VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

DATABÁZOVÉ SYSTÉMY I

Sbírka úloh ke cvičení včetně řešení

Petr Lukáš, Peter Chovanec, Radim Bača

20. října 2020

Obsah

Úvod

Tento studijní materiál slouží pro výuku dotazování v jazyce SQL. Materiál je rozdělen celkem do pěti okruhů, které odpovídají látce probírané na jednotlivých cvičeních předmětu Databázové systémy 1 (DS1). Každý okruh obsahuje přibližně 30 příkladů. První okruh se věnuje základnímu použití příkazu SELECT, druhý se zabývá především spojováním tabulek, třetí používáním agregačních funkcí, čtvrtý množinovými operacemi a poslední, pátý, poddotazy a složitějšími dotazy. Studentům jsou k dispozici dvě verze tohoto materiálu: verze bez řešení a verze s řešením včetně slovního vysvětlení. Na cvičení studenti pracují výhradně s verzí bez řešení. Verze s řešením je pak vhodná pro samostatnou přípravu na test.

Prosíme studenty, aby případné nedostatky (nejednoznačná zadání, chyby v řešení, nejasnosti v popisu řešení nebo jiné návrhy na zlepšení) reportovali na jednu z e-mailových adres: `petr.lukas@vsb.cz`, `peter.chovanec@vsb.cz` nebo `radim.baca@vsb.cz`. Přispějete tím ke zkvalitnění výuky pro další studenty. Zároveň děkujeme všem studentům, kteří již odhalili předchozí nedostatky.

Jmenovité poděkování za odhalení chyb patří následujícím studentům:
Matěj Šimko, Alfons Václavík, Borek Tuleja

Databáze Sakila

V následujících cvičeních předmětu DS1 budeme pracovat s databází fiktivní filmové půjčovny pojmenovanou jako Sakila. Tato databáze byla původně navržena pro demonstraci příkladů nad databázovým systémem MySQL[1]. Postupem času však vznikly porty i pro jiné systémy[2] jako třeba Microsoft SQL Server, který my budeme používat. Skripty pro vytvoření struktury databáze a naplnění daty naleznete na stránkách předmětu na `dbedu.cs.vsb.cz`. Pro potřeby předmětu byl obsah databáze mírně modifikován, aby bylo možné lépe demonstrovat některé rysy jazyka SQL, tzn. aby dotazy např. nevracely prázdný výsledek nebo aby vracely zjevně nesprávný výsledek v případě použití nesprávné konstrukce.

Relační datový model

Strukturu relační databáze obvykle znázorňujeme tzv. E-R (Entity-Relationship) diagramem. E-R diagram databáze Sakila vidíme na Obrázku 1. Doporučujeme si obrázek vytisknout, jelikož do něj budeme, alespoň ze začátku, nahlížet velmi často.

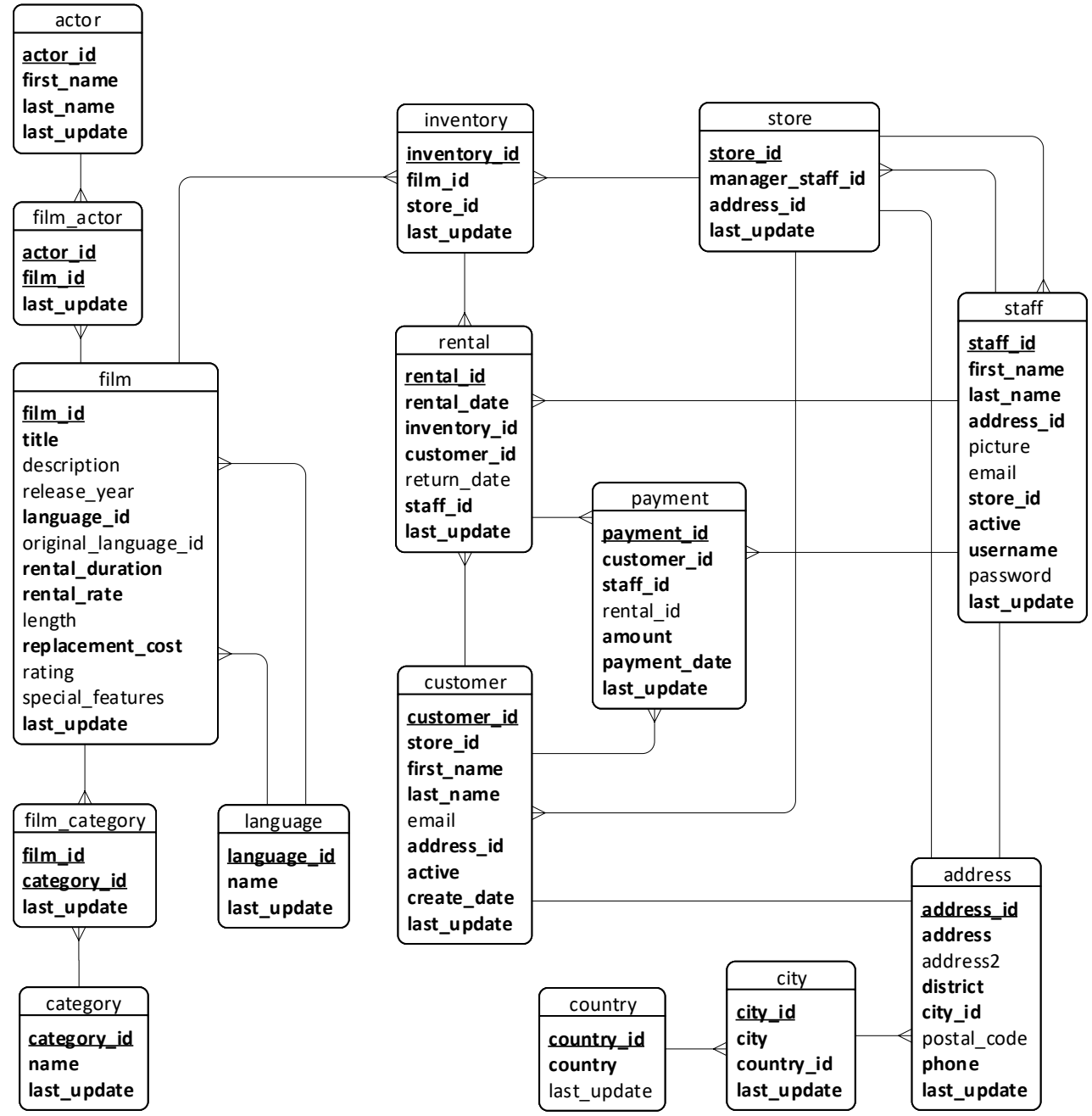
Na obrázku vlevo uprostřed se nachází tabulka `film`, tedy tabulka filmů, která je pomocí vazební tabulky `film_actor` propojena s tabulkou `actor` obsahující herce. Mezi filmy a herci je tedy vazba M:N, což znamená, že v jednom filmu může hrát více herců a jeden herec může hrát ve více filmech. Podobně je tomu s filmovými kategoriemi v tabulce `category`, které jsou s filmy propojeny vazební tabulkou `film_category`. Jeden film tedy může být zařazen do více kategorií (horror, komedie apod.) a naopak. Film je dále ve vazbě N:1 propojen s tabulkou `language` představující jazyky. Vazbu mezi filmy a jazyky vidíme hned dvakrát. První výskyt představuje vazbu na skutečný jazyk filmu, druhý vazbu na originální jazyk v případě, že byl film dabován.

Pokračujme dále na tabulku `inventory` představující jednotlivé kopie filmů. Půjčovna totiž může jeden film vlastnit ve více kopiích. Mezi filmem a kopií je přirozeně vazba 1:N. Následuje tabulka `rental` představující jednotlivé výpůjčky. Výpůjčka se vždy týká nějaké kopie filmu, provedl ji nějaký zákazník v tabulce `customer` a vyřídil ji nějaký zaměstnanec v tabulce `staff`. Mezi tabulkou `rental` a tabulkami `inventory`, `customer` a `staff` jsou tedy vazby N:1. Dále si všimněme tabulky `payment` představující platby. Platbu vždy provedl nějaký zákazník v tabulce `customer` a zpracoval ji nějaký zaměstnanec v tabulce `staff`. Platba se pak může ale také nemusí vztahovat k výpůjčce. Platby, které se nevztahují k výpůjčce mohou představovat např. předplatné.

V databázi dále nalezneme tabulky `country`, `city` a `address`, které na sebe postupně navazují vazbami 1:N, tzn. v jednom státu se nachází více měst a v jednom městě nalezneme více adres. Adresa je pak vazbou 1:1 navázaná na zákazníka, sklad nebo zaměstnance. Každá adresa se váže výlučně na jeden záznam z těchto tří tabulek, tzn. buď na zákazníka, nebo na sklad, nebo na zaměstnance.

[1] `https://dev.mysql.com/doc/sakila/en/`

[2] `https://github.com/jOOQ/jOOQ/tree/master/jOOQ-examples/Sakila`



primární klíč
povinný atribut
nepovinný atribut

Obrázek 1: E-R diagram databáze Sakila

Datový slovník

Přestože jsou názvy tabulek i atributů v databázi Sakila většinou samopopisné, uvádíme zde pro pořádek kompletní datový slovník. Datový slovník není nutné podrobně studovat, můžete však do něj nahlédnout, kdykoli si nebudete jisti významem některého z atributů.

NULL informace, zda je atribut nepovinný, tzn. zda může nabývat hodnoty `NULL`
PK informace, zda jde o primární klíč
FK informace, zda jde o cizí klíč

RENTAL
tabulka výpůjček

sloupec	datový typ	NULL	PK	FK	popis
rental_id	celé číslo	ne	ano	ne	automaticky generovaný PK
rental_date	datum a čas	ne	ne	ne	datum zahájení výpůjčky
inventory_id	celé číslo	ne	ne	ano	ID vypůjčené kopie filmu
customer_id	celé číslo	ne	ne	ano	ID zákazníka, který výpůjčku provedl
return_date	datum a čas	ano	ne	ne	čas vrácení, pokud není vyplněno, jde o běžící výpůjčku
staff_id	celé číslo	ne	ne	ano	ID zaměstnance
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace záznamu

ACTOR
tabulka herců

sloupec	datový typ	NULL	PK	FK	popis
actor_id	celé číslo	ne	ano	ne	automaticky generovaný PK
first_name	řetězec, max. 45 znaků	ne	ne	ne	křestní jméno
last_name	řetězec, max. 45 znaků	ne	ne	ne	příjmení
last_update	datum a čas	ne	ne	ne	datum a čas poslední aktualizace

COUNTRY
tabulka států

sloupec	datový typ	NULL	PK	FK	popis
country_id	celé číslo	ne	ano	ne	automaticky generovaný PK
country	řetězec, max. 50 znaků	ne	ne	ne	název státu
last_update	datum a čas	ano	ne	ne	čas poslední aktualizace záznamu

CITY
tabulka měst

sloupec	datový typ	NULL	PK	FK	popis
city_id	celé číslo	ne	ano	ne	automaticky generovaný PK
city	řetězec, max. 50 znaků	ne	ne	ne	název města
country_id	celé číslo	ne	ne	ano	ID státu
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace záznamu

ADDRESS
tabulka adres

sloupec	datový typ	NULL	PK	FK	popis
address_id	celé číslo	ne	ano	ne	automaticky generovaný PK
address	řetězec, max. 50 znaků	ne	ne	ne	první řádek adresy
address2	řetězec, max. 50 znaků	ano	ne	ne	nepovinný druhý řádek adresy
district	řetězec, max. 20 znaků	ne	ne	ne	region
city_id	celé číslo	ne	ne	ano	ID města
postal_code	řetězec, max. 10 znaků	ano	ne	ne	poštovní směrovací číslo
phone	řetězec, max. 20 znaků	ne	ne	ne	telefon
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace záznamu

LANGUAGE

tabulka jazyků

sloupec	datový typ	NULL	PK	FK	popis
language_id	celé číslo	ne	ano	ne	ID jazyka
name	řetězec, max. 20 znaků	ne	ne	ne	název jazyka
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace záznamu

CATEGORY

tabulka filmových kategorií

sloupec	datový typ	NULL	PK	FK	popis
category_id	celé číslo	ne	ano	ne	automaticky generovaný PK
name	řetězec, max. 25 znaků	ne	ne	ne	název kategorie
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace záznamu

CUSTOMER

tabulka zákazníků

sloupec	datový typ	NULL	PK	FK	popis
customer_id	celé číslo	ne	ano	ne	automaticky generovaný PK
store_id	celé číslo	ne	ne	ano	ID skladu
first_name	řetězec, max. 45 znaků	ne	ne	ne	křestní jméno
last_name	řetězec, max. 45 znaků	ne	ne	ne	příjmení
email	řetězec, max. 50 znaků	ano	ne	ne	e-mail
address_id	celé číslo	ne	ne	ano	ID adresy
active	řetězec, max. 1 znaků	ne	ne	ne	příznak, zda je zákazník aktivní
create_date	datum a čas	ne	ne	ne	čas vytvoření záznamu
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace záznamu

FILM

tabulka filmů

sloupec	datový typ	NULL	PK	FK	popis
film_id	celé číslo	ne	ano	ne	automaticky generovaný PK
title	řetězec, max. 255 znaků	ne	ne	ne	název filmu
description	text	ano	ne	ne	popis nebo stručný obsah filmu
release_year	řetězec, max. 4 znaků	ano	ne	ne	rok vydání
language_id	celé číslo	ne	ne	ano	ID jazyka
original_language_id	celé číslo	ano	ne	ano	ID originálního jazyka v případě dabovaného filmu
rental_duration	celé číslo	ne	ne	ne	standardní délka výpůjčky ve dnech
rental_rate	des. číslo	ne	ne	ne	částka za výpůjčku trvající standardní délku
length	celé číslo	ano	ne	ne	délka filmu v minutách
replacement_cost	des. číslo	ne	ne	ne	částka za náhradu v případě ztráty nebo poškození média
rating	řetězec, max. 10 znaků	ano	ne	ne	klasifikace filmu MPAA
special_features	řetězec, max. 255 znaků	ano	ne	ne	speciální vlastnosti
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace záznamu

FILM_ACTOR

vazební tabulka mezi herci a filmy

sloupec	datový typ	NULL	PK	FK	popis
actor_id	celé číslo	ne	ano	ano	ID herce
film_id	celé číslo	ne	ano	ano	ID filmu
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace záznamu

FILM_CATEGORY

vazební tabulka mezi kategoriemi a filmy

sloupec	datový typ	NULL	PK	FK	popis
film_id	celé číslo	ne	ano	ano	ID filmu
category_id	celé číslo	ne	ano	ano	ID kategorie
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace

INVENTORY

tabulka kopií filmů, jeden film může půjčovna vlastnit ve více kopiích

sloupec	datový typ	NULL	PK	FK	popis
inventory_id	celé číslo	ne	ano	ne	automaticky generovaný PK
film_id	celé číslo	ne	ne	ano	ID filmu
store_id	celé číslo	ne	ne	ano	ID skladu
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace záznamu

STAFF

tabulka zaměstnanců půjčovny

sloupec	datový typ	NULL	PK	FK	popis
staff_id	celé číslo	ne	ano	ne	automaticky generovaný PK
first_name	řetězec, max. 45 znaků	ne	ne	ne	křestní jméno
last_name	řetězec, max. 45 znaků	ne	ne	ne	příjmení
address_id	celé číslo	ne	ne	ano	ID adresy
picture	image	ano	ne	ne	fotografie
email	řetězec, max. 50 znaků	ano	ne	ne	e-mail
store_id	celé číslo	ne	ne	ano	ID skladu
active	bit	ne	ne	ne	příznak, zda jde o aktivního zaměstnance
username	řetězec, max. 16 znaků	ne	ne	ne	uživatelské jméno
password	řetězec, max. 40 znaků	ano	ne	ne	heslo zašifrované pomocí SHA1
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace záznamu

STORE

tabulka skladů

sloupec	datový typ	NULL	PK	FK	popis
store_id	celé číslo	ne	ano	ne	automaticky generovaný PK
manager_staff_id	celé číslo	ne	ne	ano	ID zaměstnance – správce skladu
address_id	celé číslo	ne	ne	ano	ID adresy
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace

PAYMENT

tabulka plateb, platba se nemusí vždy vztahovat jen k výpůjčce

sloupec	datový typ	NULL	PK	FK	popis
payment_id	celé číslo	ne	ano	ne	automaticky generovaný PK
customer_id	celé číslo	ne	ne	ano	ID zákazníka – plátce
staff_id	celé číslo	ne	ne	ano	ID zaměstnance, který platbu zpracoval
rental_id	celé číslo	ano	ne	ano	ID výpůjčky
amount	des. číslo	ne	ne	ne	částka
payment_date	datum a čas	ne	ne	ne	čas platby
last_update	datum a čas	ne	ne	ne	čas poslední aktualizace záznamu

1 Základy SQL, příkaz SELECT

Na tomto cvičení se zaměříme na základní syntaxi příkazu SELECT. Ve všech dotazech budeme pracovat vždy jen s jednou tabulkou, přičemž si vyzkoušíme jednoduchou selekci, projekci, skládání logických podmínek, základní funkce pro práci s datem a textem a tzv. agregační funkce.

1. Vypište e-mailové adresy všech neaktivních zákazníků.

```
SELECT email
FROM customer
WHERE active = 0
```

   Úloha demonstruje selekci a projekci, což jsou dva základní operátory tzv. relační algebry, která stojí na pozadí zpracování každého SQL dotazu. Selekce (někdy také nazývaná jako restrikce) omezuje řádky, projekce omezuje sloupce. V tomto případě požadujeme pouze řádky, kde atribut `active` z tabulky `customer` má hodnotu 0. Z takových řádků pak vybíráme jen sloupec `email`.

   Zůstaňme ještě chvíli u syntaxe zápisu dotazu. Vidíme, že dotaz se skládá z tzv. klauzulí `SELECT`, `FROM` a `WHERE`. Později uvidíme, že se v dotazu mohou nacházet ještě klauzule `GROUP BY`, `HAVING` a `ORDER BY`. Každá z uvedených klauzulí má svůj specifický význam. Měli bychom také umět rozlišovat pojmy „klauzule `SELECT`“ označující prostor mezi klíčovými slovy `SELECT` a `FROM` a „příkaz SELECT“, který představuje celý dotaz. Všimněme si odlišné sazby písma pro označení klauzule.

2. Vypište názvy a popisy všech filmů s klasifikací (atribut `rating`) G. Výstup bude setřízen sestupně podle názvu filmu.

```
SELECT title, description
FROM film
WHERE rating = 'G'
ORDER BY title DESC
```

   Řešení této úlohy demonstruje setřízení výsledku pomocí klauzule `ORDER BY`. Přidáním klíčového slova `DESC` (z anglického „descending“) bude třízení probíhat sestupně. Opakem k `DESC` je klíčové slovo `ASC` (ascending), které je však možné vynechat. Všimněme si, že na rozdíl od předchozí úlohy musíme konstantu G zapsat mezi apostrofy. Prostor mezi apostrofy v SQL označuje řetězcové konstanty. Apostrofy bychom naopak neměli používat k zápisu číselných konstant. Pozor, v SQL má svůj význam i používání uvozovek, které však neslouží k zápisu řetězcových konstant.

3. Vypište všechny údaje o platbách, které proběhly v roce 2006 nebo později a částka byla menší než 2.

```
SELECT *
FROM payment
WHERE payment_date >= '2006-01-01' AND amount < 2
```

   Tato úloha demonstruje jednoduché skládání podmínek pomocí logického součinu, který je v SQL reprezentován klíčovým slovem `AND`. Všimněme si, že datum je v této úloze zapsáno jako řetězcová konstanta ve tvaru „rok–měsíc–den“. Tento způsob datumového zápisu je specifický pro Microsoft SQL Server. U jiných systémů může být zápis data

mírně odlišný. Pomocí symbolu * za `SELECT` říkáme, že chceme vybrat všechny sloupce (tj. neprovádíme žádnou projekci).

4. Vypište popisy všech filmů klasifikovaných jako G nebo PG.

```
SELECT description
FROM film
WHERE rating = 'G' OR rating = 'PG'
```

   Oproti předchozí úloze zde vidíme použití logického součtu, tj. klíčového slova `OR`.

5. Vypište popisy všech filmů klasifikovaných jako G, PG nebo PG-13.

```
SELECT description
FROM film
WHERE rating IN ('G', 'PG', 'PG-13')
```

   Tuto úlohu bychom samozřejmě mohli vyřešit přidáním dalšího `OR` k řešení předchozí úlohy. Takový zápis by ale byl zbytečně zdlouhavý. Proto v SQL existuje speciální konstrukce `IN`, kterou se ptáme, zda určitý atribut (nebo obecně výraz) spadá do nějaké množiny. V dalších cvičeních uvidíme, že tato množina nemusí být vyjádřená pouze statickým výčtem konstant.

6. Vypište popisy všech filmů, které nejsou klasifikovány jako G, PG nebo PG-13.

```
SELECT description
FROM film
WHERE rating NOT IN ('G', 'PG', 'PG-13')
```

   Abychom měli sadu logických spojek kompletní, zde vidíme použití negace, tj. klíčového slova `NOT`. Obvykle se v jiných programovacích jazycích píše symbol negace (ať už je jakýkoli) před podmínku, kterou chceme negovat. Tzn. možná bychom spíš čekali zápis `NOT (rating IN (...))`. Takový zápis by byl v SQL zcela v pořádku, nicméně v kombinaci s `IN` je zápis v ukázkovém řešení výše běžnější.

7. Vypište všechny údaje filmů, jejichž délka přesahuje 50 minut a doba výpůjčky je 3 nebo 5 dní.

```
SELECT *
FROM film
WHERE length > 50 AND (rental_duration = 3 OR rental_duration = 5)
```

   Řešení této úlohy demonstruje kombinování spojek `AND` a `OR`. Všimněme si použití závorek za `AND`. Pokud bychom na závorky zapomněli, vyhodnocoval by se nejdříve logický součin `length > 50 AND rental_duration = 3` a až potom `OR rental_duration = 5`, což by nebylo správně. Pamatujme tedy na to, že `AND` má vyšší prioritu než `OR`, a kdykoli si prioritou nejsme 100 % jisti, použijme závorky.

8. Vypište názvy filmů, které obsahují „RAINBOW“ nebo začínají na „TEXAS“ a jejich délka přesahuje 70 minut. Zamyslete se nad nejednoznačností formulace této úlohy v přirozeném jazyce.

```
SELECT title
FROM film
WHERE
  (title LIKE '%RAINBOW%' OR title LIKE 'TEXAS%')
  AND length > 70
```

Podobně jako v předchozí úloze kombinujeme použití `OR` a `AND`. Ze zadání však není úplně jednoznačné, zda máme podmínku interpretovat jako:

```
(title LIKE '%RAINBOW%' OR title LIKE 'TEXAS%') AND length > 70
```

nebo jako

```
title LIKE '%RAINBOW%' OR (title LIKE 'TEXAS%' AND length > 70)
```

Obě varianty řešení by tedy mohly být považovány za správné, nicméně musíme si být naprosto jisti, co jedním nebo druhým zápisem myslíme.

Tato úloha navíc demonstruje použití operátoru `LIKE`, který slouží k porovnání atributu (nebo obecně výrazu) s regulárním výrazem, kde symbol „%“ reprezentuje libovolný počet libovolných znaků.

9. Vypište názvy všech filmů, v jejichž popisu se vyskytuje „And“, jejich délka spadá do intervalu 80 až 90 minut a standardní doba výpůjčky (atribut `rental_duration`) je liché číslo.

```
SELECT title
FROM film
WHERE
  description LIKE '%And%' AND length BETWEEN 80 AND 90
  AND rental_duration % 2 = 1
```

Tato úloha demonstruje použití konstrukce `BETWEEN`. Ekvivalentním zápisem k `length BETWEEN 80 AND 90` je `length >= 80 AND length <= 90`. Zápis s `BETWEEN` je však úspornější a v praxi se používá častěji. Všimněme si, že klíčové slovo `AND` mezi konstantami 80 a 90 neznamená logický součin, ale je součástí konstrukce `BETWEEN`. Tato úloha navíc ukazuje použití operátoru %, který v SQL představuje operaci modulo, tj. zbytek po dělení, pomocí kterého snadno poznáme lichá čísla.

10. Vypište vlastnosti (atribut `special_features`) všech filmů, kde částka za náhradu škody (atribut `replacement_cost`) je v intervalu 14 až 16. Zajistěte, aby se vlastnosti ve výspiu neopakovaly. Setřiďte vybrané vlastnosti abecedně. Zamyslete se, proč je výsledek i bez explicitního požadavku na setřízení již abecedně setřízený.

```
SELECT DISTINCT special_features
FROM film
WHERE replacement_cost BETWEEN 14 AND 16
ORDER BY special_features
```

V této úloze opět vidíme použití `BETWEEN`. Novým prvkem zde však je modifikátor `DISTINCT`, který zajistí, že všechny opakující se řádky budou z výsledku vynechány. Pozor, pokud bychom pomocí dotazu vybírali více sloupců, budeme `DISTINCT` psát pouze jednou za klíčové slovo `SELECT`. `DISTINCT` je modifikátor celého dotazu a nevztahuje se k jednotlivým sloupcům, jak by se možná mohlo na první pohled zdát.

Odebereme-li klauzuli `ORDER BY`, s největší pravděpodobností zjistíme, že je výsledek stále abecedně setřízen. Souvisí to s tím, jakým způsobem databázový systém eliminuje duplicitní výsledky (což požadujeme použitím `DISTINCT`). Není totiž nic jednoduššího než setřídit si všechny výsledky abecedně. Pak už lze duplicity detekovat velmi snadno. Pamatujme si ale, že pokud v dotazu neuvedeme klauzuli `ORDER BY`, pak je čistě a jen na databázovém systému, v jakém pořadí nám výsledky vrátí.

11. Vypište všechny údaje filmů, jejichž standardní doba výpůjčky je menší než 4 dny, nebo jsou klasifikovány jako PG. Nesmí však splňovat obě podmínky zároveň.

```
SELECT title
FROM film
WHERE
  rental_duration < 4 AND rating != 'PG' OR
  rental_duration >= 4 AND rating = 'PG'
```

Úloha řeší problém tzv. exkluzivního logického součtu. V některých programovacích jazycích pro toto existuje speciální operátor XOR (eXclusive OR). V SQL však takovýto operátor nemáme a podmínku musíme rozepsat pomocí `AND` a `OR`. Všimněme si, že narozdíl od úloh 7 a 8 zde nemusíme používat závorky, jelikož skutečně požadujeme, aby se podmínky spojené pomocí `AND` vyhodnotily prioritně. Dokázali byste exkluzivní logický součet vyjádřit pomocí `AND` a `OR` i jiným způsobem?

12. Vypište všechny údaje o adresách, které mají vyplněno PSČ.

```
SELECT *
FROM address
WHERE postal_code IS NOT NULL
```

V databázích se velmi často setkáme s konstantou `NULL` představující prázdný nebo nevyplněný atribut. Pro práci s touto konstantou je zavedena tzv. tříhodnotová logika, kterou se zde zatím zabývat nemusíme. Zapamatujme si ale, že kdykoli chceme provést test na hodnotu `NULL`, používáme speciální konstrukci `IS NULL` nebo `IS NOT NULL`. Zápis `postal_code != NULL` by nebyl správně. Co by bylo výsledkem?

13. Vypište ID všech zákazníků, kteří aktuálně mají vypůjčený nějaký film. Dokázali byste spočítat, kolik takových zákazníků je?

```
SELECT DISTINCT customer_id
FROM rental
WHERE return_date IS NULL
```

Oproti předchozí úloze zde naopak požadujeme záznamy, kde atribut `return_date` je nevyplněný, tj. použijeme podmínku `return_date IS NULL`. Výpůjčky s nevyplněným `return_date` jsou právě ty, které prozatím nebyly vráceny. Získáme-li z těchto výpůjček ID zákazníků, získáme ty zákazníky, kteří mají aktuálně vypůjčený nějaký film. Jelikož může mít zákazník ve výpůjčce více filmů, je vhodné použít klíčové slovo `DISTINCT`, aby se ID zákazníků ve výsledku neopakovala. Jestliže použijeme klíčové slovo `DISTINCT`, bude počet řádků ve výsledku reprezentovat počet zákazníků, kteří mají vypůjčený nějaký film.

