)
2	3	12.03.16	(null)	vypujceno	7	2	3	(null)
3	5	04.04.16	(null)	vypujceno	8	4	3	(null)
4	6	15.04.16	(null)	vypujceno	9	5	2	(null)
5	7	26.04.16	(null)	vypujceno	10	6	1	(null)

Údaje v tabulce *výpůjčka* před uzavřením první výpůjčky

	VID	DATUM_VYPUJCENI	DATUM_VRACENI	STAV	PRIRUSTKOVE_CISLO	CC	OC_VYTVORIL	OC_UZAVREL
1	1	28.02.16	29.04.16	vraceno	1	2	2	1
2	3	12.03.16	(null)	vypujceno	7	2	3	(null)
3	5	04.04.16	(null)	vypujceno	8	4	3	(null)
4	6	15.04.16	(null)	vypujceno	9	5	2	(null)
5	7	26.04.16	(null)	vypujceno	10	6	1	(null)

Údaje v tabulce *výpůjčka* po jejím uzavření

Druhý trigger s názvem `nastav_prirustkove_cislo` je příkladem triggeru pro automatické generování hodnot primárního klíče tabulky ze sekvence. V našem případě se jedná o tabulku *výtisk* a hodnoty jsou generovány ze sekvence *seq_pc*. Tento trigger je vyvolán před vložením údajů do této tabulky a do vkládaného záznamu automaticky doplní následující hodnotu přírůstkového čísla. Předvedení tohoto triggeru probíhá vložením nového výtisku do databáze s nevyplněným primárním klíčem, který je vyplněn pomocí triggeru.

	PRIRUSTKOVE_CISLO	TID	NAKLADATELSTVI	STAV
1	1	1	Dobrovsky	dobry
2	2	1	Dobrovsky	dobry
3	3	1	Dobrovsky	poskozeno
4	4	2	Dobrovsky	novy
5	5	2	Dobrovsky	vyrazeno
6	6	2	Dobrovsky	vyrazeno
7	7	3	Scholastic	dobry
8	8	5	Odeon	dobry
9	9	6	Kosmas	dobry
10	10	7	Odeon	dobry

	PRIRUSTKOVE_CISLO	TID	NAKLADATELSTVI	STAV
1	11	5	Odeon	dobry
2	1	1	Dobrovsky	dobry
3	2	1	Dobrovsky	dobry
4	3	1	Dobrovsky	poskozeno
5	4	2	Dobrovsky	novy
6	5	2	Dobrovsky	vyrazeno
7	6	2	Dobrovsky	vyrazeno
8	7	3	Scholastic	dobry
9	8	5	Odeon	dobry
10	9	6	Kosmas	dobry
11	10	7	Odeon	dobry

Údaje v tabulce *výtisk* před a po použití triggeru

Procedury

Náš skript také obsahuje dvě procedury včetně následného předvedení jejich použití. První procedura s názvem `odstran_stare` slouží k odstranění všech výpůjček, které byly uzavřeny před datem, které bylo zadáno jako parametr. V této proceduře je využita proměnná s datovým typem `%ROWTYPE` a kurzor. Její předvedení je potom vyvoláno pomocí příkazu `ODSTRAN_STARE(SYSDATE);` a způsobí, že všechny výpůjčky uzavřené do dnešního data jsou odstraněny z databáze.

	VID	DATUM_VYPUJCENI	DATUM_VRACENI	STAV	PRIRUSTKOVE_CISLO	CC	OC_VYTVORIL	OC_UZAVREL
1	1	28.02.16	29.04.16	vraceno	1	2	2	1
2	2	28.01.16	21.02.16	vraceno	3	1	1	2
3	3	12.03.16	(null)	vypujceno	7	2	3	(null)
4	4	12.08.14	01.09.14	vraceno	1	2	3	2
5	5	04.04.16	(null)	vypujceno	8	4	3	(null)
6	6	15.04.16	(null)	vypujceno	9	5	2	(null)
7	7	26.04.16	(null)	vypujceno	10	6	1	(null)

Údaje v tabulce *výpůjčka* před použitím procedury

	VID	DATUM_VYPUJCENI	DATUM_VRACENI	STAV	PRIRUSTKOVE_CISLO	CC	OC_VYTVORIL	OC_UZAVREL
1	3	12.03.16	(null)	vypujceno	7	2	3	(null)
2	5	04.04.16	(null)	vypujceno	8	4	3	(null)
3	6	15.04.16	(null)	vypujceno	9	5	2	(null)
4	7	26.04.16	(null)	vypujceno	10	6	1	(null)

Údaje v tabulce *výpůjčka* po použití procedury

Druhá procedura má jméno `rezervuj` a slouží k vytvoření nového řádku v tabulce *rezervace*. Jejími vstupními parametry jsou identifikační číslo knihy, kterou si chce čtenář rezervovat, a čtenářské číslo zákazníka, který rezervaci požaduje. Její součástí je i ošetření výjimek při neúspěšném vložení kvůli neplatným vstupním údajům. Demonstrace jejího využití probíhá příkazem `REZERVUJ(3,3);`, který by měl vytvořit novou rezervaci čtenáře s číslem 3 na titul s identifikačním číslem 3.

	TID	CC
1	1	3
2	2	3
3	3	1
4	3	5

Údaje v tabulce *rezervace* před použitím procedury

	TID	CC
1	1	3
2	2	3
3	3	1
4	3	3
5	3	5

Údaje v tabulce *rezervace* po použití procedury

`EXPLAIN PLAN` a použití indexu

Klauzule `EXPLAIN PLAN` slouží k zobrazení plánu, jak databáze zpracovává zadaný příkaz. Pro optimalizaci SQL příkazu pomocí této klauzule jsme zvolili příkaz `SELECT` zobrazující kolik výpůjček již provedl čtenář s členským číslem 2. Tento příkaz využívá tabulky *čtenář* a *výpůjčka*.

```
SELECT jmeno, prijmeni, COUNT(datum_vypujceni)
pocet_vypujcek
FROM CTENAR LEFT JOIN VYPUJCKA USING (CC)
WHERE CC = 2
GROUP BY CC, jmeno, prijmeni;
```

Nejprve se vykoná `EXPLAIN PLAN` nad tímto příkazem `SELECT` a poté se pomocí příkazu `SELECT plan_table_output FROM table(dbms_xplan.display('plan_table', null, 'advanced'));` zobrazí výsledný plán vytvořený optimalizátorem.

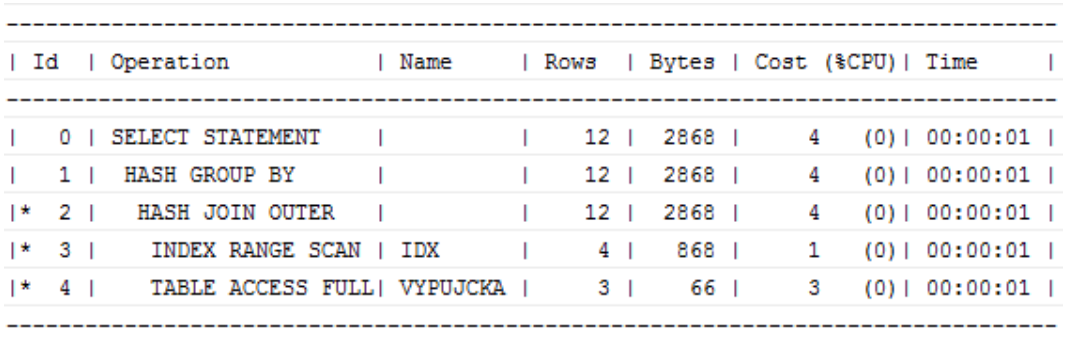
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		3	717	5 (0)	00:00:01
1	HASH GROUP BY		3	717	5 (0)	00:00:01
2	NESTED LOOPS OUTER		3	717	5 (0)	00:00:01
3	TABLE ACCESS BY INDEX ROWID	CTENAR	1	217	2 (0)	00:00:01
* 4	INDEX UNIQUE SCAN	SYS_C001591388	1		1 (0)	00:00:01
* 5	TABLE ACCESS FULL	VYPUJCKA	3	66	3 (0)	00:00:01