14. Pro každé ID platby vypište v samostatných sloupcích rok, měsíc a den, kdy platba proběhla. Sloupce vhodně pojmenujte.

```
SELECT payment_id, YEAR(payment_date) AS rok, MONTH(payment_date) AS mesic
    , DAY(payment_date) AS den
FROM payment
```

Na řešení tohoto příkladu vidíme, že se za `SELECT` nemusí nacházet jen výčet atributů, ale obecně výčet výrazů. V tomto případě jde o funkce `YEAR()`, `MONTH()` a `DAY()`, jejichž význam je zřejmý. Nutno však upozornit, že tyto tři funkce mohou mít v jiných databázových systémech jiný ekvivalent. Všimněme si dále pojmenování sloupců pomocí `AS`. Dotaz by sice byl spustitelný i s nepojmenovanými sloupci, nicméně naučme se sloupce pečlivě pojmenovávat. Bude se nám to hodit později při skládání dotazů, kde bude pojmenování v určitých případech dokonce nezbytné.

15. Vypište filmy, jejichž délka názvu není 20 znaků.

```
SELECT *
FROM film
WHERE LEN(title) != 20
```

Na úloze vidíme demonstraci použití textové funkce `LEN()`, která vrací délku řetězce. Opět jen upozorňujeme, že v jiném databázovém systému může mít funkce jiný ekvivalent (např. v systému Oracle Database jde o funkci `LENGTH()`).

16. Pro každou ukončenou výpůjčku (její ID) vypište dobu jejího trvání v minutách.

```
SELECT rental_id, DATEDIFF(minute, rental_date, return_date) AS minuty
FROM rental
WHERE return_date IS NOT NULL
```

Získat rozdíl mezi dvěma datumy je poměrně častá úloha. Na SQL Serveru k tomuto slouží funkce `DATEDIFF()` se třemi parametry: první parametr definuje požadovaný časový úsek (sekunda, minuta, hodina, atd.), další dva parametry představují datum od a do. Ve výsledku požadujeme pouze údaje ukončených výpůjček, takže je potřeba ověřit, zda `return_date` není `NULL`.

17. Pro každého aktivního zákazníka vypište jeho celé jméno v jednom sloupci. Výstup tedy bude obsahovat dva sloupce – `customer_id` a `full_name`.

```
SELECT customer_id, first_name + ' ' + last_name AS full_name
FROM customer
WHERE active = 1
```

Úloha demonstruje skládání řetězců pomocí operátoru „+“. V jiných systémech se ale můžeme setkat i s jiným operátorem (např. v databázi Oracle se ke skládání používá „||“).

18. Pro každou adresu (atribut `address`) vypište PSČ. Jestliže PSČ nebude vyplněno, bude se místo něj zobrazovat text „(prázdné)“.

```
SELECT address, COALESCE(postal_code, '(prázdné)') AS psc
FROM address
```

Tato úloha se vrací ke konstantě `NULL`. Koncovému uživateli bychom měli tuto hodnotu vždy nějak rozumně naformátovat, tzn. místo `NULL` zobrazit např. „(prázdné)“. V SQL naštěstí existuje standardní funkce `COALESCE()`, která má obecně neomezené množství

parametrů. Výsledek funkce je roven prvnímu z parametrů (v pořadí, jak jsou uvedeny), který není NULL.

19. Pro všechny uzavřené výpůjčky vypište v jednom sloupci interval od – do (tj. obě data oddělená pomlčkou), kdy výpůjčka probíhala.

```
SELECT rental_id, CAST(rental_date AS VARCHAR) + ' - ' + CAST(return_date
    AS VARCHAR) AS interval
FROM rental
WHERE return_date IS NOT NULL
```

Tento příklad demonstruje použití přetypování, tedy konstrukce CAST. Přetypování je nutné proto, aby operátor „+“ považoval své argumenty za řetězce (datový typ VARCHAR). V opačném případě by došlo ke „sčítání“ datumu a řetězce, což by vedlo k chybě při zpracování dotazu.

20. Pro všechny výpůjčky vypište v jednom sloupci interval od – do (tj. obě data oddělená pomlčkou), kdy výpůjčka probíhala. Pokud výpůjčka dosud nebyla vrácena, vypište pouze datum od.

```
SELECT rental_id, CAST(rental_date AS VARCHAR) + COALESCE(' - ' + CAST(
    return_date AS VARCHAR), '') AS interval
FROM rental
```

Předchozí úloha byla mírně zjednodušená tím, že jsme uvažovali pouze uzavřené výpůjčky, u nichž bylo zaručeno, že mají vyplněno datum vrácení return_date. V tomto případě však datum vrácení vyplněno být nemusí. Z toho důvodu použijeme funkci COALESCE(), která, v případě, že return_date je NULL, vrátí prázdný řetězec. Přesněji řečeno, funkce COALESCE() je aplikována na zřetězení prefixu „–“ a data vrácení. Pokud je však datum vrácení NULL, bude i výsledek řetězení NULL, tzn. funkce COALESCE() vrátí druhý z parametrů, tedy prázdný řetězec.

21. Vypište počet všech filmů v databázi.

```
SELECT COUNT(*) AS pocet_filmu
FROM film
```

V tomto příkladu se poprvé setkáváme s použitím tzv. agregační funkce. Agregační funkce mají v SQL speciální význam a platí pro ně speciální pravidla, která budou demonstrována podrobněji na dalších cvičeních. V řešení této úlohy jsme použili agregační funkci COUNT, čili počet. Argumentem funkce je symbol *, což znamená, že chceme jednoduše počítat počet řádků ve výsledku – v tomto případě tedy získáme počet filmů.

22. Vypište počet různých klasifikací filmů (atribut rating).

```
SELECT COUNT(DISTINCT rating) AS pocet_kategorii
FROM film
```

Řešení této úlohy se od toho předchozího liší použitím klíčového slova DISTINCT, které zajistí, že budeme počítat pouze unikátní hodnoty atributu rating. Pokud bychom DISTINCT vynechali, byl by výsledekem počet všech hodnot rating včetně hodnot, které se opakují. To ve výsledku znamená, že bychom opět spočítali počet filmů, protože každý film má nějakou klasifikaci.

23. Vypište jedním dotazem počet adres, počet adres s vyplněným PSČ a počet různých PSČ.

```
SELECT
  COUNT(*) AS pocet_celkem,
  COUNT(postal_code) AS pocet_s_psc,
  COUNT(DISTINCT postal_code) AS pocet_psc
FROM address
```

Agregačních funkcí můžeme v jednom dotazu použít více. První `COUNT` vrací počet všech řádků v tabulce `address`. Druhý vrací počet hodnot atributu `postal_code` včetně opakování, tzn. vrací počet adres s vyplněným PSČ. Poslední `COUNT` vrací počet unikátních hodnot PSČ. Druhé použití `COUNT` demonstruje fakt, že agregační funkce přeskakují hodnoty `NULL`.

24. Vypište nejmenší, největší a průměrnou délku všech filmů. Ověřte si zjištěnou průměrnou délku pomocí podílu součtu a počtu.

```
SELECT MIN(length) AS nejmensi, MAX(length) AS nejvetsi, AVG(CAST(length
   AS FLOAT)) AS prumerna
FROM film
```

Zde opět vidíme použití více agregačních funkcí v jednom dotaze. Co by nás mohlo překvapit je, že pokud bychom ve funkci `AVG()` nepoužili přetypování na `FLOAT`, bylo by výsledkem celé číslo. Pro systém Microsoft SQL Server totiž platí podobná logika jako např. v jazyce C++. Podílem dvou celých čísel bude zase celé číslo. Agregační funkce `AVG()` jednoduše dělí součet počtem, kde obě hodnoty představují celá čísla. Pokud tedy chceme správně spočítat průměr z celých čísel, je potřeba argument funkce `AVG()` přetypovat na `FLOAT` (číslo s plovoucí desetinnou čárkou).

25. Vypište počet a součet všech plateb, které byly provedeny v roce 2005.

```
SELECT COUNT(*) AS pocet, SUM(amount) AS soucet
FROM payment
WHERE YEAR(payment_date) = 2005
```

Tento příklad pouze demonstruje použití agregačních funkcí v kombinací s klauzulí `WHERE`. Pamatujme si, že agregační funkce se vyhodnocují až po zpracování `WHERE`.

26. Vypište celkový počet znaků v názvech všech filmů.

```
SELECT SUM(LEN(title))
FROM film
```

Tento příklad ukazuje, že argumentem agregační funkce nemusí být vždy jen atribut, ale obecně jakýkoli výraz. Jak jsme si již ukazovali, funkce `LEN()` vrací délku řetězce. Aplikací agregační funkce `SUM()` na `LEN()` tedy získáme součet délek všech názvů filmů. Pozor, agregační funkce se nikdy nesmí nacházet v jiné agregační funkci, tzn. zápis jako např. `MIN(SUM(atribut))` je nesmyslný. `LEN()` však není agregační funkce, takže zápis `SUM(LEN(title))` je v pořádku.

2 Spojování tabulek

Předchozí cvičení bylo zaměřeno na práci vždy s jedinou tabulkou. Za klauzulí `FROM` se však obvykle nachází tabulek více. Na tomto cvičení si ukážeme, jak tabulky spojovat, přičemž se zaměříme na tzv. vnitřní spojení a levé vnější spojení. Při řešení nebudeme potřebovat žádné agregační funkce, poddotazy nebo konstrukce jako `IN` nebo `EXISTS`. Všechny úkoly bude možné řešit pouze vhodným spojením několika tabulek a případně omezením duplicitních výsledků pomocí klíčového slova `DISTINCT`.

1. Vypište všechny informace o městech včetně odpovídajících informací o státech, kde se tato města nachází.

   ```
   SELECT *
   FROM city JOIN country ON city.country_id = country.country_id
   ```

   Na řešení tohoto příkladu vidíme ukázku tzv. vnitřního spojení, což je nejjednodušší typ spojení dvou tabulek. Před klíčové slovo `JOIN` se u vnitřního spojení může nepovinně psát navíc `INNER`. Vnitřní spojení v tomto případě ke každému řádku v tabulce `city` připojí žádný nebo více řádků z tabulky `country` tak, aby byla splněna podmínka specifikovaná za `ON`. V naprosté většině případů se za `ON` vyskytuje porovnání primárního klíče z jedné tabulky a cizího klíče z druhé tabulky. Výsledkem je tedy „velká“ tabulka obsahující sloupce z `city` i `country`. Jelikož v tomto případě neprovádíme žádnou další projekci (požadujeme všechny sloupce z obou tabulek zápisem `SELECT *`), reprezentuje výsledek celého dotazu i výsledek samotné operace `JOIN`. Protože obě tabulky často obsahují stejně pojmenované atributy, používáme k jejich rozlišení tečkovou konvenci `tabulka.atribut`.

2. Vypište názvy všech filmů včetně jejich jazyka.

   ```
   SELECT film.title, language.name
   FROM film JOIN language ON film.language_id = language.language_id
   ```

   Tato úloha je oproti předchozí mírně rozšířená tím, že provádíme projekci na atributy `film.title` a `language.name`. Výsledek tedy reprezentuje přesně to, co se po nás v zadání požaduje.

3. Vypište ID všech výpůjček zákazníka s příjmením SIMPSON.

   ```
   SELECT rental_id
   FROM rental JOIN customer ON
     rental.customer_id = customer.customer_id
   WHERE customer.last_name = 'SIMPSON'
   ```

   Spojení neprovádíme vždy jen proto, abychom celou „poskládanou“ tabulku zobrazili. Pomocí spojení můžeme podle atributů z jedné tabulky filtrovat záznamy z jiné tabulky. V tomto případě nejprve spojíme odpovídající záznamy z tabulek `rental` a `customer`, z takto poskládaných záznamů vybereme jen ty, které vyhovují podmínce na příjmení, a nakonec provedeme projekci na atribut `rental_id`.

4. Vypište adresu (atribut `address` v tabulce `address`) zákazníka s příjmením SIMPSON. Porovnejte tento příklad s předchozím co do počtu řádků ve výsledku.

```
SELECT address
FROM customer JOIN address ON
  customer.address_id = address.address_id
WHERE customer.last_name = 'SIMPSON'
```

Zatímco v předchozí úloze bylo výsledkem přirozeně více různých ID, jelikož zákazník může mít více výpůjček, v tomto případě je jasné, že vypsaná adresa bude s největší pravděpodobností jen jedna, protože zákazník má vždy jedinou adresu (předpokládáme, že v databázi je jediný SIMPSON).

5. Pro každého zákazníka (jeho jméno a příjmení) vypište adresu bydliště včetně názvu města.

```
SELECT first_name, last_name, address, postal_code, city
FROM
  customer
  JOIN address ON customer.address_id = address.address_id
  JOIN city ON address.city_id = city.city_id
```

Řešení tohoto příkladu demonstruje spojení více než dvou tabulek. Pro spojení obecně n tabulek požijeme operaci `JOIN` obvykle $n - 1$ krát.

6. Pro každého zákazníka (jeho jméno a příjmení) vypište název města, kde bydlí.

```
SELECT first_name, last_name, city
FROM
  customer
  JOIN address ON customer.address_id = address.address_id
  JOIN city ON address.city_id = city.city_id
```

Řešení této úlohy samozřejmě přímo vychází z řešení předchozí úlohy – z klauzule `SELECT` pouze odebereme atributy `address` a `postal_code`. Smyslem je ukázat, že přestože z tabulky `address` žádné atributy ani nevypisujeme, ani nepoužíváme v `WHERE`, musí být tato tabulka přesto obsažena ve `FROM`, aby propojila tabulky `customer` a `city`. Mělo by být jasné, že spojovat můžeme pouze tabulky, které mezi sebou mají vazbu. Spojit tabulky `customer` a `city` bez použití tabulky `address` by nedávalo smysl.

7. Vypište ID všech výpůjček včetně jména zaměstnance, jména zákazníka a názvu filmu.

```
SELECT rental_id, staff.first_name AS staff_first_name,
  staff.last_name AS staff_last_name,
  customer.first_name AS customer_first_name,
  customer.last_name AS customer_last_name,
  film.title
FROM
  rental
  JOIN staff ON rental.staff_id = staff.staff_id
  JOIN customer ON rental.customer_id = customer.customer_id
  JOIN inventory ON rental.inventory_id = inventory.inventory_id
  JOIN film ON inventory.film_id = film.film_id
```

Spojení většího množství tabulek by nás nemělo ničím překvapit. V praxi se můžeme běžně potkat s propojením třeba i několika desítek tabulek.

8. Pro každý film (jeho název) vypište jména a příjmení všech herců, kteří ve filmu hrají. Kolik řádků bude ve výsledku tohoto dotazu?

```
SELECT film.title, actor.first_name, actor.last_name
FROM
  film
  JOIN film_actor ON film.film_id = film_actor.film_id
  JOIN actor ON film_actor.actor_id = actor.actor_id
ORDER BY film.title
```

Mělo by být jasné, že úlohu vyřešíme spojením tabulek `film`, `film_actor` a `actor`. I když to ze zadání přímo nevyplývá, bylo by vhodné výsledek setřídit dle názvu filmu, ať na první pohled vidíme, kteří herci hrají v nějakém určitém filmu. Pro zběžnou kontrolu správnosti řešení bychom si vždy měli uvědomit, jak velký by měl být výsledek (alespoň přibližně). Pokud nepoužíváme selekci (klauzuli `WHERE`), bude počet řádků ve výsledku obvykle odpovídat počtu záznamů v jedné z tabulek. V tomto případě to bude vazební tabulka `film_actor`. Dokázali byste odvodit pravidlo, které dle použitých vazeb určí, která tabulka bude udávat velikost výsledku (předpokládejte pouze povinné atributy v databázi)?

9. Pro každého herce (jeho jméno a příjmení) vypište jména všech filmů, kde herec hrál. Čím se bude tento dotaz lišit od předchozího? Co můžeme říct o operaci vnitřního spojení tabulek?

```
SELECT actor.first_name, actor.last_name, film.title
FROM
  film
  JOIN film_actor ON film.film_id = film_actor.film_id
  JOIN actor ON film_actor.actor_id = actor.actor_id
ORDER BY actor.last_name, actor.first_name
```

Oproti předchozí úloze zde pouze pro pořádek přeuspořádáme atributy za `SELECT` a výsledek setřídíme podle herců, abychom na první pohled pro každého z nich viděli, ve kterých filmech hrál. Část dotazu za `FROM` ale zůstane stejná. Pokud pomineme určité detaily, je operace `JOIN` komutativní, tzn. `A JOIN B` je (téměř) to samé jako `B JOIN A`. Dokázali byste najít sémantický rozdíl mezi oběma zápisy? Pro jistotu jen připomínáme, že bez klauzule `ORDER BY` nezáleží na pořadí řádků ve výsledku.

10. Vypište názvy všech filmů v kategorii „Horror“.

```
SELECT film.title
FROM
  category
  JOIN film_category ON
    category.category_id = film_category.category_id
  JOIN film ON film_category.film_id = film.film_id
WHERE name = 'Horror'
```

Zde pouze vidíme ukázku toho, že i při spojení více tabulek nás nakonec ve výpisu může zajímat jen jediný atribut.

11. Pro každý sklad (jeho ID) vypište jméno a příjmení jeho správce. Dále vypište adresu skladu a adresu správce (u obou adres stačí atribut atribut `address` v tabulce `address`). Řešení dále rozšiřte o výpis adresy včetně názvu města a státu.

Nejprve si tedy ukažme řešení bez názvů měst a států.

```
SELECT store.store_id, store_address.address AS store_address, store_city.
    city AS store_city, store_country.country AS store_country, staff.
    first_name, staff.last_name, staff_address.address AS staff_address,
    staff_city.city AS staff_city, staff_country.country AS staff_country
FROM
  store
  JOIN staff ON
    store.manager_staff_id = staff.staff_id
  JOIN address store_address ON
    store.address_id = store_address.address_id
  JOIN address staff_address ON
    staff.address_id = staff_address.address_id
```

Zde se poprvé setkáváme s tím, že určitou tabulku (konkrétně tabulku `address`) můžeme za `FROM` použít vícekrát. Dotaz pak funguje stejně, jako bychom pracovali se dvěma různými tabulkami se stejným obsahem, pouze musíme obě tabulky vhodně pojmenovat. To se dělá tak, že za název tabulky uvedeme její alias. V tomto případě tedy dotaz pracuje s jakoby různými tabulkami `store_address` a `staff_address` – první uvedená obsahuje adresu skladu a druhá adresu správce.

Dle zadání tedy dotaz dále rozšíříme o výpis města a státu. I v tomto případě je nutné použít obě tabulky `city` a `country` dvakrát – jedou pro města a státy skladů, podruhé pro města a státy správců.

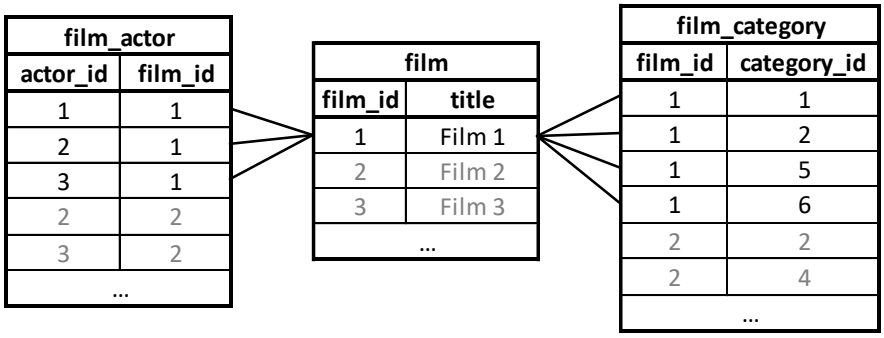
```
SELECT store.store_id, store_address.address AS store_address, store_city.
    city AS store_city, store_country.country AS store_country, staff.
    first_name, staff.last_name, staff_address.address AS staff_address,
    staff_city.city AS staff_city, staff_country.country AS staff_country
FROM
  store
  JOIN staff ON
    store.manager_staff_id = staff.staff_id
  JOIN address store_address ON
    store.address_id = store_address.address_id
  JOIN city store_city ON
    store_address.city_id = store_city.city_id
  JOIN country store_country ON
    store_city.country_id = store_country.country_id
  JOIN address staff_address ON
    staff.address_id = staff_address.address_id
  JOIN city staff_city ON
    staff_address.city_id = staff_city.city_id
  JOIN country staff_country ON
    staff_city.country_id = staff_country.country_id
```

12. Pro každý film (ID a název) vypište ID všech herců a ID všech kategorií, do kterých film spadá. Tzn. napište dotaz, jehož výsledkem bude tabulka s atributy `film_id`, `actor_id` a `category_id`, setřízeno dle `film_id`. Z výsledku pohledem zjistěte, kolik herců hraje ve filmu s `film_id` = 1, kolik tomuto filmu odpovídá kategorií a kolik je pro tento film celkem řádků ve výsledku.

```
SELECT film.film_id, film.title, actor_id, category_id
FROM
  film
  JOIN film_actor ON film_actor.film_id = film.film_id
```

```
  JOIN film_category ON film_category.film_id = film.film_id
ORDER BY film.film_id
```

Řešení této úlohy demonstruje něco, co je v praxi ovykle nežádoucí. Představme si film, ve kterém hrají 3 herci a film patří do 4 kategorií. Pro lepší představu je situace znázorněna na Obrázku 2. Úkolem databázového systému je vždy zobrazit všechny kombinace řádků, které odpovídají specifikovaným podmínkám. V dané situaci mu tedy nezbude nic jiného, než zkombinovat každý záznam v tabulce `film_actor` s každým záznamem v tabulce `film_category` pro určitý film. Měli bychom vědět, že takovému kombinování se říká *kartézský součin* a v databázích se mu většinou potřebujeme vyhnout. Spustíme-li tedy dotaz nad daty na Obrázku 2, budeme mít ve výsledku celkem 12 řádků pro film s ID = 1.



Obrázek 2: Demonstrace nevhodného spojení tabulek

13. Vypište všechny kombinace atributů ID herce a ID kategorie, kde daný herec hrál ve filmu v dané kategorii. Výsledek setřiďte dle ID herce. Dotaz dále rozšiřte o výpis jména a příjmení herce a názvu kategorie.

```
SELECT DISTINCT actor_id, category_id
FROM
  film
  JOIN film_actor ON film_actor.film_id = film.film_id
  JOIN film_category ON film_category.film_id = film.film_id
ORDER BY film_actor.actor_id
```

Vidíme, že řešení první části této úlohy výše je velmi podobné řešení předchozí úlohy, kde byl však výsledek poměrně obtížně interpretovatelný. Zde však vidíme, že při spojení stejných tabulek a vynechání atributů týkajících se filmu bude výsledek představovat relativně smysluplný výstup obsahující ID herců a odpovídající ID filmových kategorií. Důležité je použití `DISTINCT`, jinak bychom měli stejné kombinace herců a filmových kategorií ve výsledku obsažené vícekrát (pokaždé pro jiný film, který nás však nezajímá).

Uživatele v praxi obvykle samotná ID nezajímají, proto dotaz rozšiřme o tabulky `actor` a `category`, abychom vypsali jména a příjmení herců a názvy kategorií:

```
SELECT DISTINCT actor.actor_id, actor.first_name, actor.last_name,
    category.category_id, category.name
FROM
  film
  JOIN film_actor ON film_actor.film_id = film.film_id
```

```
  JOIN film_category ON film_category.film_id = film.film_id
  JOIN actor ON film_actor.actor_id = actor.actor_id
  JOIN category ON film_category.category_id = category.category_id
ORDER BY actor.actor_id
```

14. Vypište jména filmů, které půjčovna vlastní alespoň v jedné kopii.

```
SELECT DISTINCT film.title
FROM film JOIN inventory ON film.film_id = inventory.film_id
```

Řešení tohoto příkladu demonstruje důležitou vlastnost vnitřního spojení. Jestliže pro nějaký film nebude existovat záznam v inventáři, bude takový film z výsledku úplně vynechán. Později uvidíme, že toto je přesně ta vlastnost, kterou se vnitřní spojení liší od toho vnějšího. Dále bychom si měli uvědomit, že film může být v inventáři obsažen vícekrát a proto je nutné omezit duplicitní názvy klíčovým slovem `DISTINCT`.

15. Zjistěte jména herců, kteří hrají v nějaké komedii (kategorie „Comedy“).

```
SELECT DISTINCT actor.actor_id, actor.first_name, actor.last_name
FROM
  film
  JOIN film_actor ON film_actor.film_id = film.film_id
  JOIN actor ON film_actor.actor_id = actor.actor_id
  JOIN film_category ON film_category.film_id = film.film_id
  JOIN category ON film_category.category_id = category.category_id
WHERE category.name = 'Comedy'
```

Řešení této úlohy vychází z řešení úlohy 13. Podobně jako v úlohách 12 a 13, i zde dojde ke kartézskému součinu kategorií a herců pro stejné filmy, nicméně v klauzuli `SELECT` vybíráme sloupce pouze z tabulky `actor` a klíčovým slovem `DISTINCT` zajistíme, že se nám herci ve výsledku nakonec opakovat nebudou.

16. Vypište jména všech zákazníků, kteří pochází z Itálie a někdy měli nebo mají půjčený film s názvem MOTIONS DETAILS.

```
SELECT DISTINCT customer.first_name, customer.last_name
FROM
  customer
  JOIN address ON customer.address_id = address.address_id
  JOIN city ON address.city_id = city.city_id
  JOIN country ON city.country_id = country.country_id
  JOIN rental ON customer.customer_id = rental.customer_id
  JOIN inventory ON rental.inventory_id = inventory.inventory_id
  JOIN film ON inventory.film_id = film.film_id
WHERE country.country = 'Italy' AND film.title = 'MOTIONS␣DETAILS'
```

Základem této úlohy je správně spojit všechny tabulky od `country` přes `customer` až po `film`. V principu pak hledáme výpůjčky filmu MOTIONS DETAILS zákazníkem z Itálie. Jelikož zákazník může mít film ve výpůjčce vícekrát, je nakonec opět nutné použít klíčové slovo `DISTINCT`, které zajistí, že se nám stejný zákazník nebude ve výsledku opakovat pro každou výpůjčku daného filmu.

17. Zjistěte jména a příjmení všech zákazníků, kteří mají aktuálně vypůjčený nějaký film, kde hraje herec SEAN GUINESS.

```
SELECT DISTINCT customer.first_name, customer.last_name
FROM
  actor
  JOIN film_actor ON actor.actor_id = film_actor.actor_id
  JOIN film ON film_actor.film_id = film.film_id
  JOIN inventory ON film.film_id = inventory.film_id
  JOIN rental ON inventory.inventory_id = rental.inventory_id
  JOIN customer ON rental.customer_id = customer.customer_id
WHERE actor.first_name = 'SEAN' AND actor.last_name = 'GUINESS' AND rental
    .return_date IS NULL
```

Řešení této úlohy je velmi podobné předchozí úloze, principiální rozdíl je pouze v podmínce `WHERE`, kde mimo jiné specifikujeme výpůjčky, kde není vyplněno datum vrácení. Nezapomeňme, že pro porovnání s hodnotou `NULL` používáme speciální operátor `IS NULL`.