Výpis plánu optimalizátoru před vytvořením indexu

Na nultém řádku je `SELECT STATEMENT`, který představuje provedení zkoumaného příkazu `SELECT`. `HASH GROUP BY` na prvním řádku reprezentuje použití této funkce na seskupení podle čtenářského čísla a jména a příjmení čtenáře. Dále je využita metoda NESTED LOOPS OUTER, která spojí tabulky a vrátí všechny řádky, které vyhovují podmínce spojení a také některé z řádků jedné tabulky, které se nepodařilo s žádným řádkem druhé tabulky spojit. Na třetím řádku je přístup k datům pomocí `ROWID`, které reprezentují fyzickou adresu dat. Dále byl použit `INDEX UNIQUE SCAN`, který tabulky prochází přes B-strom a kontroluje rovnost nad sloupci s unikátními hodnotami. Tabulka *výpůjčka* byla prohledána pomocí přístupu `TABLE ACCES FULL`, tedy řádek po řádku bez jakýchkoliv indexů.

Abychom snížili cenu provedení celého příkazu, vytvořili jsme index *idx* v tabulce *čtenář* na sloupce *CC* (číslo čtenáře), *jméno* a *příjmení*. Po aplikaci indexu jsme znovu použili příkaz na vypsání plánu optimalizátoru.

```
CREATE INDEX idx ON ctenar (CC, jmeno, prijmeni);
```



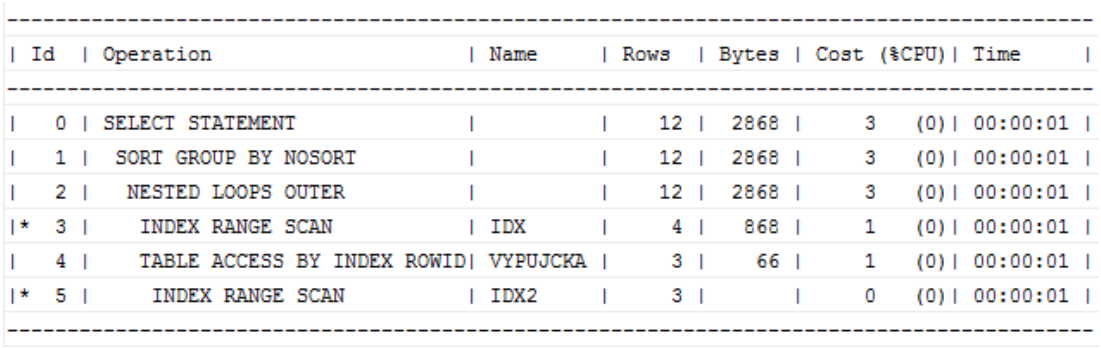
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		12	2868	4 (0)	00:00:01
1	HASH GROUP BY		12	2868	4 (0)	00:00:01
* 2	HASH JOIN OUTER		12	2868	4 (0)	00:00:01
* 3	INDEX RANGE SCAN	IDX	4	868	1 (0)	00:00:01
* 4	TABLE ACCESS FULL	VYPUJCKA	3	66	3 (0)	00:00:01

Výpis plánu optimalizátoru po vytvoření indexu

V druhém a třetím sloupci stojí za povšimnutí, že dříve se musela procházet tabulka *čtenář* celá řádek po řádku, zatímco při druhém průchodu bylo užito vytvořeného indexu. Při pohledu do sloupce `Cost` lze zjistit, že se snížil odhad zdrojů potřebných k vykonání příkazu. Ve sloupcei `Bytes` vidíme, že se zvýšil odhadovaný počet bytů dat, ke kterým se bude během operace přistupovat. Stejně tak se zvýšila i hodnota ve sloupci `Rows`, který udává počet řádků, které jsou očekávány na výstupu tohoto příkazu. Na zhoršení těchto dvou parametrů má vliv režie spojená s indexem a jeho způsobem uložení dat. Vzhledem k malému počtu údajů v naší databázi se optimalizace pomocí indexu neprojevila tolik výrazně, avšak v reálné databázi s rozsáhlejšími daty by výhody jeho použití byly významnější.

Další možnosti optimalizace jsou možné například v tabulce *výpůjčka*, která se stále prochází celá. Vytvořili jsme proto v této tabulce druhý index s názvem *idx2* na sloupec *CC* (číslo čtenáře) a opět si nechali zobrazit plány optimalizátoru.

```
CREATE INDEX idx2 ON vypujcka (CC);
```



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		12	2868	3 (0)	00:00:01
1	SORT GROUP BY NOSORT		12	2868	3 (0)	00:00:01
2	NESTED LOOPS OUTER		12	2868	3 (0)	00:00:01
* 3	INDEX RANGE SCAN	IDX	4	868	1 (0)	00:00:01
4	TABLE ACCESS BY INDEX ROWID	VYPUJCKA	3	66	1 (0)	00:00:01
* 5	INDEX RANGE SCAN	IDX2	3		0 (0)	00:00:01

Cena provedení příkazu se opět snížila a je patrné, že dochází k využití obou indexů při přístupu k tabulkám. Za pomocí indexů tedy došlo k celkem výraznému poklesu ceny provedení příkazu.

Vytvoření přístupových práv

Naším předposledním úkolem bylo vytvoření přístupových práv i pro druhého uživatele. Jelikož byl již druhý uživatel v databázi definován, nebylo třeba jej explicitně přidávat a stačilo využít příkazu `GRANT ALL`, případně `GRANT ALL PRIVILEGES` postupně pro všechny tabulky a ostatní objekty v databázi.

Materializovaný pohled

Skript obsahuje jeden materializovaný pohled. K jeho vytvoření jsme použili příkaz `CREATE MATERIALIZED VIEW`. Námi vytvořený materializovaný pohled slouží k zobrazení všech čtenářů, kteří mají aktuálně nějaké neuzavřené výpůjčky. Využili jsme zde dvou klauzulí, a to `BUILD IMMEDIATE` pro okamžité vytvoření pohledu a `REFRESH COMPLETE` pro obnovení celého obsahu při vyvolání příkazu sloužícímu k obnovení pohledu. Pro vytvoření materializovaného pohledu druhým uživatelem jsme použili příkaz `CONNECT` *`login/heslo`*. Stejně tak se musí připojit druhý uživatel pro aktualizaci nebo zrušení tohoto pohledu. V částech skriptu, kde je připojený druhý uživatel, je potřeba k objektům databáze přistupovat přes login prvního uživatele, který je vytvořil a je jejich vlastníkem.

Použití materializovaného pohledu probíhá sekvencí příkazů k zobrazení jeho obsahu. Poté se provede vložení dalších údajů do jedné z tabulek, které materializovaný pohled využívá, a opětovné zobrazení obsahu pohledu, přičemž v tento okamžik zůstává pohled nezměněn. Následuje příkaz `EXECUTE DBMS_SNAPSHOT.REFRESH('ctenari_neuzavreno','COMPLETE')`, který aktualizuje obsah pohledu a nakonec je naposledy vypsán jeho obsah, ve kterém se již zobrazí nově vložený řádek.

	JMENO	PRIJMENI
1	Jan	Kopal
2	Pavel	Prihradsky
3	Klara	Dusanova
4	Monika	Ondrackova

Schéma materializovaného pohledu před vložením nového řádku

	JMENO	PRIJMENI
1	Jan	Kopal
2	Marie	Prihradska
3	Pavel	Prihradsky
4	Klara	Dusanova
5	Monika	Ondrackova

Schéma materializovaného pohledu po vložení nového řádku a aktualizaci obsahu