18. Vypište ID a částku všech plateb a u každé platby uveďte datum výpůjčky, tj. hodnotu atributu `rental_date` v tabulce `rental`. U plateb, které se nevztahují k žádné výpůjčce bude datum výpůjčky `NULL`.

```
SELECT payment.payment_id, payment.amount, rental.rental_date
FROM
  payment
  LEFT JOIN rental ON payment.rental_id = rental.rental_id
```

Zde se poprvé setkáváme s použitím tzv. levého vnějšího spojení, tedy se zápisem `LEFT JOIN` (nepovinně také `LEFT OUTER JOIN`). Levé vnější spojení se od vnitřního spojení liší tím, že do výsledku umístí všechny záznamy v tabulce nebo výrazu před `LEFT JOIN` (tj. na „levé straně“) a to i v případě, že k nim nebudou existovat odpovídající záznamy v tabulce nebo výrazu za `LEFT JOIN`. V tomto konkrétním případě tedy budou vypsány i takové platby, které se nevztahují k žádné výpůjčce, protože mají nevyplněný atribut `payment.rental_id`. Pro takové platby budou všechny atributy z tabulky `rental` nabývat hodnoty `NULL`. Pokud vám vysvětlení pořád není jasné, zkuste z řešení umazat klíčové slovo `LEFT` a porovnejte výsledek.

Analogicky k levému vnějšímu spojení existuje i pravé vnější spojení – `RIGHT JOIN`. Navíc existuje ještě tzv. úplné spojení (`FULL OUTER JOIN`), jehož význam si můžete domyslet, nicméně používá se pouze výjimečně a my se jím zde nebudeme zabývat.

19. Pro každý jazyk vypište názvy všech filmů v daném jazyce. Zajistěte, aby byl jazyk ve výsledku obsažen, i když k němu nebude existovat odpovídající film.

```
SELECT language.name, film.title
FROM
  language
  LEFT JOIN film ON language.language_id = film.language_id
```

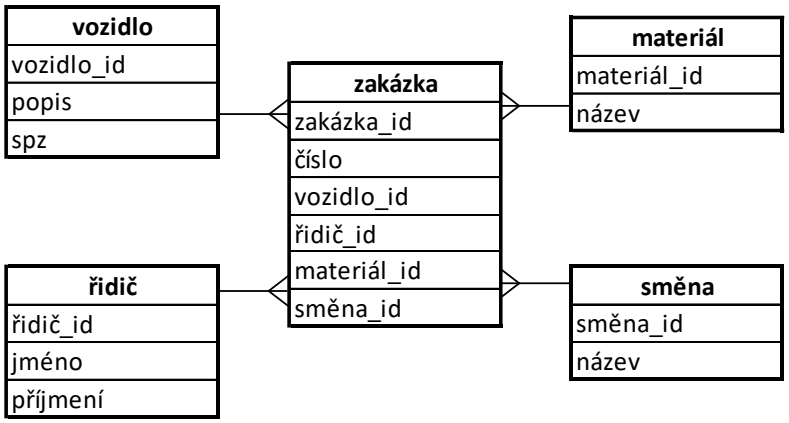
Zadání příkladu opět ukazuje na nutnost použití vnějšího spojení. Pokud bychom místo `LEFT JOIN` použili pouze `JOIN`, nebyly by ve výsledku obsaženy jazyky, pro které neexistují filmy. Už v tuto chvíli stojí za to naznačit, že používání `LEFT JOIN` bude zvlášť užitečné v kombinaci s agregačními funkcemi, pomocí kterých budeme chtít spočítat např. počet filmů pro každý jazyk, přičemž nás budou zajímat i jazyky, kde takový počet bude nulový.

20. Pro každý film (ID a název) vypište jeho jazyk a jeho původní jazyk.

```
SELECT film.film_id, film.title, language.name AS language,
   original_language.name AS original_language
FROM
  film
  JOIN language ON film.language_id = language.language_id
  LEFT JOIN language original_language ON film.original_language_id =
     original_language.language_id
```

Pro spojení tabulek `film` a `language` můžeme použít `JOIN` (tedy vnitřní spojení), protože atribut `film.language_id` je povinný (viz úvodní kapitola Datový slovník). Kdybychom místo prvního `JOIN` použili `LEFT JOIN`, nebyla by to zásadní chyba, nicméně měli bychom si být vědomi toho, že nám `LEFT JOIN` nic navíc nepřinese. Naproti tomu, spojení s `original_language` již musí být vnější, protože atribut `original_language_id` je nepovinný. Pokud bychom použili vnitřní spojení, přišli bychom o všechny filmy, kde `original_language_id` je `NULL`.

Tento příklad je mimo jiné ukázkou tzv. hvězdicového schématu, kde se `LEFT JOIN` obecně typicky využívá. Ve hvězdicovém schématu je jedna „centrální" tabulka (nazývaná jako „tabulka faktů") spojena s větším množstvím „malých" tabulek (nazývaných jako „tabulky dimenzí"). V tomto případě je tabulkou faktů `film` a tabulkami dimenzí `language` a `original_language`. Pro lepší představu na chvíli opusťme naší filmovou databázi a podívejme se na schéma na Obrázku 3. Uvažujme, že atributy `vozidlo_id`, `řidič_id`, `materiál_id` a `směna_id` jsou nepovinné, protože se např. budou doplňovat až během zpracování zakázky. Jistě si umíme představit přehledovou tabulku (např. tiskovou sestavu, dashboard apod.) se sloupci: číslo zakázky, SPZ vozidla, jméno řidiče, název materiálu atd. Jestliže některý z těchto údajů chybí, určitě by to neměl být důvod, proč by zakázka neměla být v přehledové tabulce viditelná. A to je právě ten důvod, proč u hvězdicového schématu často používáme vnější spojení.



Obrázek 3: Ukázka hvězdicového schématu

21. Vypište názvy filmů, které si někdy půjčil zákazník TIM CARY, nebo je jejich délka 48 minut.

```
SELECT DISTINCT film.title
FROM
  film
  LEFT JOIN inventory ON film.film_id = inventory.film_id
  LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
  LEFT JOIN customer ON customer.customer_id = rental.customer_id
WHERE (customer.first_name = 'TIM' AND customer.last_name = 'CARY') OR
   film.length = 48
```

Také v tomto příkladě musíme použít vnější spojení, abychom nepřišli o filmy, které si žádný zákazník nepůjčil. Mezi takovými filmy se totiž mohou nacházet i ty s délkou 48 minut, které dle zadání musí být obsaženy ve výsledku.

Všimněme si dále, jakým způsobem se obvykle zapisuje vícenásobné použití `LEFT JOIN` v jednom dotazu. Začneme tou tabulkou, ze které potřebujeme všechny záznamy a tu pak postupně rozšiřujeme pomocí `LEFT JOIN` o údaje z dalších tabulek, které budou mít v případě nesplnění spojovací podmínky hodnotu `NULL`.

22. Vypište názvy filmů, které půjčovna nevlastní ani v jedné kopii (tj. nejsou obsaženy v inventáři).

```
SELECT film.title, length
FROM
  film
  LEFT JOIN inventory ON film.film_id = inventory.film_id
WHERE inventory.inventory_id IS NULL
```

Vnější spojení můžeme elegantně využít i k řešení množinového rozdílu. Pro záznamy z tabulky `film`, které nemají žádný odpovídající záznam v tabulce `inventory` budou všechny atributy `inventory` nabývat hodnoty `NULL`.

23. Vypište jména a příjmení všech zákazníků, kteří mají nějakou nezaplacenou výpůjčku.

```
SELECT DISTINCT first_name, last_name
FROM
  customer
  JOIN rental ON customer.customer_id = rental.customer_id
  LEFT JOIN payment ON rental.rental_id = payment.rental_id
WHERE payment.payment_id IS NULL
```

Řešení této úlohy je v principu velice podobné předchozí úloze. Musíme si akorát uvědomit, že nezaplacená výpůjčka znamená, že pro ni neexistuje odpovídající záznam o platbě (tedy záznam v tabulce `payment`). Protože zákazník může mít i více nezaplacených výpůjček, měli bychom použít `DISTINCT`, aby nedošlo k opakovanému výpisu stejného zákazníka.

24. U každého názvu filmu vypište jazyk filmu, pokud jazyk začíná písmenem „I“, v opačném případě bude jazyk `NULL`.

```
SELECT film.title, language.name
FROM
  film
  LEFT JOIN language ON film.language_id = language.language_id AND
     language.name LIKE 'I%'
```

V řešení tohoto příkladu se poprvé setkáváme s tím, že spojovací podmínka nemusí vždy obsahovat jen porovnání primárního a cizího klíče. Naším úkolem je provést levé vnější spojení tabulky filmů s tabulkou jazyků, které začínají na „I“. Prvním řešením, které studenty často napadne je provést levé vnější spojení `film LEFT JOIN language` a pak pomocí `WHERE` zobrazit jen jazyky začínající písmenem „I“, tzn.:

```
SELECT film.title, language.name
FROM
  film
  LEFT JOIN language ON film.language_id = language.language_id
WHERE language.name LIKE 'I%'
```

Můžeme se ale snadno přesvědčit, že takový dotaz není správným řešením. `LEFT JOIN` sice zajišťuje, že levá tabulka bude ve výsledku obsažena celá, ale to platí pouze pro výsledek samotné operace vnějšího spojení, nikoli pro výsledek celého dotazu. Navíc každý film musí být v nějakém jazyce (atribut `film.language_id` je povinný), takže výsledek operace `LEFT JOIN` by v tomto případě byl stejný jako výsledek `JOIN`.

Problémem je, že klauzule `WHERE` se logicky zpracovává až po vyhodnocení výrazu v klauzuli `FROM`, tzn. filmy, jejichž jazyk nezačíná na „I“, se z výsledku dotazu stejně nakonec vyřadí. Zkuste si z dotazu odebrat klauzuli `WHERE`, podívejte se na výsledek a odmyslete si z výsledku řádky, které nesplňují podmínku `language.name LIKE 'I%'` – přesně tak funguje `WHERE`.

Přesunutím podmínky z `WHERE` přímo do spojení za `ON` nepříjdeme o výhodu vnějšího spojení a výsledek celého dotazu skutečně bude obsahovat všechny filmy a u nich případně jen jazyky začínající na „I“.

Nakonec ale ještě jedna praktická rada. Pokud to jde (v tomto případě bohužel ne), měla by spojovací podmínka za `ON` skutečně obsahovat jen porovnání atributů ze spojovaných tabulek. Tzn. ostatní podmínky bychom se měli snažit uvést v klauzuli `WHERE`.

25. Pro každého zákazníka vypište ID všech plateb s částkou větší než 9. U zákazníků, kteří takovéto platby nemají, bude `payment_id` rovno `NULL`.

```
SELECT first_name, last_name, payment.payment_id
FROM
  customer
  LEFT JOIN payment ON customer.customer_id = payment.customer_id AND
     payment.amount > 9
```

Princip řešení této úlohy je naprosto stejný jako u předchozí úlohy. Opět rozšíříme spojovací podmínku vnějšího spojení.

26. Pro každou výpůjčku (její ID) vypište název filmu, pokud obsahuje písmeno „U“, a město a stát zákazníka, jehož adresa obsahuje písmeno „A“. Podobně jako v předchozích úlohách – jestliže údaj nesplňuje danou podmínku, bude v příslušném poli uvedeno `NULL`.

```
SELECT rental_id, film.title, city.city, country.country
FROM
  rental
  LEFT JOIN inventory ON rental.inventory_id = inventory.inventory_id
  LEFT JOIN film ON inventory.film_id = film.film_id AND film.title LIKE '
     %U%'
  LEFT JOIN customer ON rental.customer_id = customer.customer_id
```

```
LEFT JOIN address ON customer.address_id = address.address_id AND
   address.address LIKE '%A%'
LEFT JOIN city ON address.city_id = city.city_id
LEFT JOIN country ON city.country_id = country.country_id
```

Tato úloha opět demonstruje rozšíření spojovací podmínky vnějšího spojení, abychom nepřišli o výpis všech výpůjček. Všimněme si, že rozšíření spojovací podmínky používáme mimo jiné u připojení tabulky `address`, na kterou dále vnějším spojením navazujeme tabulky `city` a `country`. To znamená, že jestliže adresa zákazníka nezačíná na „A“, pak nejen, že atributy tabulky `address` budou `NULL`, ale všechny atributy ze všech tabulek, které jsou navázány `address` budou také `NULL`.

27. Vypište všechny dvojice název filmu a příjmení zákazníka, kde si zákazník vypůjčil daný film. Pokud výpůjčka proběhla po datu 1.1.2006, bude příjmení zákazníka nevyplněné (tj. `NULL`). Z výsledku odstraňte duplicitní řádky a setřiďte jej podle názvu filmu.

```
SELECT DISTINCT film.title, customer.last_name
FROM
  film
  JOIN inventory ON film.film_id = inventory.film_id
  JOIN rental ON inventory.inventory_id = rental.inventory_id
  LEFT JOIN customer ON rental.customer_id = customer.customer_id AND
     rental_date <= '2006-01-01'
ORDER BY film.title
```

Pro tabulky `film`, `inventory` a `rental` použijeme vnitřní spojení. Dostaneme tak výpůjčky včetně informací o kopii filmu a o filmu samotném. Zákazníka pak připojíme pomocí levého vnějšího spojení, ve kterém rozšíříme spojovací podmínku tak, abychom zákazníka připojili pouze v případě, že výpůjčka neproběhla po datu 1.1.2006. Na řešení této úlohy je zvláštní fakt, že rozšíření spojovací podmínky se netýká atributu z tabulky `customer`, ale z tabulky `rental`. To však syntaxi SQL nijak nevadí – spojovací podmínka se může odkazovat na jakýkoli atribut z tabulky, která byla připojena dříve.

3 Agregační funkce a shlukování

S agregačními funkcemi jsme se již setkali na prvním cvičení, kde jsme je však používali tak, že výsledkem byl vždy jeden souhrnný řádek obsahující jednu nebo více vypočtených hodnot. Na tomto cvičení si ale ukážeme, že agregace lze počítat i samostatně pro určité skupiny záznamů. Výsledkem agregací tedy nemusí být jediný řádek, ale více řádků, které seskupují záznamy dle určitých podmínek. Začneme jednoduššími příklady nad jednou tabulku, poté však navážeme na znalosti z předchozího cvičení a agregace budeme aplikovat na více spojených tabulek.

1. Vypište počty filmů pro jednotlivé klasifikace (atribut `rating`).

```
SELECT rating, COUNT(*) AS pocet
FROM film
GROUP BY rating
```

   Zde se poprvé setkáváme s klauzulí `GROUP BY`. Jejím úkolem je počítat agregace za nějaké skupiny, konkrétně za jednotlivé klasifikace (hodnoty atributu `rating`). V SQL platí pravidlo, že všechny atributy, které se nachází v klauzuli `SELECT` a nejsou obsaženy v žádné agregační funkci, se musí vyskytnout v `GROUP BY`. V tomto případě jde o atribut `rating`. Pokud bychom toto pravidlo nedodrželi, nahlásí nám SQL Server syntaktickou chybu.

2. Pro každé ID zákazníka vypište počet jeho příjmení. Je ve výsledku něco překvapivého?

```
SELECT customer_id, COUNT(last_name) AS pocet
FROM customer
GROUP BY customer_id
```

   Mělo by nás napadnout, že zadání je trochu nesmyslně formulováno. Jestliže `customer_id` jednoznačně identifikuje každého zákazníka, pak `COUNT` bude pro všechny zákazníky vracet hodnotu 1, protože každý zákazník má zkrátka jedno příjmení. Dotaz by měl význam ve chvíli, kdy by příjmení nebyl povinný atribut. Pak by `COUNT` vracel hodnotu 1 nebo 0 v závislosti na tom, zda by zákazník měl nebo neměl vyplněné příjmení.

3. Vypište ID zákazníků setřízená podle součtu všech jejich plateb. Zákazníky, kteří neprovedli žádnou platbu neuvažujte.

   Při řešení takového dotazu bychom měli začít nejprve výpisem součtů plateb pro jednotlivé zákazníky:

```
SELECT customer_id, SUM(amount)
FROM payment
GROUP BY customer_id
```

   Je zřejmé, že tento dotaz bude vypisovat pouze ID zákazníků, kteří provedli nějakou platbu, jelikož pracujeme jen s tabulkou `payment`. Později si ukážeme, jak případně vypsat např. i zákazníky, kteří žádnou platbu nemají.

   Teprve až sestavíme výše uvedený dotaz, přesuneme výraz `SUM(amount)` do klauzule `ORDER BY`.

```
SELECT customer_id
FROM payment
GROUP BY customer_id
ORDER BY SUM(amount)
```

Každopádně, pokud třídíme výsledek dotazu dle nějakého výrazu, nebylo by na škodu mít tento výraz obsažený i v `SELECT`, aby koncový uživatel viděl, podle čeho je výsledek vlastně setřízený. V praxi by takovýto přístup byl spíše žádoucí. Z dvojitého použití `SUM(amount)` nemusíme mít obavy – rozumný databázový systém bude součet počítat jen jednou.

4. Pro každé jméno a příjmení herce vypište počet herců s takovým jménem a příjmením. Výsledek setřiďte dle počtu sestupně.

```
SELECT first_name, last_name, COUNT(*) AS pocet
FROM actor
GROUP BY first_name, last_name
ORDER BY pocet DESC
```

Na této úloze vidíme, že seskupovat můžeme i dle více atributů. Všimněme si, že v klauzuli `ORDER BY` můžeme používat místo výrazů rovnou aliasy sloupců (v tomto případě `pocet`).

5. Vypište součty všech plateb za jednotlivé roky a měsíce. Výsledek uspořádejte podle roků a měsíců.

```
SELECT YEAR(payment_date) AS rok, MONTH(payment_date) AS mesic, SUM(amount
    ) AS soucet
FROM payment
GROUP BY YEAR(payment_date), MONTH(payment_date)
ORDER BY rok, mesic
```

Tento příklad opět ukazuje, že klauzule pracují nejen s atributy, ale obecně s výrazy. V tomto případě tedy vytváříme skupiny podle roku a měsíce a pro tyto skupiny pak zjišťujeme součet plateb.

6. Vypište ID skladů s více než 2 300 kopiemi filmů.

```
SELECT store_id, COUNT(*)
FROM inventory
GROUP BY store_id
HAVING COUNT(*) > 2300
```

Kromě klauzule `GROUP BY` se s agregačními funkcemi váže ještě jedna klauzule – `HAVING`. Její význam je podobný jako `WHERE`, nicméně `HAVING` slouží pro specifikaci podmínek na agregované hodnoty (tj. hodnoty vypočtené agregační funkcí). Můžeme se snadno přesvědčit, že kdybychom výraz `COUNT(*) > 2300` umístili do `WHERE`, dotaz by nebylo možné spustit. Při zpracování `WHERE` totiž databázový systém ještě neví, pro jaké skupiny (specifikované v `GROUP BY`) má agregace počítat.

`HAVING` je poslední z klauzulí příkazu SELECT, kterou jsme si dosud neukázali. Pro shrnutí se tedy příkaz SELECT může skládat z klauzulí: `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING` a `ORDER BY`. Mělo by být jasné, že z logického pohledu začíná zpracování celého dotazu vždy spojením tabulek za `FROM`. Nad spojenými tabulkami proběhne selekce podle `WHERE`. Poté u agregačních dotazů dojde k vytvoření skupin dle atributů v `GROUP BY`. Následně může proběhnout selekce skupin v `HAVING` a až úplně nakonec dojde k projekci podle klauzule `SELECT` a případnému setřízení podle klauzule `ORDER BY`.

7. Vypište ID jazyků, pro které je nejkratší film delší než 46 minut.

   Při sestavování takového dotazu je vhodné začít jednodušší variantou, kdy si pro každé `language_id` vypíšeme nejkratší délku filmu. Tzn. začneme dotazem:

   ```
   SELECT language_id, MIN(length)
   FROM film
   GROUP BY language_id
   ```

   Teprve až vidíme, že takový dotaz funguje, přesuneme `MIN(length)` do klauzule `HAVING`, tzn. výsledkem je dotaz:

   ```
   SELECT language_id
   FROM film
   GROUP BY language_id
   HAVING MIN(length) > 46
   ```

8. Vypište roky a měsíce plateb, kdy byl součet plateb větší než 20 000.

   ```
   SELECT
     YEAR(payment_date) AS rok, MONTH(payment_date) AS mesic,
     SUM(amount) AS soucet
   FROM payment
   GROUP BY YEAR(payment_date), MONTH(payment_date)
   HAVING SUM(amount) > 20000
   ```

   Tento příklad navazuje na úlohu 5, kterou pouze rozšíříme o použití klauzule `HAVING`.

9. Vypište klasifikace filmů (atribut `rating`), jejichž délka je menší než 50 minut a celková délka v dané klasifikaci je větší než 250 minut. Výsledek setřiďte sestupně podle abecedy.

   ```
   SELECT rating
   FROM film
   WHERE length < 50
   GROUP BY rating
   HAVING SUM(length) > 250
   ORDER BY rating DESC
   ```

   Na tomto příkladě vidíme použití všech klauzulí příkazu SELECT, tj. `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING` a `ORDER BY` v jednom jediném dotaze. Zamyslete se ještě jednou, jaký je rozdíl mezi klauzulemi `WHERE` a `HAVING`.

10. Vypište pro jednotlivá ID jazyků počet filmů. Vynechejte jazyky, které nemají žádný film.

    ```
    SELECT language_id, COUNT(*) AS pocet
    FROM film
    GROUP BY language_id
    ```

    Jde o poměrně snadnou úlohu, kdy stačí tabulku `film` seskupit dle atributu `language_id` a tento atribut vypsat společně s počtem záznamů. Řešení tohoto příkladu budeme v dalších dvou úlohách mírně modifikovat.

11. Vypište názvy jazyků a k nim počty filmů. Vynechejte jazyky, které nemají žádný film.

```
SELECT
  language.language_id, language.name, COUNT(*) AS pocet
FROM
  language
  JOIN film ON language.language_id = film.language_id
GROUP BY language.language_id, language.name
```

Zde se poprvé dostáváme ke kombinaci agregačních funkcí a spojování tabulek, které známe již z předchozího cvičení. Řešení této úlohy pouze rozšiřuje řešení předchozí úlohy o připojení tabulky `language`, abychom mohli vypsat název jazyka. Stále však platí, že co je v klauzuli `SELECT` a není obsaženo v agregační funkci, musí být v `GROUP BY`. To znamená, že, abychom vypsali název jazyka, nestačí pouze přidat `language.name` do klauzule `SELECT`. Tento atribut musíme přidat také do `GROUP BY`.

12. Vypište názvy všech jazyků a k nim počty filmů. Ve výsledku budou zahrnuty i ty jazyky, které nemají žádný film.

```
SELECT language.language_id, language.name, COUNT(film.film_id) AS
   pocet_filmu
FROM
  language
  LEFT JOIN film ON language.language_id = film.language_id
GROUP BY language.language_id, language.name
```

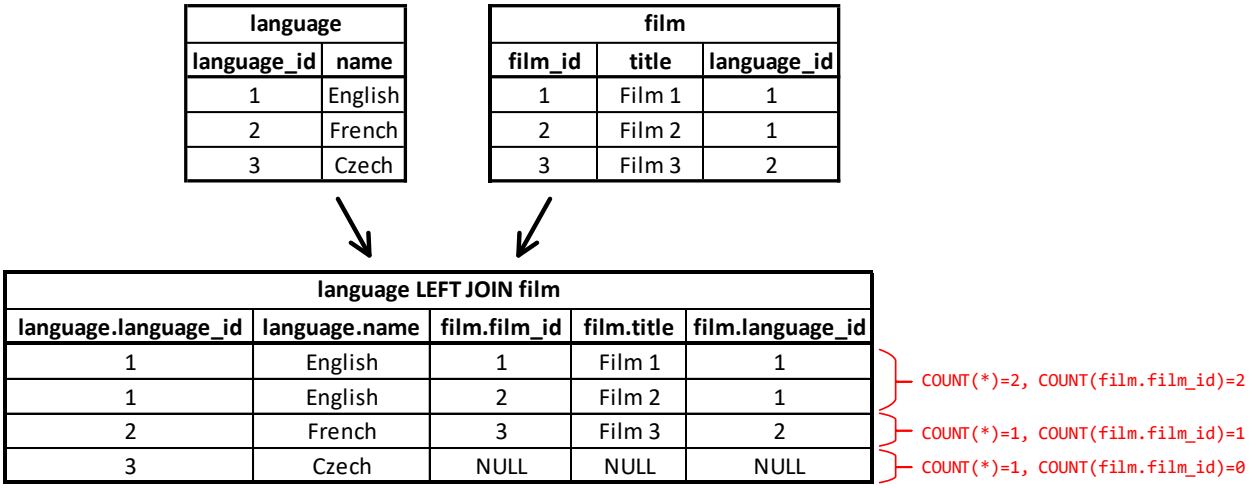
Předchozí dvě úlohy byly mírně zjednodušené tím, že jsme mohli ignorovat jazyky, pro které neexistuje žádný film. Nevypsali jsme tedy ty jazyky, kde by počet měl být nulový. V praxi nás ale právě tyto nulové počty mohou často zajímat. Proto se agregační funkce obvykle kombinují s vnějším spojením, aby se do výsledku dostaly všechny záznamy z určité tabulky, v tomto případě z tabulky `language`.

Všimněme, že jako argument funkce `COUNT` už nepoužíváme symbol * představující výskyt řádku, ale atribut `film.film_id`. Pokud bychom použili `COUNT(*)`, vrátila by tato funkce hodnotu 1 i pro jazyky bez filmů, protože i jazyk bez filmu představuje řádek, který `COUNT(*)` započítá. Využíváme zde vlastnosti vnějšího spojení, kdy pokud k nějakému jazyku operace `LEFT JOIN` nenajde žádný odpovídající film, doplní všechny atributy filmu (tj. třeba i atribut `film.film_id`) hodnotami `NULL`. Ty pak agregační funkce `COUNT` jednoduše přeskočí. Pro lepší představu se můžeme podívat na Obrázek 4, kde je patrný rozdíl mezi `COUNT(*)` a `COUNT(film.film_id)` na posledním řádku spojené tabulky.

13. Vypište pro jednotlivé zákazníky (jejich ID, jméno a příjmení) počty jejich výpůjček.

```
SELECT
  customer.customer_id, first_name, last_name,
  COUNT(rental.rental_id) AS pocet
FROM
  customer
  LEFT JOIN rental ON customer.customer_id = rental.customer_id
GROUP BY customer.customer_id, first_name, last_name
```

V návaznosti na předchozí příklad jde o obdobný problém. Je tedy nutné použít vnější spojení a správně zvolit argument funkce `COUNT`. V zadání pouze není zmíněno, že chceme vypsat i ty zákazníky, kteří nemají žádné výpůjčky, a ani v dalších úlohách nebudeme tento požadavek explicitně zmiňovat. Úlohu bychom měli vždy vyřešit tak, že výsledek bude kompletní, tj. bude obsahovat vše, co neodporuje zadání.



Obrázek 4: Rozdíl `COUNT(*)` a `COUNT(film.film_id)`

14. Vypište pro jednotlivé zákazníky (jejich ID, jméno a příjmení) počty různých filmů, které si vypůjčili.

```
SELECT customer.customer_id, first_name, last_name, COUNT(DISTINCT
    inventory.film_id) AS pocet_filmu
FROM
  customer
  LEFT JOIN rental ON customer.customer_id = rental.customer_id
  LEFT JOIN inventory ON rental.inventory_id = inventory.inventory_id
GROUP BY customer.customer_id, first_name, last_name
```

Smyslem této úlohy je uvědomit si rozdíl mezi počtem výpůjček a počtem výpůjček různých filmů. Již z prvního cvičení bychom měli vědět, že „počet různých“ řešíme agregační funkcí `COUNT`, kde před název atributu napíšeme `DISTINCT`. Oproti předchozí úloze je nutné navíc připojit ještě tabulku `inventory`, kde najdeme atribut `film_id`. Všimněme si, že samotnou tabulku `film` už nepotřebujeme, nicméně pokud bychom ji přesto připojili, výsledek by se nijak nezměnil.

15. Vypište jména a příjmení herců, kteří hrají ve více než 20-ti filmech.

```
SELECT actor.first_name, actor.last_name
FROM
  actor
  JOIN film_actor ON actor.actor_id = film_actor.actor_id
GROUP BY actor.actor_id, actor.first_name, actor.last_name
HAVING COUNT(film_actor.film_id) > 20
```

Smyslem tohoto úkolu je mimo jiné připomenout si klauzuli `HAVING`. Podobně jako v úloze 7, pokud řešíme úkol, kde předpokládáme použití `HAVING`, měli bychom si vždy nejprve napsat dotaz, kde bude agregační funkce použita za `SELECT`. Teprve až takový dotaz funguje, přesuneme agregační funkci do `HAVING` a specifikujeme podmínku.

Dále, co by nás mohlo překvapit, je použití `JOIN` místo `LEFT JOIN`, tedy vnitřního spojení místo vnějšího. Nejprve nutno říci, že použitím `LEFT JOIN` bychom se v tomto případě určitě nedopustili žádné chyby. Jde jen o to, uvědomit si, že ze zadání vyplývá, že chceme

vypsat jen herce, kteří hrají v nějakém filmu (přesněji tedy hrají alespoň ve 20-ti filmech). Z toho vyplývá, že herci, kteří nehrají v žádném filmu, nás vůbec nezajímají a tím pádem „výhoda“ vnějšího spojení je nám tady k ničemu.

Úloha ale demonstruje ještě jeden praktický problém. Zadání po nás chce, abychom vypsali jména a příjmení herců, ale v řešení v klauzuli `GROUP BY` vidíme navíc atribut `actor.actor_id`, proč? Co když se dva herci budou jmenovat stejně? V databázi takové máme – zkuste z `GROUP BY` atribut `actor.actor_id` odebrat a porovnejte výsledek.

16. Pro každého zákazníka vypište, kolik celkem utratil za výpůjčky filmů a jaká byla jeho nejmenší, největší a průměrná částka platby.

```
SELECT
  customer.customer_id, first_name, last_name,
  SUM(payment.amount) AS celkem, MIN(payment.amount) AS nejmensi,
  MAX(payment.amount) AS nejvetsi, AVG(payment.amount) AS prumer
FROM
  customer
  LEFT JOIN rental ON customer.customer_id = rental.customer_id
  LEFT JOIN payment ON rental.rental_id = payment.rental_id
GROUP BY customer.customer_id, first_name, last_name
```

Abychom stále nepoužívali pouze agregační funkci `COUNT`, zde vidíme praktické použití všech dalších agregačních funkcí v jednom dotazu.

17. Vypište pro každou kategorii průměrnou délku filmu.

```
SELECT category.category_id, category.name,
  AVG(CAST(film.length AS FLOAT)) AS prumer
FROM
  category
  LEFT JOIN film_category ON category.category_id = film_category.
     category_id
  LEFT JOIN film ON film_category.film_id = film.film_id
GROUP BY category.category_id, category.name
```

Na úloze není nic až tak zajímavého kromě přetypování argumentu agregační funkce `AVG`, což jsme prováděli už v úloze 24 na straně 15. Atribut `film.length` je totiž celé číslo a výpočty nad celými čísly opět vrací celé číslo, podobně jako např. v jazyce C++. Můžeme se snadno přesvědčit, že odebráním přetypování na `FLOAT` (desetinné číslo s plovoucí čárkou) nebude výsledek dotazu úplně správný.

18. Pro každý film vypište, jaký byl celkový příjem z výpůjček. Vypište jen filmy, kde byl celkový příjem větší než 100.

```
SELECT film.film_id, film.title, SUM(payment.amount) AS celkem
FROM
  film
  LEFT JOIN inventory ON film.film_id = inventory.film_id
  LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
  LEFT JOIN payment ON rental.rental_id = payment.rental_id
GROUP BY film.film_id, film.title
HAVING SUM(payment.amount) > 100
```

Na této úloze si pro jistotu ještě jednou vyzkoušíme klauzuli `HAVING`, tentokrát v kombinaci se 4-mi spojenými tabulkami. Podobně jako u příkladu 15 bychom místo `LEFT JOIN`

v tomto případě mohli použít i JOIN, protože ze zadání vyplývá, že pracujeme jen s filmy, kterým odpovídá minimálně jedna platba (jinak by celkový příjem nemohl být > 100).

19. Pro každého herce vypište, v kolika různých kategoriích filmů hraje.

```
SELECT
  actor.actor_id, actor.first_name, actor.last_name,
  COUNT(DISTINCT film_category.category_id) AS pocet_kategorii
FROM
  actor
  LEFT JOIN film_actor ON actor.actor_id = film_actor.actor_id
  LEFT JOIN film_category ON film_actor.film_id = film_category.film_id
GROUP BY actor.actor_id, actor.first_name, actor.last_name
```

Na řešení této spíše jednodušší úlohy by nás mohlo překvapit, že tabulka film není uvedena v klauzuli FROM. Tabulka film_category je přímo spojena s tabulkou film_actor na základě atributu film_id, přestože ani v jedné z těchto dvou tabulek není film_id sám o sobě primárním klíčem. Připojit tabulku film je totiž při troše zamyšlení zcela zbytečné, jelikož kromě atributu film_id z ní nic nepotřebujeme. Na druhou stranu, pokud bychom tabulku připojili, nebyla by to chyba. Databázový systém by pravděpodobně pouze provedl několik zbytečných kroků navíc. Zajímavostí je, že moderní databázové systémy (červen 2019) neumí přítomnost takovéto „zbytečné“ tabulky v dotazu detekovat a v rámci optimalizace tabulku z klauzule FROM případně odstranit.

20. Vypište adresy zákazníků (atribut address.address) včetně názvu města a státu, kde ve filmech, které si zákazníci půjčili, hrálo dohromady alespoň 40 různých herců.

```
SELECT address.address, city.city, country.country
FROM
  customer
  JOIN address ON customer.address_id = address.address_id
  JOIN city ON address.city_id = city.city_id
  JOIN country ON city.country_id = country.country_id
  LEFT JOIN rental ON customer.customer_id = rental.customer_id
  LEFT JOIN inventory ON rental.inventory_id = inventory.inventory_id
  LEFT JOIN film_actor ON inventory.film_id = film_actor.film_id
GROUP BY address.address, city.city, country.country
HAVING COUNT(DISTINCT film_actor.actor_id) >= 40
```

Smyslem úlohy je ukázat, že klauzule GROUP BY může obsahovat i atributy z různých tabulek. Opět bychom měli začít tak, že si pro každého zákazníka (jeho adresu) vypíšeme počet různých herců hrajících ve filmech, které si zákazník půjčil. Potom na základě tohoto počtu vytvoříme podmínku v HAVING.

Dále nás může překvapit, proč jsou některé tabulky připojeny vnitřním a některé vnějším spojením. Pro spojení tabulek customer, address, city a country můžeme bez problémů použít JOIN, jelikož na základě povinností atributů (viz Datový slovník na str. 6) víme, že každý zákazník má adresu, každá adresa má město a každé město patří do nějakého státu. Počínaje tabulkou rental však tato povinnost již neplatí – zákazník nemusí mít výpůjčku a tím pádem ani položku v inventáři nebo film. Na druhou stranu, ani v tomto případě by nevadilo, kdybychom všude použili LEFT JOIN. Neměli bychom ale nabýt dojmu, že JOIN je vlastně zbytečný. Dokázali byste vymyslet dotaz, kde se použitím LEFT JOIN na místo JOIN dopustíme chyby?

21. Pro všechny filmy (ID a název) spadající do kategorie „Horror“ uveďte, v kolika různých městech bydlí zákazníci, kteří si daný film někdy půjčili.

```
SELECT
  film.film_id, film.title, COUNT(DISTINCT address.city_id) AS pocet_mest
FROM
  film
  JOIN film_category ON film.film_id = film_category.film_id
  JOIN category ON film_category.category_id = category.category_id
  LEFT JOIN inventory ON film.film_id = inventory.film_id
  LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
  LEFT JOIN customer ON rental.customer_id = customer.customer_id
  LEFT JOIN address ON customer.address_id = address.address_id
WHERE category.name = 'Horror'
GROUP BY film.film_id, film.title
```

Tato úloha je v principu velmi podobná té předchozí. Všimněme si akorát, že v tomto případě nepožadujeme úplně všechny filmy, ale jen filmy spadající do určité kategorie. Proto zde má smysl podmínka v klauzuli `WHERE`. Ze stejného důvodu také tabulky `film`, `film_category` a `category` spojujeme vnitřním spojením.

22. Pro všechny zákazníky z Polska vypište, do kolika různých kategorií spadají filmy, které si tito zákazníci vypůjčili.

```
SELECT customer.customer_id, customer.first_name, customer.last_name,
    COUNT(DISTINCT film_category.category_id) AS pocet_kategorii
FROM
  country
  JOIN city ON country.country_id = city.country_id
  JOIN address ON city.city_id = address.city_id
  JOIN customer ON address.address_id = customer.address_id
  LEFT JOIN rental ON customer.customer_id = rental.customer_id
  LEFT JOIN inventory ON rental.inventory_id = inventory.inventory_id
  LEFT JOIN film ON inventory.film_id = film.film_id
  LEFT JOIN film_category ON film.film_id = film_category.film_id
WHERE country.country = 'Poland'
GROUP BY customer.customer_id, customer.first_name, customer.last_name
```

Pro procvičení zde máme ještě jednu podobnou úlohu. Úloha je komplikovaná pouze nutností správně propojit poměrně velké množství tabulek.

23. Vypište názvy všech jazyků k nim počty filmů v daném jazyce, které jsou delší než 350 minut.

```
SELECT language.name, COUNT(film.film_id) AS pocet
FROM
  language
  LEFT JOIN film ON language.language_id = film.language_id
    AND film.length > 350
GROUP BY language.name
```

Podobně jako na minulém cvičení se dostáváme do situace, kdy je nutné rozšířit podmínku vnějšího spojení. Jako první obvykle studenty napadne umístit podmínku `film.length > 350` do klauzule `WHERE`. Můžeme se ale snadno přesvědčit, že výsledek pak nebude obsahovat jazyky s nulovým počtem filmů. Výsledkem operace `LEFT JOIN` totiž sice

jsou všechny jazyky, i když k nim neexistuje film, ale teprve potom by se zpracovala klauzule `WHERE`, která by nám tyto jazyky stejně nakonec vyfiltrovala. Přesunutím podmínky z `WHERE` do `LEFT JOIN` (za klíčové slovo `ON`, které je samozřejmě nedílnou součástí této konstrukce) se bude podmínka zpracovávat v rámci vnějšího spojení a výsledkem bude přesně to, co se po nás požaduje v zadání. Pokud máte pocit, že je na vás tato problematika příliš složitá, doporučujeme vrátit se k příkladu 24 na předchozím cvičení.

24. Vypište, kolik jednotliví zákazníci utratili za výpůjčky, které začaly v měsíci červnu.

```
SELECT
  customer.customer_id, first_name, last_name,
  COALESCE(SUM(payment.amount), 0) AS celkem
FROM
  customer
  LEFT JOIN rental ON customer.customer_id = rental.customer_id
    AND MONTH(rental.rental_date) = 6
  LEFT JOIN payment ON rental.rental_id = payment.rental_id
GROUP BY customer.customer_id, first_name, last_name
```

Také tato úloha vyžaduje zápis podmínky na měsíc červen do vnějšího spojení. Opět se přesvědčte, že přesunutím podmínky `MONTH(rental.rental_date) = 6` do `WHERE` bychom nedostali správný výsledek, protože někteří zákazníci by ve výpisu chyběli. Oproti předchozí úloze si všimněme, že podmínka se týká jiné tabulky, než ze které počítáme agregaci.

Pro připomenutí zde máme použití funkce `COALESCE`, která vrací první z uvedených atributů, který není `NULL`. Pokud agregační funkce `SUM` nezapočítá žádnou hodnotu, není jejím výsledkem 0, ale hodnota `NULL`, která může být pro běžného uživatele matoucí.

25. Vypište seznam kategorií setřízený podle počtu filmů, jejichž jazyk začíná písmenem „E“.

```
SELECT
  category.name
FROM
  category
  LEFT JOIN film_category ON category.category_id = film_category.
     category_id
  LEFT JOIN film ON film_category.film_id = film.film_id
  LEFT JOIN language ON film.language_id = language.language_id AND
     language.name LIKE 'E%'
GROUP BY category.name
ORDER BY COUNT(language.language_id)
```

Pro ukázku zde máme příklad, kde je agregace použita v klauzuli `ORDER BY`. Mělo by být samozřejmé, že při sestavování takového dotazu nejprve agregaci umístíme do `SELECT` a teprve až jsme si jisti, že dotaz vrací to, co má, přesuneme ji do `ORDER BY`.

Zvláštností na tomto příkladu je fakt, že argumentem agregační funkce `COUNT` je atribut `language.language_id`, přestože se po nás chce spočítat filmy (splňující určitou podmínku). Podobným trikem jako v předchozích dvou úlohách zde připojíme k filmům jen jazyky, jejichž název začíná na „E“. Pokud tedy jazyk filmu tuto podmínku nesplní, budou pro daný film všechny atributy tabulky `language` nabývat hodnoty `NULL`. Dále víme, že funkce `COUNT`, počítá jakýkoli (i opakovaný) výskyt každé hodnoty, která není

NULL. Z toho vyplývá, že počítáním výskytů hodnoty language.language_id počítáme právě filmy, jejichž jazyk splňuje danou podmínku.

26. Vypište názvy filmů s délkou menší než 50 minut, které si zákazníci s příjmením BELL půjčili přesně 1x.

```
SELECT film.film_id, film.title, customer.last_name, COUNT(customer.
    customer_id)
FROM
  film
  LEFT JOIN inventory ON film.film_id = inventory.film_id
  LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
  LEFT JOIN customer ON rental.customer_id = customer.customer_id AND
      customer.last_name = 'BELL'
WHERE film.length < 50
GROUP BY film.film_id, film.title, customer.last_name
HAVING COUNT(customer.customer_id) = 1
```

U této úlohy si ukážeme dvě správná řešení. První řešení výše využívá stejného principu jako řešení předchozích několika úloh. Přesouváme tedy podmínku na příjmení zákazníka z WHERE do LEFT JOIN, abychom nepřišli o filmy, které si zákazník „BELL“ nikdy nepůjčil. To se ale samozřejmě netýká podmínky na délku filmu, jelikož ve výsledku filmy s délkou ≥ 50 vůbec nechceme. Když se ale trochu zamyslíme, vadilo by, kdybychom přišli o filmy, které si zákazník „BELL“ nikdy nepůjčil? Nevadilo, protože chceme vypsat ty filmy, které si „BELL“ půjčil přesně jednou. Z toho plyne, že v této úloze vůbec nemusíme používat vnější spojení a podmínka na příjmení zákazníka může bez problémů zůstat v WHERE. Jak již bylo řečeno na předchozím cvičení, spojovací podmínky za ON bychom neměli zbytečně „zneužívat“ k jinému účelu, než ke kterému jsou určeny. Kromě toho, že tím můžeme zmást databázový systém, který pak nemusí vybrat optimální strategii pro vyhodnocení dotazu, bude dotaz sám o sobě nepřehledně formulovaný, z čehož mohou v praxi plynout zbytečné chyby.

```
SELECT film.film_id, film.title, customer.last_name, COUNT(customer.
    customer_id)
FROM
  film
  JOIN inventory ON film.film_id = inventory.film_id
  JOIN rental ON inventory.inventory_id = rental.inventory_id
  JOIN customer ON rental.customer_id = customer.customer_id
WHERE film.length < 50 AND customer.last_name = 'BELL'
GROUP BY film.film_id, film.title, customer.last_name
HAVING COUNT(customer.customer_id) = 1
```

4 Množinové operace a kvantifikátory

Až do teď jsme mohli každou úlohu vyřešit bez používání tzv. poddotazů. To znamená, že se klauzule `SELECT` v dotazu objevila vždy přesně jednou. Na tomto cvičení se zaměříme na používání konstrukcí `IN`, `EXISTS`, `ANY` a `ALL`, které používání poddotazů naopak vyžadují. Přestože některé z následujících úloh lze řešit i pomocí agregačních funkcí, pokusme se procvičit si především výše zmíněné konstrukce. Pokud nebude v zadání uvedeno jinak, je možné všechny následující úlohy vyřešit bez použití agregačních funkcí a shlukování. V praxi (nebo na testu) je pak ale jen na Vás, jaký způsob pro vyřešení úlohy použijete.

1. Vypište ID a názvy filmů, ve kterých hrál herec s ID = 1. Dotaz vyřešte bez použití `JOIN`.

```
SELECT film_id, title
FROM film
WHERE film_id IN (SELECT film_id FROM film_actor WHERE actor_id = 1)
```

Zřejmě nejúspornější zápis řešení této úlohy spočívá v použití konstrukce `IN`. S touto konstrukcí jsme se již setkali na prvním cvičení, kde jsme však do závorky za `IN` uvedli statický výčet hodnot (viz úloha 5 na straně 10). Zde však vidíme, že tyto hodnoty nemusí být „natvrdo“ vyjmenovány, ale pro jejich získání můžeme použít poddotaz. Poddotaz, tedy obsah závorky za `IN`, v tomto případě vrací ID filmů, kde hraje herec s ID = 1. Pojmenujme si množinu takových ID např. jako M. Vnější dotaz pak vrací všechna ID a názvy filmů, kde ID spadá do M.

Úlohu však můžeme elegantně řešit i pomocí konstrukce `EXISTS`:

```
SELECT film_id, title
FROM film
WHERE EXISTS (SELECT * FROM film_actor WHERE film.film_id = film_actor.
    film_id AND actor_id = 1)
```

Ptáme se, zda pro určitý film v tabulce `film` existuje záznam v tabulce `film_actor`, který odpovídá danému filmu v tabulce `film` a herci s ID = 1.

Z uvedených dvou řešení vidíme, že stejnou úlohu lze řešit pomocí více konstrukcí. Věnujme se chvíli rozdílům v syntaxi `IN` a `EXISTS`:

- `atribut IN (poddotaz)`
- `EXISTS (poddotaz)`

Nejprve bychom si měli uvědomit, že obě konstrukce vrací logickou hodnotu „pravda“ nebo „nepravda“ a v této úloze se vyhodnocují zvlášť pro každý film. Zatímco součástí konstrukce `IN` je vždy atribut před klíčovým slovem `IN`, konstrukce `EXISTS` funguje sama, bez atributu, pouze s poddotazem.

Dá se říct, že vše, co lze řešit pomocí `IN`, lze řešit i pomocí `EXISTS`. Výhodou `IN` je pouze to, že poddotaz bývá obvykle samostatně spustitelný, což umožňuje celý dotaz sestavit po částech – tzn. nejprve odladíme poddotaz a pak napíšeme vnější dotaz. Oproti tomu, poddotaz za `EXISTS` musí být vždy nějakou podmínkou propojený s vnějším dotazem (viz část `film.film_id = film_actor.film_id`) a samostatně spustitelný není. Pokud by poddotaz za `EXISTS` byl samostatně spustitelný (nezávisle na filmu), vrátil by `EXISTS` pro všechny filmy buď „pravda“, nebo pro všechny filmy „nepravda“. Výsledkem by

tedy byly buď úplně všechny filmy, nebo prázdná tabulka a nic mezi tím. To znamená, že použijeme-li `EXISTS` a poddotaz za `EXISTS` lze samostatně spustit, máme někde v zápisu určitě logickou chybu.

Zajisté nás může napadnout ptát se, zda bude z hlediska zpracování dotazu efektivnější varianta s `IN` nebo `EXISTS`. I když jsme v předchozím odstavci tvrdili, že obě konstrukce se vyhodnocují zvlášť pro každý film, je to pravda pouze z logického pohledu na věc. Databázový systém se pokusí najít efektivnější strategii, která může záviset na aktuálním obsahu tabulek a nastavených indexech. Rozumný databázový systém pravděpodobně nakonec vyhodnotí obě varianty naprosto stejným algoritmem. Přesněji řečeno, tzv. plán vykonání dotazu bude pro obě varianty stejný, ale o tom více až v dalším semestru v předmětu DAIS (Databázové a informační systémy).

Představme si ale ještě jedno řešení, které by Vás již mělo napadnout i bez znalosti `IN` nebo `EXISTS`. Jde totiž o typickou úlohu na spojení dvou tabulek. I toto řešení nakonec nejspíš bude vyhodnoceno stejným algoritmem jako varianta s `IN` a `EXISTS`:

```
SELECT film.film_id, title
FROM film JOIN film_actor ON film.film_id = film_actor.film_id
WHERE actor_id = 1
```

2. Vypište ID filmů, ve kterých hrál herec s ID = 1.

   Tato úloha je zjednodušením předchozí úlohy a lze ji samozřejmě vyřešit tak, že z klauzule `SELECT` vnějšího dotazu odebereme atribut `title`:

   ```
   SELECT film_id
   FROM film
   WHERE film_id IN (SELECT film_id FROM film_actor WHERE actor_id = 1)
   ```

   Smyslem úlohy je však uvědomit si, že výsledek uvedeného řešení je naprosto stejný jako výsledek samotného poddotazu, který tedy sám o sobě představuje elegantnější řešení.

   ```
   SELECT film_id
   FROM film_actor
   WHERE actor_id = 1
   ```

   Dokázali byste přesně zdůvodnit, proč obě řešení vrací stejný výsledek, a vymyslet příklad, kdy by podobné zjednodušení ke stejnému výsledku nevedlo?

3. Vypište ID a názvy filmů, ve kterých hrál herec s ID = 1 zároveň s hercem s ID = 10.

   První pokusy o řešení této úlohy obvykle končí následujícím zápisem, který ale samozřejmě není správně:

   ```
   SELECT film_id, title
   FROM film
   WHERE film_id IN (SELECT film_id FROM film_actor WHERE actor_id = 1 AND
      actor_id = 10)
   ```

   V podstatě tímto požadujeme vybrat záznamy z tabulky `film_actor`, kde `actor_id` je 1 a zároveň 10, což je z principu jednoduše nesmysl (tzv. kontradikce). Musíme si uvědomit, že řešíme problém průniku dvou množin. Řešení takového problému obvykle vede k dvojitému použití konstrukce `IN`:

```
SELECT film_id, title
FROM film
WHERE
  film_id IN (SELECT film_id FROM film_actor WHERE actor_id = 1) AND
  film_id IN (SELECT film_id FROM film_actor WHERE actor_id = 10)
```

První z poddotazů za `IN` vrátí množinu ID filmů pro herce s ID = 1. Druhý z poddotazů vrátí množinu ID filmů pro herce s ID = 2. Pojmenujme si tyto množiny jako M_1 a M_2. Vnější dotaz nakonec vrátí ID a názvy filmů, kde ID spadá do obou z množin M_1 a M_2.

Také tuto úlohu můžeme řešit i pomocí konstrukce `EXISTS`:

```
SELECT film_id, title
FROM film
WHERE
  EXISTS (SELECT * FROM film_actor WHERE film.film_id = film_actor.film_id
      AND actor_id = 1) AND
  EXISTS (SELECT * FROM film_actor WHERE film.film_id = film_actor.film_id
      AND actor_id = 10)
```

Opět nesmíme zapomenout, že poddotazy za `EXISTS` musí být propojeny podmínkou s vnějším dotazem. I zde vycházejme z představy, že se část dotazu za `WHERE` provede zvlášť pro každý film. Řešení se tedy dá interpretovat tak, že hledáme filmy, pro které existuje záznam v tabulce `film_actor` pro daný film a herce s ID = 1, resp. ID = 2.

4. Vypište ID a názvy filmů, ve kterých hrál herec s ID = 1 nebo herec s ID = 10.

   Také u této úlohy si ukážeme několik různých řešení. Oproti předchozí úloze se nám pouze změnila množinová operace – místo průniku řešíme sjednocení. Není nic jednoduššího než využít řešení předchozí úlohy a změnit logickou spojku `AND` na `OR`:

```
SELECT film_id, title
FROM film
WHERE
  film_id IN (SELECT film_id FROM film_actor WHERE actor_id = 1) OR
  film_id IN (SELECT film_id FROM film_actor WHERE actor_id = 10)
```

   Takové řešení je naprosto správné a stejný postup by fungoval i pro variantu s `EXISTS`. Nicméně zápis lze ještě zjednodušit:

```
SELECT film.film_id, title
FROM film
WHERE film_id IN (
  SELECT film_id
  FROM film_actor
  WHERE actor_id = 1 OR actor_id = 10
)
```

   Oproti množinovému průniku lze sjednocení obvykle řešit jednoduše použitím logické operace `OR`. Poddotaz tedy vybere ID filmů, kde hrál jeden nebo druhý herec, a vnější dotaz tento výsledek rozšíří o výpis názvů.

   Abychom nezapomněli, úlohu můžeme snadno vyřešit i prostým spojením obou tabulek:

```
SELECT DISTINCT film.film_id, title
FROM
  film
```

```
  JOIN film_actor ON film.film_id = film_actor.film_id
WHERE film_actor.actor_id = 1 OR film_actor.actor_id = 10
```

Jelikož je k tabulce `film` připojena vazební tabulka `film_actor`, bude se určitý film ve výsledku opakovat pro každý odpovídající záznam ve vazební tabulce.

5. Vypište ID filmů, ve kterých nehrál herec s ID = 1.

   Představme si nejprve nesprávné řešení této úlohy, které obvykle studenty napadne jako první:

```
SELECT film_id
FROM film_actor
WHERE actor_id != 1
```

   Proč je takové řešení špatně? Uvedený dotaz vrací ID filmů z vazební tabulky `film_actor`, kde ID herce není 1. Představme si film s ID = x, ve kterém hrají herci s ID = 1 a ID = 2. Obsah tabulky `film_actor` tedy vypadá takto:

film_id	actor_id
x	1
x	2

   Je zřejmé, že výsledek dotazu bude obsahovat ID = x, protože druhý řádek vyhoví podmínce v klauzuli `WHERE`. To je ale špatně, protože víme, že ve filmu x hraje herec s ID = 1 a takový film by tedy dle zadání ve výsledu být neměl.

   Nyní tedy správné řešení:

```
SELECT film_id
FROM film
WHERE film_id NOT IN (
  SELECT film_id
  FROM film_actor
  WHERE actor_id = 1
)
```

   Vnitřní dotaz vybírá všechna ID filmů, kde hraje herec s ID = 1. Takovou množinu si můžeme označit jako M. Vnější dotaz pak vybere všechna ID filmů, které nespadají do M, tj. ID filmů, kde nehraje herec s ID = 1. V principu tedy řešíme problém množinového rozdílu nebo z opačného problému řešíme doplněk do množiny všech herců.

   Pro úplnost si představme ještě řešení pomocí `EXISTS`:

```
SELECT film_id
FROM film
WHERE NOT EXISTS (
  SELECT film_id
  FROM film_actor
  WHERE film.film_id = film_actor.film_id AND actor_id = 1
)
```

   Opět nesmíme zapomenout propojit vnitřní dotaz s vnějším dotazem pomocí podmínky `film.film_id = film_actor.film_id`.

6. Vypište ID a názvy filmů, ve kterých hrál herec s ID = 1 nebo herec s ID = 10, ale ne oba dohromady.

```
SELECT film_id, title
FROM film
WHERE
  film_id IN (
    SELECT film_id FROM film_actor
    WHERE actor_id = 1 OR actor_id = 10
  )
  AND NOT
  (
    film_id IN (
      SELECT film_id FROM film_actor WHERE actor_id = 1
    )
    AND
    film_id IN (
      SELECT film_id FROM film_actor WHERE actor_id = 10
    )
  )
```

Po vyřešení předchozích úloh by toto pro nás neměl být problém, jde pouze o to správně zkombinovat podmínky z úloh 3 a 4. Důležité je neztratit se v zápisu a správně uzavřít odpovídající si závorky. Varinatu s `EXISTS` si zkuste napsat sami.

7. Vypište ID a názvy filmů, ve kterých hrál herec PENELOPE GUINESS zároveň s hercem CHRISTIAN GABLE.

```
SELECT film_id, title
FROM film
WHERE
  film_id IN (
    SELECT film_id
    FROM actor JOIN film_actor ON
      actor.actor_id = film_actor.actor_id
    WHERE
      actor.first_name = 'PENELOPE' AND
      actor.last_name = 'GUINESS'
  )
  AND film_id IN (
    SELECT film_id
    FROM actor JOIN film_actor
      ON actor.actor_id = film_actor.actor_id
    WHERE
      actor.first_name = 'CHRISTIAN' AND
      actor.last_name = 'GABLE'
  )
```

Zřejmě pro nás nebude překvapením, že uvedení herci v zadání mají v databázi ID 1 a 10, a že výsledek tedy musí odpovídat výsledku úkolu 3. Abychom mohli pracovat se jmény herců a nejen s jejich ID, je nutné v obou poddotazech za `IN` připojit tabulku `actor`.

8. Vypište ID a názvy filmů, ve kterých nehraje herec PENELOPE GUINESS.

```
SELECT film_id, title
FROM film
WHERE
```

```
  film_id NOT IN (
    SELECT film_id
    FROM actor JOIN film_actor ON
      actor.actor_id = film_actor.actor_id
    WHERE actor.first_name = 'PENELOPE' AND actor.last_name = 'GUINESS'
  )
```

Pro procvičení zde opět pouze rozšiřujeme řešení úlohy 5 tak, že místo daného ID uvádíme přímo jméno herce.

9. Vypište jména zákazníků, kteří si půjčili všechny filmy ENEMY ODDS, POLLOCK DELIVERANCE a FALCON VOLUME.

```
SELECT customer.customer_id, customer.first_name, customer.last_name
FROM customer
WHERE
  customer_id IN
  (
    SELECT customer_id
    FROM
      rental
      JOIN inventory ON
        rental.inventory_id = inventory.inventory_id
      JOIN film ON inventory.film_id = film.film_id
    WHERE film.title = 'ENEMY ODDS'
  ) AND customer_id IN
  (
    SELECT customer_id
    FROM
      rental
      JOIN inventory ON
        rental.inventory_id = inventory.inventory_id
      JOIN film ON inventory.film_id = film.film_id
    WHERE film.title = 'POLLOCK DELIVERANCE'
  ) AND customer_id IN
  (
    SELECT customer_id
    FROM
      rental
      JOIN inventory ON
        rental.inventory_id = inventory.inventory_id
      JOIN film ON inventory.film_id = film.film_id
    WHERE film.title = 'FALCON VOLUME'
  )
```

I když se může zdát řešení komplikované, jde pouze o obdobu úlohy 7. Vnitřní tři poddotazy vrací ID zákazníků, kteří si vypůjčili filmy uvedené v zadání. Vnější dotaz pak vybírá zákazníky, jejichž ID spadá do výsledku všech tří poddotazů zároveň. Řešíme tedy průnik tří množin.

10. Vypište jména a příjmení zákazníků, kteří si půjčili film GRIT CLOCKWORK v květnu i v červnu (libovolného roku).

```
SELECT first_name, last_name
FROM customer
WHERE
```

```
  customer_id IN (
    SELECT customer_id
    FROM
      rental
      JOIN inventory ON rental.inventory_id = inventory.inventory_id
      JOIN film ON inventory.film_id = film.film_id
    WHERE
      film.title = 'GRIT␣CLOCKWORK' AND
      MONTH(rental.rental_date) = 5
  ) AND customer_id IN (
    SELECT customer_id
    FROM
      rental
      JOIN inventory ON rental.inventory_id = inventory.inventory_id
      JOIN film ON inventory.film_id = film.film_id
    WHERE
      film.title = 'GRIT␣CLOCKWORK' AND
      MONTH(rental.rental_date) = 6
  );
```

Tato úloha je zde opět pro procvičení množinového průniku, kde jsou různé množiny specifikovány na základě měsíce. Poddotazy za `IN` vrací ID zákazníků, kteří si půjčili daný film v květnu, resp. v červnu.

11. Vypište jména a příjmení zákazníků, kteří se příjmením jmenují stejně jako nějaký herec.

```
SELECT first_name, last_name
FROM customer
WHERE last_name IN (SELECT last_name FROM actor)
```

Až dosud jsme v poddotaze za `IN` vraceli vždy seznam ID. Na této úloze však vidíme, že není problém vracet například také množinu textových řetězců.

Dotaz lze také velmi snadno přepsat pomocí konstrukce `EXISTS`:

```
SELECT first_name, last_name
FROM customer
WHERE EXISTS (
  SELECT *
  FROM actor
  WHERE actor.last_name = customer.last_name
)
```

12. Vypište jména filmů, které jsou stejně dlouhé, jako nějaké jiné filmy.

```
SELECT title
FROM film f1
WHERE EXISTS (
  SELECT *
  FROM film f2
  WHERE f1.length = f2.length AND f1.film_id != f2.film_id
)
```

Tato úloha je oproti předchozí o něco komplikovanější tím, že pracujeme dvakrát se stejnou tabulku `film`. Proto musíme tabulce ve vnějším i vnitřním dotaze přidělit alias – jednou `f1` a podruhé `f2`. Hledáme tedy film `f1`, pro který existuje jiný film `f2`, který

má stejnou délku. Nesmíme zapomenout na podmínku `f1.film_id != f2.film_id`, která zaručuje, že skutečně najdeme „jiný“ film. Kdyby podmínka v dotazu chyběla, byl by výsledkem výpis všech filmů, protože pro každý film f najdeme ten samý film f, který má samozřejmě stejnou délku.

Dotaz lze také řešit pomocí konstrukce `IN`:

```
SELECT title
FROM film f1
WHERE length IN (
  SELECT length
  FROM film f2
  WHERE f1.film_id != f2.film_id
)
```

Zda je pro Vás pochopitelnější varianta s `EXISTS` nebo `IN` může být subjektivní záležitost. V tomto případě však ztrácíme výhodu `IN`, kterou bývá to, že poddotaz můžeme samostatně spustit.

13. Vypište názvy filmů, které jsou kratší než nějaký film, ve kterém hraje BURT POSEY.

```
SELECT title
FROM film
WHERE length < ANY (
  SELECT film.length
  FROM
    actor
    JOIN film_actor ON actor.actor_id = film_actor.actor_id
    JOIN film ON film_actor.film_id = film.film_id
  WHERE actor.first_name = 'BURT' AND actor.last_name = 'POSEY'
)
```

Zadání této úlohy nás přímo vybízí k použití konstrukce – kvantifikátoru `ANY` nebo `SOME`. Obě konstrukce mají v SQL naprosto ekvivalentní význam. Mělo by být zřejmé, že poddotaz vrací délky všech filmů, kde hraje BURT POSEY. Pomocí `ANY` nebo `SOME` pak ve vnějším dotaze vybíráme filmy, jejichž délka je menší než některá z délek vrácených poddotazem. Při troše zamyšlení vrací celý dotaz filmy kratší než je nejdelší film herce BURT POSEY.

Podobně jako `IN`, je i konstrukce `ANY` snadno nahraditelná pomocí `EXISTS`:

```
SELECT title
FROM film f1
WHERE EXISTS
(
  SELECT *
  FROM
    actor
    JOIN film_actor ON actor.actor_id = film_actor.actor_id
    JOIN film f2 ON film_actor.film_id = f2.film_id
  WHERE actor.first_name = 'BURT' AND actor.last_name = 'POSEY' AND f1.
     length < f2.length
)
```

Hledáme tedy všechny filmy `f1`, pro které existuje alespoň jeden film `f2`, kde hraje BURT POSEY, který je delší než `f1`.

14. Vypište jména herců, kteří hráli v nějakém filmu kratším než 50 minut.

```
SELECT actor.first_name, actor.last_name
FROM actor
WHERE 50 > ANY (
  SELECT length
  FROM film JOIN film_actor ON film.film_id = film_actor.film_id
  WHERE film_actor.actor_id = actor.actor_id
)
```

Také zde můžeme snadno využít kvantifikátor `ANY`. Zatímco v úloze 13 však byl poddotaz za `ANY` samostatně spustitelný, je v tomto případě množina délek filmů závislá na každém konkrétním herci. Nemělo by pro nás být překvapením, že před `ANY` nemusí být jen atribut, ale také např. konstanta.

Zřejmě tušíme, že dotaz lze formulovat také pomocí `EXISTS`:

```
SELECT actor.first_name, actor.last_name
FROM actor
WHERE EXISTS (
  SELECT *
  FROM film JOIN film_actor ON film.film_id = film_actor.film_id
  WHERE film_actor.actor_id = actor.actor_id AND film.length < 50
)
```

Nakonec ale nezapomeňme na tu pravděpodobně nejsnazší variantu, která využívá pouze spojení tabulek `actor`, `film_actor` a `film`.

```
SELECT DISTINCT first_name, last_name
FROM
  actor
  JOIN film_actor ON actor.actor_id = film_actor.actor_id
  JOIN film ON film_actor.film_id = film.film_id
WHERE film.length < 50
```

15. Nalezněte filmy, které byly půjčeny alespoň dvakrát.

Při řešení této úlohy se může zdát, že se neobejdeme bez agregačních funkcí. Na zadání se však můžeme dívat i tak, že hledáme výpůjčku v_1, pro kterou existuje jiná výpůjčka v_2 stejného filmu. Pokud takovou v_1 najdeme, byl její film vypůjčen alespoň dvakrát. Takováto formulace nás dovede k následujícímu dotazu:

```
SELECT DISTINCT f1.title
FROM
  rental r1
  JOIN inventory i1 ON r1.inventory_id = i1.inventory_id
  JOIN film f1 ON i1.film_id = f1.film_id
WHERE
  EXISTS (
    SELECT *
      FROM
        rental r2
        JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
      WHERE i2.film_id = i1.film_id AND r1.rental_id != r2.rental_id
  )
```

Rozeberme si podmínku `WHERE` v poddotazu `EXISTS`. Podmínka je složená ze dvou částí, kde `r1.film_id = r2.film_id` testuje, zda pro výpůjčku `r1` existuje výpůjčka

`r2` stejného filmu, a `r1.rental_id != r2.rental_id` ověřuje, zda jde o jinou výpůjčku, tedy, že `r1` není `r2`. Zvlášť na druhou část lze snadno zapomenout.

Vzhledem k tomu, že stejný film může být ve výpůjčkách vícekrát (v tomto případě dle zadání dokonce musí), uvedeme za `SELECT` vnějšího dotazu také modifikátor `DISTINCT`, aby výpis neobsahoval duplicity.

Jak bylo napsáno v úvodu, smyslem tohoto cvičení je především procvičit si konstrukce `IN`, `EXISTS`, `ANY` a `ALL`. Tuto úlohu však samozřejmě můžeme velmi elegantně řešit také pomocí agregační funkce `COUNT` následujícím způsobem:

```
SELECT film.title
FROM
  film
  JOIN inventory ON film.film_id = inventory.film_id
  JOIN rental ON inventory.inventory_id = rental.inventory_id
GROUP BY film.film_id, film.title
HAVING COUNT(rental.customer_id) > 1
```

Výhodou řešení přes agregační funkci je, že můžeme velmi snadno upravit dotaz tak, aby vracel filmy vypůjčené třeba 10 krát, což by u varianty s `EXISTS` nešlo nebo šlo jen velmi komplikovaně.

16. Nalezněte filmy, které si půjčili alespoň 2 různí zákazníci.

```
SELECT DISTINCT f1.title
FROM
  rental r1
  JOIN inventory i1 ON r1.inventory_id = i1.inventory_id
  JOIN film f1 ON i1.film_id = f1.film_id
WHERE
  AND EXISTS (
    SELECT *
    FROM
      rental r2
      JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
    WHERE i2.film_id = i1.film_id AND r1.customer_id != r2.customer_id
  )
```

Jde o velmi podobnou úlohu jako v předchozím případě. Nyní však nehledáme jinou výpůjčku ale výpůjčku jiným zákazníkem. To vede pouze k malé úpravě podmínky `WHERE` v poddotazu. Také tuto úlohu můžeme snadno vyřešit použitím agregační funkce, tentokrát `COUNT(DISTINCT rental.customer_id)` pro počítání unikátních zákazníků:

```
SELECT film.title
FROM
  film
  JOIN inventory ON film.film_id = inventory.film_id
  JOIN rental ON inventory.inventory_id = rental.inventory_id
GROUP BY film.film_id, film.title
HAVING COUNT(DISTINCT rental.customer_id) > 1
```

17. Vypište zákazníky, kteří měli v určitou chvíli ve výpůjčce zároveň více různých filmů.

Při řešení této úlohy si musíme nejprve uvědomit, kdy se překrývají dva intervaly. Zkusme se na problém podívat z opačného pohledu a nejprve zjistit, kdy se dva intervaly naopak

nepřekrývají. Uvažujme dvě výpůjčky v_1 a v_2. Intervaly těchto dvou výpůjček se nebudou překrývat, jestliže v_1 skončí dříve než začne v_2 nebo když v_1 začne později než skončí v_2. Tuto podmínku můžeme zapsat jako:

v_1`.return_date < `v_2`.rental_date OR `v_1`.rental_date > `v_2`.return_date`

Pro testování opačného případu, tedy zda se dva intervaly překrývají, můžeme podmínku negovat operátorem `NOT`, tzn.:

`NOT(`v_1`.return_date < `v_2`.rental_date OR `v_1`.rental_date > `v_2`.return_date)`

Elegantnější však bude využít De Morganův zákon pro negaci disjunkce, který říká, že $\neg(a \vee b) = \neg a \wedge \neg b$. Finální podoba podmínky tedy bude vypadat takto:

v_1`.return_date >= `v_2`.rental_date AND `v_1`.rental_date <= `v_2`.return_date`

Pro sestavení SQL dotazu využijeme stejný princip jako v předchozích úlohách. Budeme se ptát, zda pro určitou výpůjčku `r1` existuje výpůjčka `r2` jiného filmu pro stejného zákazníka s překrývajícím se intervalem:

```
SELECT DISTINCT customer.customer_id, customer.first_name, customer.
    last_name
FROM
  rental r1
  JOIN inventory i1 ON r1.inventory_id = i1.inventory_id
  JOIN customer ON r1.customer_id = customer.customer_id
WHERE EXISTS (
  SELECT *
  FROM
    rental r2
    JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
  WHERE
    r1.customer_id = r2.customer_id AND
    i1.film_id != i2.film_id AND
    r1.return_date >= r2.rental_date AND
    r1.rental_date <= r2.return_date
  )
```

18. Vypište jména a příjmení zákazníků, kteří si půjčili film GRIT CLOCKWORK v květnu i červnu téhož roku.

```
SELECT first_name, last_name
FROM
  customer
  JOIN rental r1 ON customer.customer_id = r1.customer_id
  JOIN inventory i1 ON r1.inventory_id = i1.inventory_id
  JOIN film f1 ON i1.film_id = f1.film_id
WHERE
  f1.title = 'GRIT␣CLOCKWORK'
  AND MONTH(r1.rental_date) = 5
  AND EXISTS (
    SELECT *
    FROM
      rental r2
      JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
      JOIN film f2 ON i2.film_id = f2.film_id
    WHERE
      r1.customer_id = r2.customer_id
```

```
        AND f2.title = 'GRIT␣CLOCKWORK'
        AND  MONTH(r2.rental_date) = 6
        AND YEAR(r1.rental_date) = YEAR(r2.rental_date)
    )
```

Na první pohled je úloha velmi podobná úloze 10. Malý rozdíl v tom, že vyžadujeme měsíce ze stejného roku, vede k mírně odlišnému přístupu k řešení úlohy. Vnějším dotazem hledáme výpůjčky daného filmu v měsíci květnu. Vnitřním dotazem za `EXISTS` hledáme opět výpůjčky daného filmu ale v měsíci červnu. To, že už vnější dotaz zahrnuje tabulku výpůjček (těch květnových), nám umožní, abychom ve vnitřním dotaze ověřili stejný rok. Pokud bychom řádek s podmínkou na rok odstranili, byl by výsledek ekvivalentní k výsledku úlohy 10.

Někoho by mohlo napadnout zrušit ve vnitřním dotaze porovnání na název filmu a rovnou testovat shodnost `f1.film_id = f2.film_id`, což je pak to samé jako testovat `f1.film_id = i2.film_id`. Takovou úpravou bychom se dostali k dotazu:

```
SELECT first_name, last_name
FROM
  customer
  JOIN rental r1 ON customer.customer_id = r1.customer_id
  JOIN inventory i1 ON r1.inventory_id = i1.inventory_id
  JOIN film f1 ON i1.film_id = f1.film_id
WHERE
  f1.title = 'GRIT␣CLOCKWORK'
  AND MONTH(r1.rental_date) = 5
  AND EXISTS (
    SELECT *
    FROM
      rental r2
      JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
    WHERE
      f1.film_id = i2.film_id
      AND r1.customer_id = r2.customer_id
      AND  MONTH(r2.rental_date) = 6
      AND YEAR(r1.rental_date) = YEAR(r2.rental_date)
  )
```

Bylo by však výsledkem to samé? Téměř ano. Rozdíl by mohl nastat v případě, že bychom v databázi měli dva filmy se stejným názvem. Druhé z uvedených řešení by zajišťovalo, že si zákazník skutečně půjčil ten samý film. První řešení by připustilo možnost, že si zákazník půjčil dva filmy, které se náhodou jmenují stejně. V praxi by pravděpodobně druhé z uvedených řešení bylo „správnější“.

19. Vypište názvy filmů, které jsou kratší než všechny filmy, ve kterých hraje BURT POSEY.

```
SELECT title
FROM film
WHERE length < ALL (
  SELECT film.length
  FROM
    actor
    JOIN film_actor ON actor.actor_id = film_actor.actor_id
    JOIN film ON film_actor.film_id = film.film_id
  WHERE actor.first_name = 'BURT' AND actor.last_name = 'POSEY'
)
```

Úloha je v principu velmi podobná úloze 13, pouze použijeme jiný kvantifikátor – `ALL`. Vracíme tedy filmy kratší než je nejkratší film, kde hraje BURT POSEY. Tuto úlohu můžeme vyřešit také pomocí `EXISTS`, přesněji tedy `NOT EXISTS`:

```
SELECT title
FROM film f1
WHERE NOT EXISTS
(
  SELECT *
  FROM
    actor
    JOIN film_actor ON actor.actor_id = film_actor.actor_id
    JOIN film f2 ON film_actor.film_id = f2.film_id
  WHERE actor.first_name = 'BURT' AND actor.last_name = 'POSEY' AND f2.
      length <= f1.length
)
```

Hledáme takové filmy `f1`, pro které neexistuje film `f2`, kde by hrál herec BURT POSEY a délka `f2` $\leq$ délka `f1`. Jinak řečno, pokud bychom takový `f2` našli, nebude `f1` kratší než jakýkoli film herce BURT POSEY.

Na této úloze jsme si ukázali poslední z konstrukcí `IN`, `EXISTS`, `ANY` a `ALL`, které si na tomto cvičení máme vyzkoušet. Je však vidět, že všechny konstrukce `IN`, `ANY` i `ALL` lze relativně snadno nahradit pomocí `EXISTS`. (Ve skutečnosti je problém s ekvivalencí těchto konstrukcí ještě trochu složitější). Pokud se Vám tedy varianty řešení s `EXISTS` líbí nejvíce, máte výhodu, neboť ostatní konstrukce nepotřebujete.

20. Vypište jména herců, kteří hráli jen ve filmech kratších než 180.

```
SELECT actor.first_name, actor.last_name
FROM actor
WHERE
  180 > ALL (
    SELECT length
    FROM film JOIN film_actor ON film.film_id = film_actor.film_id
    WHERE film_actor.actor_id = actor.actor_id
  )
  AND actor_id IN (SELECT actor_id FROM film_actor)
```

Opět navážeme na jednu z předchozích úloh. Tato úloha se oproti úloze 14 liší v použití kvantifikátoru `ALL` místo `ANY`. Můžeme si ale všimnout nově přidané podmínky na posledním řádku. Problém je, že konstrukce `ALL` vrátí hodnotu „pravda“ i v případě, že poddotaz vrátí prázdný výsledek. Podmínka tedy zajišťuje, že herec hraje alespoň v jednom filmu.

Také zde můžeme použít konstrukci `NOT EXISTS`:

```
SELECT actor.first_name, actor.last_name
FROM actor
WHERE
  NOT EXISTS (
    SELECT *
    FROM film JOIN film_actor ON film.film_id = film_actor.film_id
    WHERE film_actor.actor_id = actor.actor_id AND film.length >= 180
  )
  AND actor_id IN (SELECT actor_id FROM film_actor)
```

Nesmí tedy existovat film, ve kterém by herec hrál a délka tohoto filmu by byla větší nebo rovna 180. Opět je ale nutné otestovat, zda herec vůbec v nějakém filmu hraje.

21. Vypište zákazníky, kteří v žádném měsíci neměli více než 3 výpůjčky. Pro zjištění počtu výpůjček v jednotlivých měsících použijte agregační funkci a shlukování.

```
SELECT first_name, last_name
FROM customer
WHERE customer_id NOT IN
(
  SELECT customer_id
  FROM rental
  GROUP BY customer_id, MONTH(rental_date)
  HAVING COUNT(*) > 3
)
```

Základem řešení této úlohy je umět sestavit dotaz, který vrátí zákazníky, kteří v nějakém měsíci provedli více než 3 výpůjčky. Přesně toto jsou pak zákazníci, kteří ve výsledku nemají být, což opět vede k použití negované podmínky, v tomto případě `NOT IN`.

22. Vypište zákazníky, kteří si půjčovali filmy pouze v letních měsících (tj. červen až srpen včetně).

Zkusme si větu v zadání této úlohy přeformulovat: Vypište zákazníky, kteří si nikdy nepůjčili film v jiném než letním měsíci. Tato formulace pak přímo vede na následující řešení pomocí `NOT EXISTS`:

```
SELECT first_name, last_name
FROM customer
WHERE NOT EXISTS (
  SELECT *
  FROM rental
  WHERE
    customer.customer_id = rental.customer_id AND
    MONTH(rental.rental_date) NOT BETWEEN 6 AND 8
) AND customer_id IN (SELECT customer_id FROM rental)
```

Tzn., vypisujeme zákazníky, pro které neexistuje záznam o výpůjčce v jiných než letních měsících. Podmínkou na posledním řádku opět zajistíme, že zákazník si skutečně něco půjčil (viz např. úloha 20).

Podobně můžeme úlohu řešit i pomocí `NOT IN`:

```
SELECT first_name, last_name
FROM customer
WHERE customer_id NOT IN
(
  SELECT customer_id
  FROM rental
  WHERE MONTH(rental.rental_date) NOT BETWEEN 6 AND 8
) AND customer_id IN (SELECT customer_id FROM rental)
```

Poddotaz nejprve vybere zákazníky, kteří si někdy půjčili film v jiném než letním měsíci. Vnější dotaz pak vrátí všechny zákazníky, kteří nepatří do výsledku tohoto poddotazu.

Již na základě úloh 19, 20 nebo 21 můžeme dojít k obecně platnému pravidlu, že jakmile vidíme v zadání úlohy slovo „pouze“, „jen“, „vždy“ apod., měli bychom začít uvažovat

o řešení za použití `NOT EXISTS` nebo `NOT IN`. Musíme totiž obecně ověřit, zda neexistuje záznam, který by danou podmínku jakkoli porušil.

23. Vypište zákazníky, kteří vždy vrátili film do 8-mi dnů. Výpůjčky, které zákazník dosud nevrátil, ignorujte.

```
SELECT *
FROM customer
WHERE NOT EXISTS (
  SELECT *
  FROM rental
  WHERE
    rental.customer_id = customer.customer_id
    AND DATEDIFF(day, rental.rental_date, rental.return_date) > 8
) AND customer_id IN (SELECT customer_id FROM rental)
```

V principu řešíme stále stejný typ úlohy. Budeme hledat zákazníky, pro které neexistuje výpůjčka, která by byla delší než 8 dní. Podmínka na posledním řádku opět zajišťuje, že zákazník má alespoň jednu výpůjčku. Dovětek o ignorování nevrácených výpůjček, znamená jen to, že není potřeba ošetřovat situaci, kdy zákazník má nějakou výpůjčku otevřenou, a to třeba už i dlouhou dobu.

24. Vypište zákazníky, jejichž všechny výpůjčky byly delší než 1 den a půjčili si film, kde hraje DEBBIE AKROYD.

```
SELECT first_name, last_name
FROM customer
WHERE
  customer_id NOT IN (
    SELECT customer_id
    FROM rental
    WHERE DATEDIFF(DAY, rental_date, return_date) <= 1
  )
  AND customer_id IN (
    SELECT customer_id
    FROM
      rental
      JOIN inventory ON rental.inventory_id = inventory.inventory_id
      JOIN film ON inventory.film_id = film.film_id
      JOIN film_actor ON film.film_id = film_actor.film_id
      JOIN actor ON film_actor.actor_id = actor.actor_id
    WHERE
      actor.first_name = 'DEBBIE' AND actor.last_name = 'AKROYD'
  )
```

Zadání této úlohy specifikuje dvě podmínky, které musí platit současně. Nejprve musíme zjistit, zda jsou všechny výpůjčky daného zákazníka delší než 1 den. To provedeme pomocí `NOT IN` tak, že ověříme, zda zákazník naopak nemá výpůjčku kratší než 1 den nebo trvající přeně 1 den. Druhou podmínkou pak ověříme, zda zákazník provedl výpůjčku filmu, kde hraje DEBBIE AKROYD.

25. Vypište jména a příjmení zákazníků, kteří provedli přesně jednu výpůjčku.

V zadání bychom si opět měli všimnout slova „přesně“. Podobně jako např. v úloze 26 se velmi snadno obejdeme bez agregačních funkcí. Pokusíme se pro nějakou výpůjčku v_1

najít jinou výpůjčku v_2 stejného zákazníka. Oproti úloze 26, pokud takovou v_2 najdeme, je naopak v_1 výpůjčka, kterou ve výpisu nechceme. Dotaz lze tedy formulovat pomocí `NOT EXISTS` takto:

```
SELECT customer.first_name, customer.last_name
FROM
  rental r1
  JOIN customer ON r1.customer_id = customer.customer_id
WHERE NOT EXISTS (
  SELECT *
  FROM rental r2
  WHERE r1.customer_id = r2.customer_id AND
    r1.rental_id != r2.rental_id
)
```

Podobně jako úlohu 15 lze i tuto řešit pomocí agregační funkce a shlukování:

```
SELECT customer.first_name, customer.last_name
FROM
  rental
  JOIN customer ON rental.customer_id = customer.customer_id
GROUP BY customer.customer_id, customer.first_name, customer.last_name
HAVING COUNT(*) = 1
```

26. Vypište názvy filmů, kde hraje jediný herec.

    Tuto úlohu zde zařazujeme jen pro procvičení. Princip jejího řešení je naprosto stejný jako v předchozí úloze, takže jej už nebudeme podrobně komentovat. Uveďme si nejprve řešení s `NOT EXISTS`, které by v tuto chvíli mělo být cílem:

    ```
    SELECT film.film_id, film.title
    FROM
      film
      JOIN film_actor fa1 ON film.film_id = fa1.film_id
    WHERE NOT EXISTS (
      SELECT *
      FROM film_actor fa2
      WHERE fa1.film_id = fa2.film_id AND fa1.actor_id != fa2.actor_id
    )
    ```

    Pokud byste dotaz v praxi nebo na testu vyřešili pomocí agregační funkce, určitě by to bylo také v pořádku:

    ```
    SELECT film.film_id, film.title
    FROM
      film
      JOIN film_actor ON film.film_id = film_actor.film_id
    GROUP BY film.film_id, film.title
    HAVING COUNT(*) = 1
    ```

27. Vypište zákazníky, kteří si půjčovali vždy jeden a ten samý film.

    ```
    SELECT DISTINCT customer.first_name, customer.last_name
    FROM
      rental r1
      JOIN inventory i1 ON r1.inventory_id = i1.inventory_id
      JOIN customer ON r1.customer_id = customer.customer_id
    ```

```
WHERE NOT EXISTS (
  SELECT *
  FROM
    rental r2
    JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
  WHERE r1.customer_id = r2.customer_id AND i1.film_id != i2.film_id
)
```

Řešení této úlohy je velmi podobné řešení úlohy 25. Je potřeba provést dvě změny. Nejprve musíme vnější i vnitřní dotaz rozšířit o připojení tabulky `inventory` (jednou pojmenované jako `i1`, podruhé jako `i2`), ze které pak můžeme zjistit hodnotu `film_id`. V podmínce `WHERE` v poddotazu pak hledáme výpůjčku jiného filmu. Pokud takovou najdeme, provedl zákazník výpůjčky `r1` ještě další výpůjčku jiného filmu a tím pádem nás takový zákazník nezajímá. Vzhledem k tomu, že v této úloze zákazník může provést více výpůjček (vždy však pro ten samý film), měli bychom pomocí `DISTINCT` zamezit duplicitním výsledkům.

Také tuto úlohu můžeme jednoduše vyřešit i použitím agregační funkce.

```
SELECT customer.first_name, customer.last_name
FROM
  rental
  JOIN customer ON rental.customer_id = customer.customer_id
  JOIN inventory ON inventory.inventory_id = rental.inventory_id
GROUP BY customer.customer_id, customer.first_name, customer.last_name
HAVING COUNT(DISTINCT inventory.film_id) = 1
```

`COUNT(DISTINCT film_id)` bude pro jednotlivé zákazníky počítat unikátní hodnoty atributu `film_id`. Klauzulí `HAVING` zajistíme, aby byl počet unikátních výskytů roven 1.

28. Vypište názvy filmů, které si někdy půjčili zákazníci, kteří si nikdy nepůjčili jiný film.

```
SELECT DISTINCT film.title
FROM
  rental r1
  JOIN inventory i1 ON r1.inventory_id = i1.inventory_id
  JOIN film ON i1.film_id = film.film_id
  JOIN customer ON r1.customer_id = customer.customer_id
WHERE NOT EXISTS (
  SELECT *
  FROM
    rental r2
    JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
  WHERE r1.customer_id = r2.customer_id AND i1.film_id != i2.film_id
)
```

Přestože se zadání této úlohy může zdát velmi komplikované, jde pouze o minimální úpravu předchozí úlohy. V předchozí úloze bylo požadováno vypsat zákazníky, kteří si půjčovali vždy jen jediný film. V této úloze máme vypsat právě název onoho filmu. Proto pouze rozšíříme vnější dotaz o připojení tabulky `film`.

29. Vypište všechny zákazníky (jména a příjmení) a jazyky, pokud si zákazník půjčoval pouze filmy v daném jazyce.

Při troše zamyšlení je tato úloha pouze obdobou předchozích dvou. Hledáme výpůjčky, pro které neexistuje jiná výpůjčka stejného zákazníka ale filmu v jiném jazyce.

```
SELECT DISTINCT customer.first_name, customer.last_name, language.name
FROM
  customer
  JOIN rental r1 ON customer.customer_id = r1.customer_id
  JOIN inventory i1 ON r1.inventory_id = i1.inventory_id
  JOIN film f1 ON i1.film_id = f1.film_id
  JOIN language ON f1.language_id = language.language_id
WHERE NOT EXISTS (
  SELECT *
  FROM
    rental r2
    JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
    JOIN film f2 ON i2.film_id = f2.film_id
  WHERE r2.customer_id = r1.customer_id AND f2.language_id != f1.
      language_id
)
```

Také tuto úlohu můžeme řešit i pomocí agregačních funkcí:

```
SELECT customer.first_name, customer.last_name, MIN(language.name) AS name
FROM
customer
  LEFT JOIN rental ON customer.customer_id = rental.customer_id
  LEFT JOIN inventory ON rental.inventory_id = inventory.inventory_id
  LEFT JOIN film ON inventory.film_id = film.film_id
  LEFT JOIN language ON film.language_id = language.language_id
GROUP BY customer.customer_id, customer.first_name, customer.last_name
HAVING COUNT(DISTINCT language.language_id) = 1
```

V případě použití agregačních funkcí musíme vyřešit malý problém s tím, že v `SELECT` nemůžeme vypsat rovnou atribut `language.name`, protože tento atribut není (a nemůže být) uveden v `GROUP BY` (jinak by `COUNT(DISTINCT language_id)` vracelo vždy 1 a podmínka v `HAVING` by tedy nefungovala správně). Jelikož však víme, že vypisujeme zákazníky, kde všechny jazyky filmů jsou stejné, stačí umístit `language.name` např. do funkce `MIN` nebo `MAX` (jazyk je pro zákazníka jen jeden, takže minimum i maximum bude stejné).

30. Vypište názvy filmů, které si vždy půjčovali jen zákazníci, kteří si nikdy jiný film nepůjčili.

```
SELECT title
FROM film
WHERE
  film_id NOT IN
  (
    SELECT i1.film_id
    FROM
      rental r1
      JOIN inventory i1 ON r1.inventory_id = i1.inventory_id
    WHERE EXISTS (
      SELECT *
      FROM rental r2 JOIN inventory i2 ON r2.inventory_id = i2.
          inventory_id
      WHERE r1.customer_id = r2.customer_id AND i1.film_id != i2.film_id
    )
  )
```

```
AND film_id IN (
  SELECT film_id
  FROM
    inventory
    JOIN rental ON inventory.inventory_id = rental.inventory_id
)
```

Na této úloze je vidět, že záměna „někdy půjčili“ z úlohy 28 za „vždy půjčovali“ je poměrně zásadní. Začněme tím, že si zadání přeformulujeme tak, že nemáme vypisovat názvy filmů, které si někdy půjčil zákazník, který si někdy půjčil i jiný film. Poddotaz za `NOT IN` vybírá ID filmů, které si půjčil zákazník, který si někdy půjčil i jiný film. Tato ID představují přesně ty filmy, které ve výpisu nechceme. Výběr filmů tedy nakonec pomocí `NOT IN` otočíme. Podobně jako např. v úloze 20 pomocí dalšího `IN` pak zajistíme, že film byl alespoň jednou vypůjčen.

31. Vypište jména a příjmení zákazníků, kteří si vždy půjčovali pouze filmy, kde hrál herec CHRISTIAN GABLE.

```
SELECT first_name, last_name
FROM customer
WHERE customer_id NOT IN
(
  SELECT DISTINCT customer_id
  FROM
    rental
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
  WHERE film_id NOT IN (
    SELECT film_id
    FROM
      film_actor
      JOIN actor ON film_actor.actor_id = actor.actor_id
    WHERE first_name = 'CHRISTIAN' AND last_name = 'GABLE'
  )
) AND customer_id IN (SELECT customer_id FROM rental)
```

Slovo „pouze“ v zadání nám opět napovídá, že budeme zase potřebovat `NOT EXISTS` nebo `NOT IN`. Hledáme zákazníky, pro které neexistuje výpůjčka filmu, kde nehraje herec CHRISTIAN GABLE. Vidíme, že v řešení obsahuje dvě zanořené konstrukce `NOT IN` a jednu konstrukci `IN` na konci řádku, jejíž význam by měl být zřejmý (viz např. úloha 20). Abychom se v řešení lépe orientovali, pojmenujme si obsah závorek za druhým výskytem `NOT IN` jako Q_3, obsah závorek za prvním `NOT IN` jako Q_2 a celý dotaz jako Q_1. Q_3 vrací ID filmů, kde hraje CHRISTIAN GABLE, Q_2 pak vrací ID zákazníků z výpůjček filmů, kde CHRISTIAN GABLE nehraje, a to jsou právě ti zákazníci, kteří nás nezajímají, což nakonec řeší celý dotaz Q_1.

32. Vypište herce, kteří hráli vždy jen ve filmu, který půjčovna vlastní alespoň ve třech kopiích. Pro zjištění počtu kopií v inventáři použijte agregační funkci.

```
SELECT first_name, last_name
FROM actor
WHERE actor_id NOT IN
(
  SELECT actor_id
  FROM film_actor
```

```
    WHERE film_id NOT IN
    (
      SELECT film.film_id
      FROM
        film
        LEFT JOIN inventory ON film.film_id = inventory.film_id
      GROUP BY film.film_id
      HAVING COUNT(*) >= 3
    )
  ) AND actor_id IN (SELECT actor_id FROM film_actor)
```

Řešení této úlohy je založeno na stejném principu jako v předchozí úloze. Také zde nám pomůže mírná změna formulace zadání, samozřejmě při zachování stejného významu. Hledáme herce, kteří nikdy nehráli ve filmu, který v inventáři není obsažen alespoň 3x. Označme si opět Q_3 jako obsah závorek za druhým `NOT IN`, Q_2 jako obsah závorek za prvním `NOT IN` a Q_1 jako celý dotaz. Q_3 vrací filmy, které jsou v inventáři obsaženy alespoň třikrát, zde se použití agregační nelze vyhnout. Q_2 vrací herce, kteří v takových filmech nehrají a to jsou nakonec ti herci, kteří nás nezajímají, což řeší dotaz Q_1. Část dotazu na posledním řádku opět zajišťuje, že herec hraje alespoň v jednom filmu.

Ukažme si ještě další řešení této úlohy, kde druhé použití konstrukce `NOT IN` změníme na `IN`, ale otočíme podmínku v klauzuli `HAVING`. To znamená, že dotaz Q_3 vrací filmy, které jsou v inventáři obsaženy méně než 2x a Q_2 hledá herce, kteří v takových filmech hrají.

```
SELECT first_name, last_name
FROM actor
WHERE actor_id NOT IN
(
  SELECT actor_id
  FROM film_actor
  WHERE film_id IN
  (
    SELECT film.film_id
    FROM
      film
      LEFT JOIN inventory ON film.film_id = inventory.film_id
    GROUP BY film.film_id
    HAVING COUNT(*) < 3
  )
) AND actor_id IN (SELECT actor_id FROM film_actor)
```

Druhé uvedené řešení lze nakonec ještě o něco zjednodušit a to tak, že dotazy Q_2 a Q_3 spojíme:

```
SELECT first_name, last_name
FROM actor
WHERE actor_id NOT IN
(
  SELECT film_actor.actor_id
  FROM
    film
    JOIN film_actor ON film.film_id = film_actor.film_id
    LEFT JOIN inventory ON film.film_id = inventory.film_id
  GROUP BY film.film_id, film_actor.actor_id
```

```
    HAVING COUNT(*) < 3
) AND actor_id IN (SELECT actor_id FROM film_actor)
```

Dotaz za `NOT IN` tedy rovnou vrací ID herců, kteří hrají ve filmech, které jsou obsaženy v inventáři méně než 3x.

Všimněme si, že v zadání není přesně specifikováno, co konkrétně máme u herců vypsat. Mělo by být jasné, že v takovém případě požadujeme nějaký rozumný popisný atribut nebo atributy, jako jsou v tomto případě jméno a příjmení.

33. Nalezěte filmy, jejichž všechny kopie byly půjčeny alespoň 4x. Pro zjištění počtu kopií v inventáři použijte agregační funkci.

```
SELECT title
FROM film
WHERE film_id NOT IN
(
  SELECT inventory.film_id
  FROM
    inventory
    LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
  GROUP BY inventory.inventory_id, inventory.film_id
  HAVING COUNT(rental.rental_id) < 4
)
AND film_id IN (SELECT film_id FROM inventory)
```

Tato úloha je pouze obdobou předchozí úlohy a zařazujeme ji zde zejména z důvodu procvičení. Vnitřní dotaz vrací ID filmů z položek v inventáři, ke kterým se vztahují méně než 4 výpůjčky. Vnější dotaz pak vrací všechny ostatní filmy, tedy takové, kde všechny kopie filmu mají alespoň 4 výpůjčky. Posledním řádkem opět zajistíme, že dotaz nevrátí filmy, které nejsou obsaženy v inventáři.

34. Vypište herce, jejichž všechny filmy, kde hráli, jsou delší než filmy, ve kterých hrál herec CHRISTIAN GABLE.

```
SELECT first_name, last_name
FROM actor
WHERE actor_id NOT IN (
  SELECT film_actor.actor_id
  FROM
    film_actor
    JOIN film ON film_actor.film_id = film.film_id
  WHERE film.length < SOME (
    SELECT film.length
    FROM
      actor
      JOIN film_actor ON actor.actor_id = film_actor.actor_id
      JOIN film ON film_actor.film_id = film.film_id
    WHERE actor.first_name = 'CHRISTIAN' AND
      actor.last_name = 'GABLE'
  )
) AND actor_id IN (SELECT actor_id FROM film_actor)
```

Také v této úloze nejprve přeformulujeme zadání a budeme hledat herce, kteří nikdy nehráli ve filmu, který by byl kratší než některý z filmu, kde hraje CHRISTIAN GABLE.

Označme si obsah závorek za `SOME` jako Q_3, část dotazu za `NOT IN` jako Q_2 a celý dotaz jako Q_1. Mělo by být jasné, že Q_3 vrací délky filmů, kde hraje CHRISTIAN GABLE. Q_2 pak vrací herce hrající ve filmech, které jsou kratší než některá z délek vrácených pomocí Q_3. Herci vrácení pomocí Q_2 jsou nakonec přesně ti, co nás nezajímají, což opět řeší Q_1. Význam posledního řádku v celém dotazu by měl být po pochopení předchozích úloh zřejmý.

35. Vypište herce, jejichž filmy delší než 180 si půjčovali vždy zákazníci ze stejné země.

```
SELECT actor.actor_id, first_name, last_name
FROM actor
WHERE NOT EXISTS (
  SELECT film_actor.actor_id
  FROM
    film_actor
    JOIN film ON film_actor.film_id = film.film_id
    JOIN inventory i1 ON film.film_id = i1.film_id
    JOIN rental r1 ON i1.inventory_id = r1.inventory_id
    JOIN customer c1 ON r1.customer_id = c1.customer_id
    JOIN address a1 ON c1.address_id = a1.address_id
    JOIN city ct1 ON a1.city_id = ct1.city_id
  WHERE film_actor.actor_id = actor.actor_id AND film.length > 180 AND
     EXISTS (
      SELECT *
      FROM
        inventory i2
        JOIN rental r2 ON i2.inventory_id = r2.inventory_id
        JOIN customer c2 ON r2.customer_id = c2.customer_id
        JOIN address a2 ON c2.address_id = a2.address_id
        JOIN city ct2 ON a2.city_id = ct2.city_id
      WHERE i2.film_id = i1.film_id AND ct2.country_id != ct1.country_id
    )
)
```

V této úloze je nekomplikovanější zorientovat se v zadání. Jde o to, že pokud herec hraje ve filmu delším než 180, pak si tento film mohou půjčovat pouze zákazníci pocházející ze stejné země. Slovo „vždy“ nám opět říká, že budeme pracovat s negací. Uvedené řešení se tedy dá interpretovat tak, že hledáme herce, pro které neexistuje výpůjčka filmu delšího než 180, pro který existuje ještě jiná výpůjčka, kde zákazník pochází z jiné země. Ze zadání v tomto případě nevyplývá, že herec musí hrát ve filmu delším než 180 a proto řešení neobsahuje podmínku, která by toto zajišťovala.

5 Poddotazy

Poslední cvičení na dotazování v jazyce SQL je zaměřeno na používání poddotazů a to nejen v souvislosti s konstrukcemi, které jsme si ukázali na cvičení předchozím. Poddotazy nám nakonec umožní řešit poměrně komplikované úkoly.

1. Pro každý film vypište, kolik v něm hraje herců a v kolika se nachází kategoriích.

   Představme si nejprve nesprávné řešení, které studenty obvykle napadne jako první:

```
SELECT
  film.film_id, film.title, COUNT(actor_id) AS pocet_hercu,
  COUNT(category_id) AS pocet_kategorii
FROM
  film
  LEFT JOIN film_category ON film.film_id = film_category.film_id
  LEFT JOIN film_actor ON film.film_id = film_actor.film_id
GROUP BY film.film_id, film.title
```

   Po spuštění dotazu rychle zjistíme, že u každého filmu je počet herců stejný jako počet kategorií a to je minimálně velmi podezřelé. Proč je toto řešení špatně jsme si již ukazovali na úloze 12, str. 19. Dotaz totiž pro každý film vytvoří kartézský součin herců a kategorií a pak pomocí agregační funkcí `COUNT` počítá jakékoli (i opakované) výskyty ID. Abychom dostali správný výsledek, stačí před atributy ve funkcích `COUNT` přidat klíčové slovo `DISTINCT`, které zajistí, že spočítáme jen unikátní výskyty hodnot:

```
SELECT
  film.film_id, film.title, COUNT(DISTINCT actor_id) AS pocet_hercu,
  COUNT(DISTINCT category_id) AS pocet_kategorii
FROM
  film
  LEFT JOIN film_category ON film.film_id = film_category.film_id
  LEFT JOIN film_actor ON film.film_id = film_actor.film_id
GROUP BY film.film_id, film.title;
```

   Takovéto řešení je sice již správně, nicméně je tak trochu specifické pro případ, že používáme agregační funkci `COUNT`. Představme si jiné, univerzálnější řešení, na jehož principu bychom mohli počítat i jiné agregace než počet:

```
SELECT
  film.film_id, film.title,
  (
    SELECT COUNT(*)
    FROM film_actor
    WHERE film_actor.film_id = film.film_id
  ) AS pocet_hercu,
  (
    SELECT COUNT(*)
    FROM film_category
    WHERE film_category.film_id = film.film_id
  ) AS pocet_kategorii
FROM film
```

   Zde poprvé vidíme, že i klauzule `SELECT` může jako výrazy obsahovat poddotazy. Pravidlem je, že každý poddotaz musí být uzávorkovaný a musí vracet přesně jednu hodnotu,

tj. jeden řádek a jeden sloupec. První z poddotazů spočítá ke každému filmu počet herců a druhý počet kategorií. Na poddotazy se můžeme dívat jako na jakési funkce, jejichž vstupem (argumentem) je v tomto případě atribut `film_id`. Z logického pohledu se pak tyto funkce vyhodnocují zvlášť pro každý film. Nemusíme zde řešit žádné záludnosti v podobě nechtěných kartézských součinů.

Zkusme si všimnout také rozdílu řešení této úlohy oproti řešení úlohy 16 na straně 32, kde bylo možné skutečně jednoduše použít více agregačních funkcí za `SELECT`. Rozdíl je, že zde počítáme agregované hodnoty pro úplně jiné skupiny záznamů (jednou pro herce, podruhé pro kategorie), kdežto ve zmiňované úloze na straně 32 se všechny agregační funkce vyhodnocovaly nad stejnou skupinou záznamů (nad platbami stejného zákazníka).

K řešení můžeme přistoupit ale také jiným způsobem. Studentům obvykle nedělá problém sestavit dva samostatné dotazy, kdy jeden dotaz ke každému filmu vypíše počet herců a druhý počet kategorií. Problém pak ale je spojit oba výsledky dohromady. Představme si, že máme dvě fiktivní tabulky `pocty_hercu` a `pocty_kategorii`, které ke každému filmu uchovávají počty herců, resp. počty kategorií. Tabulka `pocty_hercu` tedy obsahuje atributy `film_id`, `title` a `pocet_hercu`, a tabulka `pocty_kategorii` atributy `film_id`, `title` a `pocet_kategorii`. Uměli bychom tyto tabulky spojit tak, že na jeden řádek dostaneme jak atribut `pocet_hercu`, tak atribut `pocet_kategorii`? To by pro nás jistě neměl být problém:

```
SELECT
  pocty_hercu.film_id, pocty_hercu.title, pocet_hercu, pocet_kategorii
FROM
  pocty_hercu
  JOIN pocty_kategorii ON pocty_hercu.film_id = pocty_kategorii.film_id
```

Přestože víme, že obě tabulky `pocty_hercu` i `pocty_kategorii` ve skutečnosti neexistují, je uvedené řešení jen malý krok od kompletního řešení. Stačí totiž před tyto „fiktivní“ tabulky napsat poddotaz, který vrací jejich obsah:

```
SELECT pocty_hercu.film_id, pocty_hercu.title, pocet_hercu,
   pocet_kategorii
FROM
  (
    SELECT film.film_id, film.title, COUNT(film_actor.film_id) AS
       pocet_hercu
    FROM film LEFT JOIN film_actor ON film.film_id = film_actor.film_id
    GROUP BY film.film_id, film.title
  ) pocty_hercu
  JOIN
  (
    SELECT film.film_id, COUNT(film_category.film_id) AS pocet_kategorii
    FROM film LEFT JOIN film_category ON film.film_id = film_category.
       film_id
    GROUP BY film.film_id, film.title
  ) pocty_kategorii ON pocty_hercu.film_id = pocty_kategorii.film_id
```

Platí pravidlo, že poddotaz musí být opět v závorce a každý sloupec, který poddotaz vrací, musí být pojmenovaný. To je logické, protože na nepojmenovaný sloupec bychom se ve vnějším dotazu nemohli nijak odkázat.

Jestliže je Vám představa „fiktivních“ tabulek blízká, určitě se Vám bude líbit poměrně „nová“ konstrukce SQL (součást standardu od roku 1999). Jde o tzv. výrazy CTE (Com-

mon Table Expressions). Princip těchto výrazů spočívá v uvedení speciální konstrukce `WITH` před klauzulí `SELECT`, pomocí které nadefinujeme libovolné množství „fiktivních“ tabulek, které pak můžeme použít v samotném dotazu. Řešení pak vypadá takto:

```
WITH
  pocty_hercu AS (
    SELECT film.film_id, film.title, COUNT(film_actor.film_id) AS
       pocet_hercu
    FROM film LEFT JOIN film_actor ON film.film_id = film_actor.film_id
    GROUP BY film.film_id, film.title
  ),
  pocty_kategorii AS (
    SELECT film.film_id, COUNT(film_category.film_id) AS pocet_kategorii
    FROM film LEFT JOIN film_category ON film.film_id = film_category.
       film_id
    GROUP BY film.film_id, film.title
  )
SELECT pocty_hercu.film_id, pocty_hercu.title, pocet_hercu,
   pocet_kategorii
FROM
  pocty_hercu
  JOIN pocty_kategorii ON pocty_hercu.film_id = pocty_kategorii.film_id
```

Pozor, jestliže budeme v následujícím textu při používání CTE výrazů mluvit o „fiktivních“ tabulkách, které připravuje klauzule `WITH`, rozhodně to neznamená, že SQL server nejprve takovou tabulku fyzicky připraví, někam ji uloží a pak vyřeší zbytek dotazu. Jde pouze o logickou konstrukci, která v mnoha případech vede k přehlednějšímu zápisu. Je velmi pravděpodobné, že např. druhé uvedené správné řešení této úlohy (poddotazy za `SELECT`) bude vyhodnoceno úplně stejným postupem jako řešení využívající `WITH`.

Přestože je tato konstrukce velmi přehledná a užitečná, existují systémy (případně starší verze systémů), které ji nepodporují. Proto bychom měli vždy umět přepsat `WITH` na předchozí řešení s poddotazy. Nakonec je samozřejmě opět jen na Vás, které řešení si v praxi nebo na testu zvolíte.

2. Pro každého zákazníka vypište počet výpůjček trvajících méně než 5 dní a počet výpůjček trvajících méně než 7 dní.

   Jde o obdobu předchozího příkladu, kdy k záznamům z určité tabulky počítáme různé agregace. Představme si nejprve variantu s poddotazy za `SELECT`, kde první poddotaz vrací pro určitého zákazníka počet výpůjček vrácených do 5-ti dní a druhý počet výpůjček vrácených do 7-mi dní:

```
SELECT
  first_name, last_name,
  (
    SELECT COUNT(*)
    FROM rental
    WHERE
      rental.customer_id = customer.customer_id
      AND DATEDIFF(day, rental_date, return_date) < 5
  ) AS kratsi_5,
  (
    SELECT COUNT(*)
    FROM rental
```

```
    WHERE
      rental.customer_id = customer.customer_id
      AND DATEDIFF(day, rental_date, return_date) < 7
  ) AS kratsi_7
FROM customer
```

Dále zkusme použít představu fiktivních tabulek `k5` a `k7`, které budou představovat počty výpůjček kratších než 5, resp. 7 dní:

```
SELECT k5.first_name, k5.last_name, kratsi_5, kratsi_7
FROM
(
  SELECT customer.customer_id, customer.first_name, customer.last_name,
     COUNT(rental.rental_id) AS kratsi_5
  FROM
    customer
    LEFT JOIN rental ON customer.customer_id = rental.customer_id
      AND DATEDIFF(day, rental_date, return_date) < 5
  GROUP BY customer.customer_id, customer.first_name, customer.last_name
) k5 JOIN (
  SELECT customer.customer_id, customer.first_name, customer.last_name,
     COUNT(rental.rental_id) AS kratsi_7
  FROM
    customer
    LEFT JOIN rental ON customer.customer_id = rental.customer_id
      AND DATEDIFF(day, rental_date, return_date) < 7
  GROUP BY customer.customer_id, customer.first_name, customer.last_name
) k7 ON k5.customer_id = k7.customer_id
```

Pozor, v poddotazech, které ke každému zákazníkovi počítají počty výpůjček splňujících určitou podmínku na dobu trvání nelze tuto podmínku umístit do `WHERE`! Přišli bychom totiž o zákazníky, kteří žádnou takovou výpůjčku nemají (viz např. úloha 23 na str. 34).

Nakonec ještě řešení pomocí CTE:

```
WITH
  k5 AS
  (
    SELECT customer.customer_id, customer.first_name, customer.last_name,
       COUNT(rental.rental_id) AS kratsi_5
    FROM
      customer
      LEFT JOIN rental ON customer.customer_id = rental.customer_id
        AND DATEDIFF(day, rental_date, return_date) < 5
    GROUP BY customer.customer_id, customer.first_name, customer.last_name
  ),
  k7 AS
  (
    SELECT customer.customer_id, customer.first_name, customer.last_name,
       COUNT(rental.rental_id) AS kratsi_7
    FROM
      customer
      LEFT JOIN rental ON customer.customer_id = rental.customer_id
        AND DATEDIFF(day, rental_date, return_date) < 7
    GROUP BY customer.customer_id, customer.first_name, customer.last_name
  )
SELECT k5.first_name, k5.last_name, kratsi_5, kratsi_7
FROM k5 JOIN k7 ON k5.customer_id = k7.customer_id;
```

3. Pro každý sklad (jeho ID) vypište počet kopií filmů (tj. položek v inventáři) pro filmy v anglickém jazyce a pro filmy ve francouzském jazyce.

   Pro procvičení stále pracujeme s obdobným typem úlohy. Nejprve tedy zkusme využít poddotazy pro spočítání kopií filmů v českém nebo francouzském jazyce pro každý sklad:

```
SELECT
  store.store_id,
  (
    SELECT COUNT(*)
    FROM
      inventory
      JOIN film ON inventory.film_id = film.film_id
      JOIN language ON film.language_id = language.language_id
    WHERE inventory.store_id = store.store_id AND language.name = 'English
      '
  ) AS english,
  (
    SELECT COUNT(*)
    FROM
      inventory
      JOIN film ON inventory.film_id = film.film_id
      JOIN language ON film.language_id = language.language_id
    WHERE inventory.store_id = store.store_id AND language.name = 'French'
  ) AS czech
FROM store
```

   Ukažme si také elegantní řešení pomocí CTE, které je však postaveno na trochu odlišném principu než v předchozích dvou úlohách:

```
WITH t AS (
  SELECT inventory.store_id, language.name
  FROM
    inventory
    JOIN film ON inventory.film_id = film.film_id
    JOIN language ON film.language_id = language.language_id
)
SELECT
  store_id,
  (
    SELECT COUNT(*)
    FROM t
    WHERE t.name = 'English' AND t.store_id = store.store_id
  ) AS english,
  (
    SELECT COUNT(*)
    FROM t
    WHERE t.name = 'French' AND t.store_id = store.store_id
  ) AS czech
FROM store
```

   Do „fiktivní“ tabulky `t` si připravíme dvojice hodnot `store_id` a `language.name`, které představují položky v inventáři na skladě `store_id` pro film v jazyce `language.name`. Pomocí poddotazů pak ke každému skladu spočítáme počet odpovídajících záznamů v `t` pro anglický a francouzský jazyk.

4. Pro každý film vypište následující údaje:

(a) počet herců hrajících ve filmu,

(b) počet různých zákazníků, kteří si film půjčili v srpnu,

(c) průměrnou částku platby za výpůjčku filmu.

```
SELECT
  film.film_id,
  film.title,
  (
    SELECT COUNT(*)
    FROM film_actor
    WHERE film_actor.film_id = film.film_id
  ) AS pocet_hercu,
  (
    SELECT COUNT(DISTINCT customer_id)
    FROM
      inventory
      JOIN rental ON inventory.inventory_id = rental.inventory_id
    WHERE
      inventory.film_id = film.film_id
      AND MONTH(rental.rental_date) = 8
  ) AS zak_srp,
  (
    SELECT AVG(amount)
    FROM
      payment
      JOIN rental ON payment.rental_id = rental.rental_id
      JOIN inventory ON rental.inventory_id = inventory.inventory_id
    WHERE inventory.film_id = film.film_id
  ) AS prum_platba
FROM film
```

Stále pracujeme se stejným typem úlohy. Ukažme si zde pouze řešení pomocí poddotazů. První poddotaz vrací ke každému filmu počet herců, druhý počet různých zákazníků se srpnovou výpůjčkou a třetí průměrnou částku platby.

5. Vypište zákazníky, kteří v měsíci červnu provedli více než 5 plateb a nejdelší film, který si půjčili, má alespoň 185 minut.

```
SELECT first_name, last_name
FROM customer
WHERE
  (
    SELECT COUNT(*)
    FROM payment
    WHERE payment.customer_id = customer.customer_id AND MONTH(
       payment_date) = 6
  ) > 5 AND
  (
    SELECT MAX(length)
    FROM
      film
      JOIN inventory ON film.film_id = inventory.film_id
      JOIN rental ON inventory.inventory_id = rental.inventory_id
    WHERE rental.customer_id = customer.customer_id
  ) >= 185
```

Na této úloze vidíme, že poddotazy můžeme bez problémů používat také v klauzuli `WHERE`. První poddotaz vrací pro každého konkrétního zákazníka počet výpůjček provedených v měsíci červnu. Tento počet porovnáme s konstantou 5. Druhý poddotaz vrací maximální délku filmu, který si zákazník půjčil. Výsledek druhého poddotazu pak porovnáváme s konstantou 185.

Pokud by se nám řešení s poddotazy v `WHERE` z nějakého důvodu nehodilo nebo nelíbilo, můžeme využít stejný princip jako v předchozích úlohách:

```
SELECT first_name, last_name
FROM
(
  SELECT first_name, last_name,
    (
      SELECT COUNT(*)
      FROM payment
      WHERE payment.customer_id = customer.customer_id AND MONTH(
          payment_date) = 6
    ) AS pocet,
    (
      SELECT MAX(length)
      FROM
        film
        JOIN inventory ON film.film_id = inventory.film_id
        JOIN rental ON inventory.inventory_id = rental.inventory_id
      WHERE rental.customer_id = customer.customer_id
    ) AS max_delka
  FROM customer
) t
WHERE pocet > 5 AND max_delka >= 185
```

Do „fiktivní“ tabulky `t` si nachystáme jména a příjmení zákazníků společně s počtem červnových filmů a délek jejich nejdelších filmů. Tyto dvě hodnoty si pojmenujeme jako `pocet` a `max_delka`. Vnějším dotazem pak vybereme záznamy z `t`, kde `pocet` a `max_delka` splňují požadovanou podmínku. Dokázali byste druhé z řešení přepsat pomocí CTE (konstrukce `WITH`)?

6. Vypište zákazníky, jejichž většina plateb je o částce větší než 4.

   Nejprve si musíme uvědomit, co znamená „většina plateb“. Znamená to, že zákazník provedl více plateb, kde `amount` > 4, než plateb, kde `amount` ≤ 4. Opět tedy počítáme dvě různé agregace pro záznamy z určité tabulky. Zřejmě nejjednodušší řešení pak bude vypadat takto:

```
SELECT first_name, last_name
FROM customer
WHERE
(
  SELECT COUNT(*)
  FROM payment
  WHERE payment.customer_id = customer.customer_id AND amount > 4
) >
(
  SELECT COUNT(*)
  FROM payment
  WHERE payment.customer_id = customer.customer_id AND amount <= 4
```

```
)
```

Také zde samozřejmě můžeme vyjít z představy nějaké fiktivní tabulky, nad kterou provedeme selekci:

```
SELECT first_name, last_name
FROM
(
  SELECT first_name, last_name,
  (
    SELECT COUNT(*)
    FROM payment
    WHERE payment.customer_id = customer.customer_id AND amount > 4
  ) AS nad_4,
  (
    SELECT COUNT(*)
    FROM payment
    WHERE payment.customer_id = customer.customer_id AND amount <= 4
  ) AS do_4
  FROM customer
) pocty
WHERE nad_4 > do_4
```

7. Vypište herce, kteří hrají více než 2x častěji v komediích než v hororech.

```
SELECT first_name, last_name
FROM actor
WHERE
(
  SELECT COUNT(*)
  FROM film_actor
  WHERE film_actor.actor_id = actor.actor_id AND film_id IN (
    SELECT film_id
    FROM
      film_category
      JOIN category ON film_category.category_id = category.category_id
    WHERE category.name = 'comedy'
  )
)
>
(
  SELECT COUNT(*)
  FROM film_actor
  WHERE film_actor.actor_id = actor.actor_id AND film_id IN (
    SELECT film_id
    FROM
      film_category
      JOIN category ON film_category.category_id = category.category_id
    WHERE category.name = 'horror'
  )
) * 2
```

Řešení této úlohy je podobné řešení předchozí úlohy. Základem je u každého herce spočítat filmy spadající do kategorie „comedy“ a filmy spadající do kategorie „horror“. Pro zjištění, zda film patří do určité kategorie, zde používáme konstrukci `IN`. Nakonec oba počty porovnáme s tím, že počet pro „horror“ nejprve vynásobíme dvěma. Pokud se trochu

ztrácíte v tom, proč je násobení u hororu a ne u komedie, zkuste si představit třeba jednoho herce hrajícího ve 2 komediích a 2 hororech a pak jiného herce hrajícího v 5-ti komediích a 2 hororech. Kterého z herců chceme vypsat?

8. Vypište herce, kteří hrají nejčastěji ve filmech delších než 150 minut, tzn. hrají častěji ve filmech delších než 150 minut než v ostatních filmech.

```
SELECT actor_id, first_name, last_name
FROM actor
WHERE
(
  SELECT COUNT(*)
  FROM film_actor JOIN film ON film_actor.film_id = film.film_id
  WHERE film_actor.actor_id = actor.actor_id AND length > 150
)
>
(
  SELECT COUNT(*)
  FROM film_actor JOIN film ON film_actor.film_id = film.film_id
  WHERE film_actor.actor_id = actor.actor_id AND length <= 150
)
```

Úlohu zde máme pouze pro procvičení stále stejného problému. Princip řešení tedy nebudeme dále podrobně vysvětlovat.

9. Vypište zákazníky, jejichž celkové uhrazené platby jsou menší než kolik by měli zaplatit dle atributů `film.rental_duration`, `film.rental_rate` a rozdílu `rental_date` a `return_date`. Při řešení můžete ignorovat nevrácené výpůjčky.

```
SELECT first_name, last_name
FROM customer
WHERE
(
  SELECT SUM(amount)
  FROM
    rental
    JOIN payment ON rental.rental_id = payment.rental_id
  WHERE rental.customer_id = customer.customer_id
)
<
(
  SELECT SUM(film.rental_rate * DATEDIFF(day, rental.rental_date, rental.
     return_date) / film.rental_duration)
  FROM
    rental
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
    JOIN film ON inventory.film_id = film.film_id
  WHERE rental.customer_id = customer.customer_id
)
```

Základ řešení úlohy je pořád stejný, tj. používáme poddotazy v klauzuli `WHERE`, jejichž výsledky nakonec porovnáváme. Problémem však může být pochopení samotného zadání. Jde totiž o poměrně praktickou záležitost, kdy se zkrátka snažíme odhalit dlužníky. Jak spočítat částku, kterou by zákazník měl za půjčení filmu zaplatit, zjistíme nahlédnutím do datového slovníku (str. 4). Atribut `film.rental_rate` představuje částku, kterou

má zákazník zaplatit za výpůjčku trvající `film.rental_duration` dní. Podílem těchto dvou atributů tedy získáme částku na den. Tuto částku pak násobíme počtem dní výpůjčky. Abychom předešli chybám v zaokrouhlování, jednotlivé operace trochu přeuspořádáme, tj. provedeme nejprve násobení `film.rental_rate` a délky výpůjčky, a až poté dělení atributem `film.rental_duration`.

10. Vypište zákazníky, kteří si častěji půjčovali film, kde hraje TOM MCKELLEN, než film, kde hraje GROUCHO SINATRA.

```
SELECT first_name, last_name
FROM customer
WHERE
(
  SELECT COUNT(*)
  FROM
    rental
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
  WHERE rental.customer_id = customer.customer_id AND film_id IN
  (
    SELECT film_id
    FROM
      actor
      JOIN film_actor ON actor.actor_id = film_actor.actor_id
    WHERE actor.first_name = 'TOM' AND actor.last_name = 'MCKELLEN'
  )
)
>
(
  SELECT COUNT(*)
  FROM
    rental
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
  WHERE rental.customer_id = customer.customer_id AND film_id IN
  (
    SELECT film_id
    FROM
      actor
      JOIN film_actor ON actor.actor_id = film_actor.actor_id
    WHERE actor.first_name = 'GROUCHO' AND actor.last_name = 'SINATRA'
  )
)
```

Ještě jedna úloha na porovnávání agregovaných hodnot v `WHERE`. Nejkomplikovanější je pravděpodobně spočítat počet výpůjček filmů, kde hraje jeden nebo druhý herec. Pro zjištění, zda jde o film, ve kterém hraje daný herec používáme u obou poddotazů konstrukci `IN` podobně jako v úloze 7. U obou úloh (této, i úlohy 7) bychom však mohli místo použití `IN` pokračovat v připojení tabulek pomocí `JOIN`:

```
SELECT first_name, last_name
FROM customer
WHERE
  (
    SELECT COUNT(*)
    FROM
```

```
      rental
      JOIN inventory ON rental.inventory_id = inventory.inventory_id
      JOIN film_actor ON inventory.film_id = film_actor.film_id
      JOIN actor ON film_actor.actor_id = actor.actor_id
    WHERE rental.customer_id = customer.customer_id AND actor.first_name =
        'TOM' AND actor.last_name = 'MCKELLEN'
)
>
(
    SELECT COUNT(*)
    FROM
      rental
      JOIN inventory ON rental.inventory_id = inventory.inventory_id
      JOIN film_actor ON inventory.film_id = film_actor.film_id
      JOIN actor ON film_actor.actor_id = actor.actor_id
    WHERE rental.customer_id = customer.customer_id AND actor.first_name =
        'GROUCHO' AND actor.last_name = 'SINATRA'
)
```

V určitém extrémním případě by však výsledek druhého řešení nemusel být stejný. Dokážete určit, o jaký jde případ? Napovíme, že zatímco v úloze 7 by šlo spíše o situaci vzniklou nesprávným používáním nebo ošetřením systému, v této úloze by mohlo jít o reálnou situaci.

11. Vypište zákazníky, kteří si půjčovali pouze filmy v anglickém jazyce, a u každého napište, kolik mají výpůjček.

```
SELECT first_name, last_name,
(
  SELECT COUNT(*)
  FROM rental
  WHERE rental.customer_id = customer.customer_id
) AS pocet_vypujcek
FROM customer
WHERE NOT EXISTS (
  SELECT *
  FROM
    rental
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
    JOIN film ON inventory.film_id = film.film_id
    JOIN language ON film.language_id = language.language_id
  WHERE rental.customer_id = customer.customer_id AND language.name != '
      English'
) AND customer_id IN (SELECT customer_id FROM rental)
```

Při řešení této úlohy si musíme vzpomenout na předchozí cvičení, konkrétně např. na úlohu 22. Jak poznáme, že si zákazník půjčoval pouze filmy v anglickém jazyce? Tak, že daný zákazník neprovedl žádnou výpůjčku v jiném jazyce ale zároveň provedl alespoň jednu výpůjčku. Toto řeší dvojice konstrukcí `NOT EXISTS` a `IN` ve výše uvedeném řešení. Jelikož nově umíme používat poddotazy za `SELECT`, neměl by pro nás být problém vypsat jakoukoli agregovanou hodnotu, jako např. celkový počet výpůjček, pro každého zákazníka.

V této úloze však poddotazy za `SELECT` vůbec používat nemusíme. K zákazníkům připojíme výpůjčky, výsledek seskupíme podle ID, jména a příjmení, a vypíšeme počet:

```
SELECT first_name, last_name, COUNT(rental.rental_id) AS pocet_vypujcek
FROM
  customer
  JOIN rental ON customer.customer_id = rental.customer_id
WHERE NOT EXISTS (
  SELECT *
  FROM
    rental
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
    JOIN film ON inventory.film_id = film.film_id
    JOIN language ON film.language_id = language.language_id
  WHERE rental.customer_id = customer.customer_id AND language.name != '
     English'
)
GROUP BY customer.customer_id, first_name, last_name
```

Můžeme si všimnout, že se nám ztratila podmínka, která zajišťuje, že zákazník má alespoň jednu výpůjčku. To totiž zajišťuje vnitřní spojení tabulek `customer` a `rental`.

12. Vypište všechny zákazníky, kteří si půjčili film, ve kterém hraje alespoň 15 herců, a u každého z nich vypište celkovou sumu z plateb, které provedli.

```
SELECT
  first_name, last_name,
  (
    SELECT SUM(amount)
    FROM payment
    WHERE payment.customer_id = customer.customer_id
  ) AS celkem
FROM customer
WHERE customer_id IN
(
  SELECT customer_id
  FROM
    rental
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
  WHERE inventory.film_id IN
  (
    SELECT film_id
    FROM film_actor
    GROUP BY film_id
    HAVING COUNT(*) >= 15
  )
)
```

Opět zde máme úlohu, která se silně opírá o znalosti z předchozího cvičení. Pro pořádek si označme celý dotaz jako Q_1, poddotaz za `SELECT` jako Q_2, poddotaz za prvním `IN` jako Q_3 a poddotaz za druhým `IN` jako Q_4. Poslední z poddotazů, tedy Q_4, vrací ID filmů, ve kterých hraje více než 15 herců. Q_3 pak vrací ID zákazníků, kteří si tyto filmy půjčili. Q_2 ke každému takovému zákazníkovi počítá celkovou sumu z plateb a vše dává nakonec dohromady celý dotaz Q_1.

Podobně jako v předchozí úloze, i zde bychom se mohli obejít bez poddotazu Q_2 a použít shlukování (tedy klauzuli `GROUP BY`).

13. Vypište název nejdelšího filmu.

Ukažme si nejprve záměrně špatné řešení:

```
SELECT TOP 1 title
FROM film
ORDER BY length DESC
```

Modifikátor `TOP` n jsme sice zatím nepoužívali (a ani jej používat nebudeme), nicméně tušíme, že takto můžeme omezit výpis na prvních n záznamů (pozor, syntaxe `TOP` je specifická pro Microsoft SQL Server). Myšlenka je taková, že filmy setřídíme sestupně dle délky a z takto setříženého seznamu vypíšeme pouze první film. Co je na tomto řešení špatně? To, že nejdelších filmů může být více.

Správné řešení funguje tak, že nejprve zjistíme délku nejdelšího filmu, tzn. maximální hodnotu atributu `length` v tabulce `film`, a pak vypíšeme všechny filmy, které mají tuto délku:

```
SELECT title
FROM film
WHERE length = (
  SELECT MAX(length)
  FROM film
)
```

Úlohu můžeme elegantně řešit také např. pomocí konstrukce `ALL`, kterou jsme se naučili používat na předchozím cvičení:

```
SELECT title
FROM film
WHERE length >= ALL (
  SELECT length
  FROM film
)
```

Tzn. hledáme takové filmy, jejichž délka je větší nebo rovna délce všech filmů, tzn., že vlastně hledáme nejdelší filmy.

Na minulém cvičení jsme si ale ukazovali, že `ALL`, lze velmi snadno nahradit pomocí `NOT EXISTS`, tzn. můžeme také hledat filmy `f1`, pro které neexistuje delší film `f2`:

```
SELECT title
FROM film f1
WHERE NOT EXISTS (
  SELECT *
  FROM film f2
  WHERE f2.length > f1.length
)
```

Vyhledávání záznamů s extrémní (maximální, minimální apod.) hodnotou v rámci celé tabulky nebo v rámci určité skupiny je poměrně typický problém. Proto se mu budeme věnovat i v několika následujících příkladech.

14. Vypište název nejdelšího filmu pro každou klasifikaci (atribut `film.rating`).

```
SELECT rating, title
FROM film f1
WHERE length = (
  SELECT MAX(length)
```

```
  FROM film f2
  WHERE f1.rating = f2.rating
)
ORDER BY rating
```

Oproti předchozí úloze zde vyhledáváme nejdelší film v rámci určité skupiny. Hledáme filmy, jejichž délka je rovna největší délce filmu v rámci stejné klasifikace. Řešení této úlohy je oproti řešení předchozí úlohy mírně komplikovanější tím, že poddotaz nelze spustit samostatně, tzn. celý dotaz nemůžeme odladit postupně.

Také zde můžeme využít konstrukci `ALL` obdobným způsobem:

```
SELECT rating, title
FROM film f1
WHERE length >= ALL(
  SELECT length
  FROM film f2
  WHERE f1.rating = f2.rating
)
ORDER BY rating
```

Dokázali byste dát dohromady řešení s `NOT EXISTS`?

15. Pro každého zákazníka nalezněte film, který si půjčil naposledy. Setřiďte výsledek abecedně dle příjmení a jmen zákazníků.

```
SELECT customer.customer_id, first_name, last_name, film.title
FROM
  customer
  JOIN rental r1 ON customer.customer_id = r1.customer_id
  JOIN inventory ON r1.inventory_id = inventory.inventory_id
  JOIN film ON inventory.film_id = film.film_id
WHERE r1.rental_date = (
  SELECT MAX(rental_date)
  FROM rental r2
  WHERE r1.customer_id = r2.customer_id
)
ORDER BY last_name, first_name
```

Úloha je v principu podobná předchozí úloze, jen je v řešení nutné spojit více tabulek. Poté vyhledáváme výpůjčky zákazníků, kde datum vypůjčení je rovno maximálnímu datu vypůjčení pro daného zákazníka. Za zmínku stojí ještě fakt, že ve vnitřním dotaze nepracujeme s tabulkou `customer`, jelikož atribut `customer_id` již máme obsažen v tabulce `rental`. Pokud bychom ale tabulku `customer` ve vnitřním dotaze pomocí `JOIN` připojili a v `WHERE` pak porovnávali `customer_id` z tabulek `customer`, nic zásadního by se nestalo.

Pro ukázku uvádíme také řešení pomocí `ALL`:

```
SELECT customer.customer_id, first_name, last_name, film.title
FROM
  customer
  JOIN rental r1 ON customer.customer_id = r1.customer_id
  JOIN inventory ON r1.inventory_id = inventory.inventory_id
  JOIN film ON inventory.film_id = film.film_id
  WHERE r1.rental_date >= ALL (
```

```
    SELECT rental_date
    FROM rental r2
    WHERE r1.customer_id = r2.customer_id
  )
ORDER BY last_name, first_name
```

16. Pro každého herce (jméno a příjmení) vypište název nejdelšího filmu, kde hrál.

```
SELECT actor.first_name, actor.last_name, film.title
FROM
  actor
  JOIN film_actor fa1 ON actor.actor_id = fa1.actor_id
  JOIN film ON fa1.film_id = film.film_id
WHERE film.length >= ALL (
  SELECT film.length
  FROM
    film
    JOIN film_actor fa2 ON film.film_id = fa2.film_id
  WHERE fa2.actor_id = fa1.actor_id
)
```

Pro procvičení řešíme stále stejný typ úlohy. Oproti předchozí úloze mají mezi sebou tabulky `actor` a `film` logickou vazbu M:N, což však na principu řešení nic nemění. Určitému herci odpovídá vždy nějaká skupina filmů a my chceme vyhledat ty filmy, jejichž délka je v rámci této skupiny největší. Ve vnitřním dotazu opět nemusíme připojovat tabulku `actor`, protože používáme atribut `actor_id` rovnou z tabulky `film_actor`. Pro ukázku zde máme ještě řešení pomocí `NOT EXISTS`:

```
SELECT actor.first_name, actor.last_name, f1.title
FROM
  actor
  JOIN film_actor fa1 ON actor.actor_id = fa1.actor_id
  JOIN film f1 ON fa1.film_id = f1.film_id
WHERE NOT EXISTS
(
  SELECT *
  FROM
    film_actor fa2
    JOIN film f2 ON fa2.film_id = f2.film_id
  WHERE fa2.actor_id = actor.actor_id AND f2.length > f1.length
)
```

17. Pro každý film vypište zákazníka, který měl tento film půjčený nejdelší dobu (v rámci jedné výpůjčky). Délku výpůjčky počítejte v sekundách pomocí funkce `DATEDIFF(second, rental_date, return_date)`.

```
SELECT title, first_name, last_name
FROM
  film
  JOIN inventory i1 ON film.film_id = i1.film_id
  JOIN rental r1 ON i1.inventory_id = r1.inventory_id
  JOIN customer ON r1.customer_id = customer.customer_id
WHERE
  DATEDIFF(second, r1.rental_date, r1.return_date) = (
    SELECT MAX(DATEDIFF(second, r2.rental_date, r2.return_date))
    FROM
```

```
      inventory i2
      JOIN rental r2 ON i2.inventory_id = r2.inventory_id
    WHERE i2.film_id = i1.film_id
  )
```

Uvedené řešení počítá pro každou výpůjčku `r1` nejdelší výpůjčku `r2` stejného filmu. Na úloze vidíme, že extrémní hodnota, kterou hledáme, může být také vypočtená – v tomto případě pomocí funkce `DATEDIFF`.

Pokusme se úlohu vyřešit také pomocí konstrukce `ALL`:

```
SELECT title, first_name, last_name
FROM
  film
  JOIN inventory i1 ON film.film_id = i1.film_id
  JOIN rental r1 ON i1.inventory_id = r1.inventory_id
  JOIN customer ON r1.customer_id = customer.customer_id
WHERE
  DATEDIFF(second, r1.rental_date, r1.return_date) >= ALL (
    SELECT DATEDIFF(second, r2.rental_date, r2.return_date)
    FROM
      inventory i2
      JOIN rental r2 ON i2.inventory_id = r2.inventory_id
    WHERE i2.film_id = i1.film_id AND r2.return_date IS NOT NULL
  )
```

Principiálně je toto řešení úlohy podobné, nicméně někoho může překvapit přidaná podmínka `r2.return_date IS NOT NULL`. Můžeme se přesvědčit, že odebráním této podmínky dostaneme jiný výsledek. Problém zde způsobují nevrácené výpůjčky, u kterých funkce `DATEDIFF` vrací hodnotu `NULL`, jelikož jeden z jejích argumentů (`return_date`) je `NULL`. Představme si, že pro konkrétní film f_i vrátí poddotaz v `ALL` délky $\{l_1, \ldots, l_n\}$, kde jedna z délek l_j je `NULL`. Pak logicky musí platit, že celá podmínka `ALL` nemůže být vyhodnocena jako „logická pravda“, protože alespoň v jednom případě porovnání selže (pro l_j). V předchozím řešení pomocí agregační funkce `MAX()` tento problém nenastal, protože agregační funkce hodnoty `NULL` automaticky přeskakují.

18. Pro každého zákazníka vypište poslední vypůjčený film, ve kterém hraje herec PENELOPE GUINESS. Pokud si zákazník film s hercem PENELOPE GUINESS nikdy nepůjčil, nebude zákazník vypsán. Výsledek setřiďte dle ID zákazníků.

```
SELECT customer.customer_id, first_name, last_name, film.title
FROM
  customer
  JOIN rental r1 ON customer.customer_id = r1.customer_id
  JOIN inventory i1 ON r1.inventory_id = i1.inventory_id
  JOIN film ON i1.film_id = film.film_id
WHERE
  film.film_id IN (
    SELECT film_id
    FROM film_actor JOIN actor ON film_actor.actor_id = actor.actor_id
    WHERE actor.first_name = 'PENELOPE' AND actor.last_name = 'GUINESS'
  ) AND r1.rental_date = (
    SELECT MAX(rental_date)
    FROM rental r2 JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
    WHERE r1.customer_id = r2.customer_id AND i2.film_id IN (
```

```
      SELECT film_id
      FROM film_actor JOIN actor ON film_actor.actor_id = actor.actor_id
      WHERE actor.first_name = 'PENELOPE' AND actor.last_name = 'GUINESS'
    )
  )
ORDER BY customer.customer_id
```

Pokud z uvedeného řešení odstraníme podmínky obsahující `IN`, získáme řešení úlohy 15, kde jsme pro každého zákazníka požadovali výpis posledního vypůjčeného filmu (pomiňme trochu jiný požadavek na setřízení výsledku). Podmínky obsahující `IN` v tomto řešení zajišťují, že pracujeme pouze s filmy, kde hraje PENELOPE GUINESS.

Vidíme, že podmínky na jméno herce se v řešení výše opakují. Pokud bychom v budoucnu požadovali jiného herce, museli bychom provést úpravu dotazu na dvou místech. Elegantnější bude použít CTE výraz, tj. konstrukci `WITH`, pomocí které si připravíme tabulku jen těch filmů, kde hraje PENELOPE GUINESS. Takové řešení vypadá následovně:

```
WITH film_pg AS
(
  SELECT film_id, title
  FROM film
  WHERE film_id IN
  (
    SELECT film_id
    FROM film_actor JOIN actor ON film_actor.actor_id = actor.actor_id
    WHERE actor.first_name = 'PENELOPE' AND actor.last_name = 'GUINESS'
  )
)
SELECT customer.customer_id, first_name, last_name, film_pg.title
FROM
  customer
  JOIN rental r1 ON customer.customer_id = r1.customer_id
  JOIN inventory ON r1.inventory_id = inventory.inventory_id
  JOIN film_pg ON inventory.film_id = film_pg.film_id
WHERE r1.rental_date = (
  SELECT MAX(rental_date)
  FROM
    rental r2
    JOIN inventory ON r2.inventory_id = inventory.inventory_id
    JOIN film_pg ON inventory.film_id = film_pg.film_id
  WHERE r1.customer_id = r2.customer_id
)
ORDER BY customer.customer_id
```

Pozor, ve vnitřním dotazu, kde v tabulce `rental r2` hledáme poslední datum výpůjčky stejného zákazníka, je nutné spojit `r2` s tabulkami `inventory` a `film_pg`. Jinak by vnitřní dotaz procházel všechny výpůjčky jakýchkoli filmů (nejen těch, kde hraje PENELOPE GUINESS). Podmínka `WHERE` vnějšího dotazu by pak porovnávala atributy `rental_date` z různých množin výpůjček, což by nevedlo ke správnému výsledku.

19. Vypište zákazníky, kteří si půjčili nejkratší i nejdelší film.

```
SELECT first_name, last_name
FROM customer
WHERE
  customer_id IN
```

```
(
  SELECT rental.customer_id
  FROM
    rental
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
    JOIN film ON inventory.film_id = film.film_id
  WHERE film.length = (SELECT MIN(length) FROM film)
)
AND customer_id IN
(
  SELECT rental.customer_id
  FROM
    rental
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
    JOIN film ON inventory.film_id = film.film_id
  WHERE film.length = (SELECT MAX(length) FROM film)
)
```

Při řešení této úlohy si musíme vzpomenout na minulé cvičení, konkrétně např. na úlohu 3 na straně 38. Řešíme totiž průnik dvou množin: jedna množina představuje ID zákazníků, kteří si půjčili nejkratší film, druhá ID zákazníků, kteří si půjčili nejdelší film. Přesně tyto množiny vrací poddotazy v obou konstrukcích `IN` v uvedeném řešení výše.

20. Vypište herce, kteří hráli alespoň 2x v nejdelším filmu.

```
SELECT actor.actor_id, first_name, last_name
FROM
  actor
  JOIN film_actor ON actor.actor_id = film_actor.actor_id
  JOIN film ON film_actor.film_id = film.film_id
WHERE length = (SELECT MAX(length) FROM film)
GROUP BY actor.actor_id, first_name, last_name
HAVING COUNT(film.film_id) >= 2
```

První uvedené řešení zjistí pro herce v tabulce `actor` skrze vazební tabulku `film_actor` filmy v tabulce `film`, kde tito herci hrají. Podmínkou `WHERE` pak zajistíme, že pracujeme pouze s nejdelšími filmy a s herci, kteří v nich hrají. Poté provádíme shlukování podle herců a zjišťujeme, zda hercům odpovídají alespoň 2 nejdelší filmy.

Při řešení můžeme postupovat také tak, že si připravíme „fiktivní“ tabulku `t` obsahující ID nejdelších filmů. Pak už jde o poměrně jednoduchý agregační dotaz, který místo tabulky `film` používá tabulku `t`:

```
WITH t AS
(
  SELECT film_id
  FROM film
  WHERE length = (SELECT MAX(length) FROM film)
)
SELECT actor.actor_id, actor.first_name, actor.last_name
FROM
  actor
  JOIN film_actor ON actor.actor_id = film_actor.actor_id
  JOIN t ON film_actor.film_id = t.film_id
GROUP BY actor.actor_id, actor.first_name, actor.last_name
HAVING COUNT(film_actor.film_id) >= 2
```

Pro zajímavost si ukažme ještě řešení, které nepoužívá agregační funkce:

```
WITH t AS
(
  SELECT film_id
  FROM film
  WHERE length >= ALL(SELECT length FROM film)
)
SELECT DISTINCT actor.actor_id, first_name, last_name
FROM
  actor
  JOIN film_actor fa1 ON actor.actor_id = fa1.actor_id
  JOIN t t1 ON fa1.film_id = t1.film_id
WHERE EXISTS
(
  SELECT *
  FROM
    film_actor fa2
    JOIN t t2 ON fa2.film_id = t2.film_id
  WHERE fa2.actor_id = fa1.actor_id AND fa2.film_id != fa1.film_id
)
```

Opět si nachystáme tabulku `t`, která obsahuje ID nejdelších filmů. Pomocí ní pak hledáme herce, kteří hrají v nejdelších filmech `t1` tak, aby pro daného herce existoval ještě jiný nejdelší film `t2`, kde herec hraje. Tímto způsobem tedy nalezneme herce hrající alespoň ve dvou různých nejdelších filmech.

21. Vypište filmy, které si alespoň dva zákazníci půjčili naposledy.

```
SELECT film_id, title
FROM
(
  SELECT film.film_id, film.title, customer_id
  FROM
    rental r1
    JOIN inventory ON r1.inventory_id = inventory.inventory_id
    JOIN film ON inventory.film_id = film.film_id
  WHERE r1.rental_date = (
    SELECT MAX(rental_date)
    FROM rental r2
    WHERE r1.customer_id = r2.customer_id
  )
) t
GROUP BY film_id, title
HAVING COUNT(*) >= 2
```

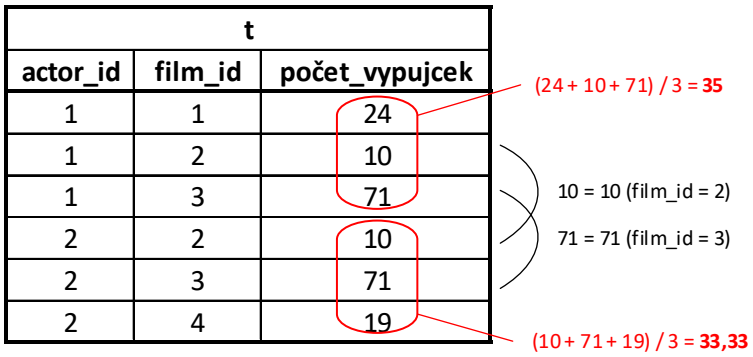
Při řešení této úlohy vyjděme z řešení úlohy 15, kde jsme pro každého zákazníka hledali poslední filmy, které si půjčil. Přesně toto vrací vnitřní poddotaz, jehož výsledek máme pojmenovaný jako `t`. Zbytek už je poměrně jednoduchý, obsah `t` stačí seskupit dle filmu (`film_id` a `title`) a zjistit, zda filmu odpovídají alespoň dva záznamy.

22. Pro každého herce vypište průměrný počet výpůjček za filmy, ve kterých hraje.

V této úloze je pravděpodobně nejsložitější zorientovat se v zadání. Představme si nějakého konkrétního herce a_1, který hraje ve filmech $f_1, \ldots, f_{10}$. Pak dejme tomu, že $c_1, \ldots, c_{10}$ představují počty výpůjček filmů $f_1, \ldots, f_{10}$. Průměrný počet výpůjček za filmy, ve kterých hraje a_1, tedy bude průměr z hodnot $c_1, \ldots, c_{10}$. Tato úvaha vede na následující řešení:

```
SELECT actor_id, first_name, last_name, AVG(CAST(pocet_vypujcek AS FLOAT))
    AS prumer
FROM
(
  SELECT actor.actor_id, first_name, last_name, film.film_id, COUNT(rental
      .rental_id) AS pocet_vypujcek
  FROM
    actor
    LEFT JOIN film_actor ON actor.actor_id = film_actor.actor_id
    LEFT JOIN film ON film_actor.film_id = film.film_id
    LEFT JOIN inventory ON film.film_id = inventory.film_id
    LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
  GROUP BY actor.actor_id, first_name, last_name, film.film_id
) t
GROUP BY actor_id, first_name, last_name
```

Vnitřní dotaz, jehož výsledek máme pojmenovaný jako `t`, vrací kombinace herců a filmů, kde herec hraje v daném filmu. Kromě toho máme k tabulkám `actor`, `film_actor` a `film` připojeny také tabulky `inventory` a `rental`, díky kterým zjistíme navíc také počet výpůjček daného filmu, kde hraje daný herec. Pro lepší představu, pokud tento dotaz spustíme, bude určitý film ve výsledku obsažen několikrát (pokaždé v kombinaci s jiným hercem), ale v každém výskytu mu bude odpovídat stejný počet výpůjček. Výpůjčky filmu totiž nijak nesouvisí s herci, kteří ve filmu hrají. Ukázka toho, co počítá vnitřní dotaz, je znázorněna na Obrázku 5. Vnější dotaz pak seskupí výsledek `t` podle herce a pro každého herce spočítá průměr z odpovídajících hodnot `pocet_vypujcek`. Výpočet vnějšího dotazu je na Obrázku 5 zvýrazněn červeně.



Obrázek 5: Ukázka výpočtu výsledku úlohy 22

Pozor, někteří studenti se obvykle pokouší napsat něco jako `AVG(COUNT(rental_id))`, tedy agregační funkci v agregační funkci, což v SQL nejde, protože je to zkrátka nesmysl. Pokud potřebujeme nějakou agregovanou hodnotu využít v jiné agregační funkci, musíme využít poddotazy nebo třeba CTE výrazy. Uvedené řešení výše je samozřejmě možné velmi snadno přepsat pomocí `WITH` následujícím způsobem:

```
WITH t AS
(
  SELECT actor.actor_id, first_name, last_name, film.film_id, COUNT(rental
      .rental_id) AS pocet_vypujcek
```

```
  FROM
    actor
    LEFT JOIN film_actor ON actor.actor_id = film_actor.actor_id
    LEFT JOIN film ON film_actor.film_id = film.film_id
    LEFT JOIN inventory ON film.film_id = inventory.inventory_id
    LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
  GROUP BY actor.actor_id, first_name, last_name, film.film_id
)
SELECT actor_id, first_name, last_name, AVG(CAST(pocet_vypujcek AS FLOAT))
    AS prumer
FROM t
GROUP BY actor_id, first_name, last_name
```

23. Pro každou klasifikaci filmu (atribut `film.rating`) vypište největší počet herců hrajících ve filmu v dané klasifikaci.

    Opět zde máme úlohu, kdy budeme nad jednou agregací počítat jinou agregaci. První, co však musíme vyřešit je, že v databázi neexistuje žádná tabulka klasifikací, protože klasifikace jsou obsaženy jen v tabulce `film` v rámci atributu `rating`. I když posléze uvidíme, že to není nutné, zkusme využít CTE výraz, který nám připraví „fiktivní" tabulku `rating` obsahující unikátní hodnoty klasifikace. Dále si připravíme tabulku `pocty_hercu`, která ke každému filmu spočítá počet herců a kromě toho vypíše i klasifikaci filmu. Nakonec obě tabulky `rating` a `pocty_hercu` spojíme na základě atributu `rating`, který v tabulce `rating` funguje jako primární klíč a v tabulce `pocty_hercu` jako cizí klíč. Pro každou klasifikaci pak vypíšeme největší hodnotu `pocet`. Řešení tedy bude vypadat takto:

```
WITH
  rating AS (
    SELECT DISTINCT rating
    FROM film
  ),
  pocty_hercu AS (
    SELECT film.rating, film.film_id, COUNT(film_actor.film_id) AS pocet
    FROM
      film
      LEFT JOIN film_actor ON film.film_id = film_actor.film_id
    GROUP BY film.rating, film.film_id
  )
SELECT rating.rating, MAX(pocet) AS max_hercu
FROM
  rating
  LEFT JOIN pocty_hercu ON rating.rating = pocty_hercu.rating
GROUP BY rating.rating;
```

    Uvedené řešení výše jsme zde demonstrovali zejména proto, že existují situace, kdy je vhodné vyčlenit si samostatnou tabulku entit, které jsou fyzicky obsaženy v jiné tabulce. V tomto případě jde o tabulku hodnot s klasifikacemi filmů. Na tuto tabulku pak připojujeme další tabulky, počítáme agregace atd. Na druhou stranu, při troše zamyšlení v tomto případě „fiktivní" tabulku `rating` vůbec nepotřebujeme. Stačí pouze seskupit tabulku `pocty_hercu` podle atributu `rating`, který tato tabulka již obsahuje:

```
WITH pocty_hercu AS (
  SELECT film.rating, film.film_id, COUNT(film_actor.film_id) AS pocet
  FROM
```

```
    film
    LEFT JOIN film_actor ON film.film_id = film_actor.film_id
  GROUP BY film.rating, film.film_id
)
SELECT rating, MAX(pocet) AS max_hercu
FROM pocty_hercu
GROUP BY rating
```

24. Vypište nejčastěji obsazované herce, tj. takové herce, kteří hrají v největším počtu filmů. Počet filmů, ve kterých herec hraje, bude součástí výpisu.

```
SELECT actor.first_name, actor.last_name, COUNT(film_actor.actor_id) AS
   pocet
FROM
  actor
  LEFT JOIN film_actor ON actor.actor_id = film_actor.actor_id
GROUP BY actor.actor_id, actor.first_name, actor.last_name
HAVING COUNT(film_actor.actor_id) =
(
  SELECT MAX(pocet)
  FROM
  (
    SELECT COUNT(film_actor.actor_id) as pocet
    FROM
      actor
      LEFT JOIN film_actor ON actor.actor_id = film_actor.actor_id
    GROUP BY actor.actor_id
  ) t
)
```

Ač se to tak na první pohled nemusí jevit, je tato úloha velmi podobná úloze 13. Hledáme záznam nebo záznamy s maximální hodnotou a to dokonce v rámci celé databáze (což bývá jednodušší než vyhledávání v rámci skupin jako třeba v úloze 14). Komplikované je pouze to, že prohledávané hodnoty jsou vypočtené agregační funkcí `COUNT`. Označme si celý dotaz jako Q_1, poddotaz za `HAVING` jako Q_2 a další vnořený poddotaz za `FROM` v Q_2 jako Q_3. Mělo by být jasné, že Q_3 pro každého herce spočítá počet filmů, ve kterých hraje. Q_2 pak zjistí největší z těchto počtů. Základ dotazu Q_1 je nakonec velmi podobný dotazu Q_3. Q_1 totiž opět počítá počty filmů pro každého herce, ale navíc klauzulí `HAVING` zajišťuje, že počet musí odpovídat výsledku Q_2, tedy největšímu z počtů.

Pokud bychom chtěli zápis trochu zpřehlednit, můžeme místo dotazu Q_2 využít vlastnosti konstrukce `ALL`:

```
SELECT actor.first_name, actor.last_name, COUNT(film_actor.actor_id) AS
   pocet
FROM
  actor
  LEFT JOIN film_actor ON actor.actor_id = film_actor.actor_id
GROUP BY actor.actor_id, actor.first_name, actor.last_name
HAVING COUNT(film_actor.actor_id) >= ALL
(
  SELECT COUNT(film_actor.actor_id) as pocet
  FROM
    actor
    LEFT JOIN film_actor ON actor.actor_id = film_actor.actor_id
```

```
  GROUP BY actor.actor_id, actor.first_name, actor.last_name
);
```

Vrátíme-li se k prvnímu z řešení, pak vidíme, že dotazy Q_1 a Q_3 v podstatě dělají téměř to samé, což by nás mělo vést na následující řešení pomocí konstrukce `WITH`:

```
WITH t AS
(
  SELECT actor.actor_id, actor.first_name, actor.last_name, COUNT(
     film_actor.actor_id) as pocet
  FROM
    actor
    LEFT JOIN film_actor ON actor.actor_id = film_actor.actor_id
  GROUP BY actor.actor_id, actor.first_name, actor.last_name
)
SELECT first_name, last_name, pocet
FROM t
WHERE pocet >= ALL(SELECT pocet FROM t)
```

Nejprve si nachystáme tabulku `t`, kde je u každého herce uveden počet filmů, kde herec hraje. Zbytek dotazu pak téměř totožný řešení úlohy 13.

25. Vypište zákazníky s největším počtem výpůjček.

    Princip řešení této úlohy je naprosto stejný jako u předchozí úlohy, tzn. nebudeme jej podrobně komentovat. Základem je umět sestavit dotaz, který pro každého zákazníka zjišťuje počet výpůjček.

```
SELECT customer.first_name, customer.last_name, COUNT(rental.rental_id) as
    pocet
FROM
  customer
  LEFT JOIN rental ON customer.customer_id = rental.customer_id
GROUP BY customer.customer_id, customer.first_name, customer.last_name
HAVING COUNT(rental.rental_id) = (
  SELECT MAX(pocet)
  FROM
  (
    SELECT COUNT(rental.rental_id) as pocet
    FROM
      customer
      LEFT JOIN rental ON customer.customer_id = rental.customer_id
    GROUP BY customer.customer_id
  ) t
)
```

Použitím konstrukce `ALL` můžeme opět eliminovat jeden z poddotazů:

```
SELECT customer.first_name, customer.last_name, COUNT(rental.rental_id) as
    pocet
FROM
  customer
  LEFT JOIN rental ON customer.customer_id = rental.customer_id
GROUP BY customer.customer_id, customer.first_name, customer.last_name
HAVING COUNT(rental.rental_id) >= ALL (
  SELECT COUNT(rental.rental_id) as pocet
  FROM
```

```
    customer
    LEFT JOIN rental ON customer.customer_id = rental.customer_id
  GROUP BY customer.customer_id
)
```

Zřejmě nejpředhlednější zápis získáme použitím `WITH`:

```
WITH t AS
(
  SELECT customer.customer_id, customer.first_name, customer.last_name,
     COUNT(rental.rental_id) as pocet
  FROM
    customer
    LEFT JOIN rental ON customer.customer_id = rental.customer_id
  GROUP BY customer.customer_id, customer.first_name, customer.last_name
)
SELECT first_name, last_name, pocet
FROM t
WHERE pocet = (SELECT MAX(pocet) FROM t)
```

26. Vypište názvy filmů, které byly vypůjčeny nejvícekrát. Počet výpůjček bude součástí výpisu.

```
SELECT film.film_id, film.title, COUNT(rental.rental_id) AS pocet
FROM
  film
  LEFT JOIN inventory ON film.film_id = inventory.film_id
  LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
GROUP BY film.film_id, film.title
HAVING COUNT(rental.rental_id) = (
  SELECT MAX(pocet)
  FROM
  (
    SELECT COUNT(rental.rental_id) AS pocet
    FROM
      film
      LEFT JOIN inventory ON film.film_id = inventory.film_id
      LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
    GROUP BY film.film_id
  ) t
)
```

Opět řešíme stále stejný typ úlohy, v tomto případě je základem sestavit dotaz, který pro každý film vrátí počet výpůjček. Řešení pomocí `ALL` nebo `WITH` zkuste v rámci procvičení dát dohromady sami.

27. Vypište zákazníky, kteří provedli největší počet plateb. Největší počet plateb vypište také.

```
SELECT customer.first_name, customer.last_name, COUNT(payment.payment_id)
   AS pocet
FROM
  customer
  LEFT JOIN payment ON customer.customer_id = payment.customer_id
GROUP BY customer.customer_id, customer.first_name, customer.last_name
HAVING COUNT(payment.payment_id) >= ALL
(
```

```
  SELECT COUNT(payment.payment_id)
  FROM
    customer
    LEFT JOIN payment ON customer.customer_id = payment.customer_id
  GROUP BY customer.customer_id, customer.first_name, customer.last_name
)
```

Pro procvičení ještě jednou podobný příklad, pouze musíme správně sestavit dotaz zjišťující počet plateb pro jednotlivé zákazníky. Zde pro změnu ukazujeme řešení s `ALL`.

28. Vypište názvy filmů s nadprůměrným počtem výpůjček. Tj., spočítejte průměrný počet výpůjček na filmy a vypište filmy, kde počet výpůjček je nad tímto průměrem.

```
SELECT film.film_id, film.title, COUNT(rental.rental_id) AS pocet
FROM
  film
  LEFT JOIN inventory ON film.film_id = inventory.film_id
  LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
GROUP BY film.film_id, film.title
HAVING COUNT(rental.rental_id) > (
  SELECT AVG(pocet)
  FROM
  (
    SELECT COUNT(rental.rental_id) AS pocet
    FROM
      film
      LEFT JOIN inventory ON film.film_id = inventory.film_id
      LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
    GROUP BY film.film_id
  ) t
)
```

Přestože to na první pohled nemusí být patrné, je řešení této úlohy téměř stejné jako řešení úlohy 26. Jediná změna je v použití agregační funkce `AVG` v prostředním poddotazu. Označíme-li si jednotlivé poddotazy jako Q_1, Q_2 a Q_3 stejným způsobem jako v úloze 24, pak vidíme, že Q_3 vrací počty výpůjček pro jednotlivé filmy. Q_2 pak vrací průměrný počet, který nakonec použijeme v klauzuli `HAVING` dotazu Q_1.

Pro zajímavost si můžeme ukázat, že místo seskupování a klauzule `HAVING` v dotaze Q_1 můžeme počet výpůjček filmu spočítat poddotazem přímo v klauzuli `WHERE`:

```
SELECT film.film_id, film.title
FROM film
WHERE
(
  SELECT COUNT(*)
  FROM inventory JOIN rental ON inventory.inventory_id = rental.
     inventory_id
  WHERE inventory.film_id = film.film_id
)
>
(
  SELECT AVG(pocet_vypujcek)
  FROM
  (
    SELECT film.film_id, film.title, COUNT(rental.rental_id) AS
       pocet_vypujcek
```

```
    FROM
      film
      LEFT JOIN inventory ON film.film_id = inventory.film_id
      LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
    GROUP BY film.film_id, film.title
  ) pocty_vypujcek
)
```

Nakonec bude zřejmě opět nejelegantnější využít konstrukci `WITH`:

```
WITH t AS
(
  SELECT film.film_id, film.title, COUNT(rental.rental_id) AS
     pocet_vypujcek
  FROM
    film
    LEFT JOIN inventory ON film.film_id = inventory.film_id
    LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
  GROUP BY film.film_id, film.title
)
SELECT film_id, title
FROM t
WHERE pocet_vypujcek > (SELECT AVG(pocet_vypujcek) FROM t)
```

29. Vypište herce, kteří hrají nejčastěji ve filmech delších než 150, tzn. v rámci filmů s délkou přes 150 jde o nejčastěji obsazované herce.

```
WITH t AS (
  SELECT actor.actor_id, actor.first_name, actor.last_name, COUNT(film.
     film_id) AS pocet
  FROM
    actor
    LEFT JOIN film_actor ON actor.actor_id = film_actor.actor_id
    LEFT JOIN film ON film_actor.film_id = film.film_id AND film.length >
       150
  GROUP BY actor.actor_id, actor.first_name, actor.last_name
)
SELECT *
FROM t
WHERE pocet = (SELECT MAX(pocet) FROM t)
```

Zadání úlohy i řešení je podobné úloze 24. Rozdíl je, že v tomto případě uvažujeme jen filmy, které splňují nějakou vlastnost. Opět je nutné umět sestavit dotaz, který u každého herce spočítá počet filmů delších než 150. Nemělo by nás překvapit rozšíření spojovací podmínky u `LEFT JOIN`, které jsme si vysvětlovali např. na úloze 23 na straně 34. Na druhou stranu, vzhledem k tomu, že nás zajímají herci, kteří hrají v největším počtu nějakých filmů, nebyla by velká chyba použít vnitřní spojení a podmínku na délku umístit do klauzule `WHERE`:

```
WITH t AS (
  SELECT actor.actor_id, actor.first_name, actor.last_name, COUNT(film.
     film_id) AS pocet
  FROM
    actor
    JOIN film_actor ON actor.actor_id = film_actor.actor_id
    JOIN film ON film_actor.film_id = film.film_id
```

```
  WHERE film.length > 150
  GROUP BY actor.actor_id, actor.first_name, actor.last_name
)
SELECT *
FROM t
WHERE pocet = (SELECT MAX(pocet) FROM t)
```

Vnitřní spojení by nám sice eliminovalo herce, kteří v žádných takových filmech nehrají, ale takoví herci nás vlastně stejně nakonec nezajímají. Mohla by ale nastat extrémní situace, která by vedla k odlišnému výsledku. Víte jaká?

Na této úloze si můžeme všimnout ještě jedné zajímavosti – zadání je totiž velmi podobné zadání úlohy 8. To, že herec hraje nejčastěji ve filmech delších než 150 lze totiž interpretovat dvěma způsoby. První je ten, že herec hraje více ve filmech, které jsou delší než 150, než ve filmech s délkou do 150 včetně, a to je právě zadání úlohy 8. Druhá interpretace je, že v rámci filmů s délkou přes 150 byl obsazen nejčastěji, což řešíme v této úloze.

30. Vypište zákazníky s největším rozdílem nimimální a maximální platby za výpůjčku filmu započatou v červnu. Rozdíl bude součástí výpisu.

```
WITH t AS
(
  SELECT customer.customer_id, customer.first_name, customer.last_name,
     MAX(amount) - MIN(amount) AS rozdil
  FROM
    customer
    LEFT JOIN rental ON customer.customer_id = rental.customer_id AND
       MONTH(rental.rental_date) = 6
    LEFT JOIN payment ON rental.rental_id = payment.rental_id
  GROUP BY customer.customer_id, customer.first_name, customer.last_name
)
SELECT first_name, last_name
FROM t
WHERE rozdil = (SELECT MAX(rozdil) FROM t)
```

Typově je úloha velmi podobná předchozí úloze, takže řešení nebudeme podrobně vysvětlovat. Zvláštností je pouze to, že místo počtu zjišťujeme rozdíl mezi minimem a maximem určitého atributu.

31. Vypište filmy, které byly vypůjčeny jedním zákazníkem nejvícekrát.

```
WITH t AS
(
  SELECT customer.customer_id, customer.first_name, customer.last_name,
     film.film_id, film.title, COUNT(rental.rental_id) AS pocet
  FROM
    customer
    LEFT JOIN rental ON customer.customer_id = rental.customer_id
    LEFT JOIN inventory ON rental.inventory_id = inventory.inventory_id
    LEFT JOIN film ON inventory.film_id = film.film_id
  GROUP BY customer.customer_id, customer.first_name, customer.last_name,
     film.film_id, film.title
)
SELECT DISTINCT title
FROM t
WHERE pocet = (SELECT MAX(pocet) FROM t)
```

Přestože je princip řešení úplně stejný jako např. v úloze 14, může nás zadání při prvním přečtení trochu zmást. Zde totiž nevyhledáváme konkrétní entity (filmy, zákazníky, herce atd.), jimž odpovídá nějaká extrémní hodnota, ale kombinace film - zákazník. Musíme tedy sestavit relativně jednoduchý agregační dotaz, který pro každou dvojici film - zákazník zjistí, kolikrát si daný film daný zákazník půjčil. Zbytek řešení by nám měl být již známý, tj. vybereme ty kombinace, kde je zjištěná hodnota největší. Z těchto kombinací nás podle zadání nakonec zajímá jen film, resp. jeho název. Jelikož se název filmu může opakovat (může existovat více zákazníků, kteří si daný film půjčili nejvícekrát), měli bychom duplicity odstranit pomocí `DISTINCT`.

32. Vypište zákazníky, kteří si nejvícekrát půjčili stejný film.

```
WITH t AS
(
  SELECT customer.customer_id, customer.first_name, customer.last_name,
     film.film_id, film.title, COUNT(rental.rental_id) AS pocet
  FROM
    customer
    LEFT JOIN rental ON customer.customer_id = rental.customer_id
    LEFT JOIN inventory ON rental.inventory_id = inventory.inventory_id
    LEFT JOIN film ON inventory.film_id = film.film_id
  GROUP BY customer.customer_id, customer.first_name, customer.last_name,
     film.film_id, film.title
)
SELECT DISTINCT first_name, last_name
FROM t
WHERE pocet = (SELECT MAX(pocet) FROM t)
```

Na této úloze je zřejmě nejsložitější uvědomit si, že řešíme v podstatě to samé jako v předchozí úloze. Rozdíl je pouze v tom, že místo názvů filmů budeme vypisovat jména a příjmení zákazníků.

33. Pro každé město vypište zákazníka s největším počtem výpůjček.

```
SELECT c1.city_id, city, customer.customer_id, first_name, last_name,
   COUNT(rental.rental_id) AS pocet
FROM
  city c1
  LEFT JOIN address ON c1.city_id = address.city_id
  LEFT JOIN customer ON address.address_id = customer.customer_id
  LEFT JOIN rental ON customer.customer_id = rental.customer_id
GROUP BY c1.city_id, city, customer.customer_id, first_name, last_name
HAVING COUNT(rental.rental_id) =
(
  SELECT MAX(pocet)
  FROM
  (
    SELECT COUNT(rental.rental_id) AS pocet
    FROM
      city c2
      LEFT JOIN address ON c2.city_id = address.city_id
      LEFT JOIN customer ON address.address_id = customer.customer_id
      LEFT JOIN rental ON customer.customer_id = rental.customer_id
    WHERE c2.city_id = c1.city_id
    GROUP BY customer.customer_id
```

```
    ) t
)
```

Oproti předchozí úloze se tato principiálně liší tím, že nevyhledáváme entity s extrémní hodnotou v rámci celé databáze, ale v rámci určitých skupin podobně jako např. v úloze 14. Oproti úloze 14 je však tato navíc komplikovanější nutností použití agregační funkce `COUNT` podobně jako např. v úloze 24.

Jelikož jde o poměrně častý typ úlohy, pojďme si zpracování dotazu z logického pohledu trochu podrobněji rozebrat. Rozdělme si celý dotaz tak jako v úloze 24 na Q_1 označující celý dotaz, Q_2 jako poddotaz za `HAVING` a Q_3 jako poddotaz za `FROM` v Q_2. Zatímco v úloze 24 bylo možné vyhodnocování celého dotazu vysvětlit jednoduše postupným vyhodnocením Q_3, Q_2 a Q_1, zde toto nelze, protože dotaz Q_3 je závislý na dotazu Q_1. Mělo by být jasné, že Q_1 zjišťuje ke každé kombinaci město - zákazník, kde zákazník bydlí v daném městě, počet výpůjček daného zákazníka. Představme si, určitého zákazníka z, město m a počet výpůjček zákazníka z označený jako p. Při vyhodnocování klauzule `HAVING` dotazu Q_1 bude nejprve vyhodnocen dotaz Q_3, který pro každého zákazníka z města m zjistí počet výpůjček. Ze zjištěných počtů pak Q_2 vybere maximum. Nakonec zpět v klauzuli `HAVING` dojde k porovnání p a počtu vráceného Q_2.

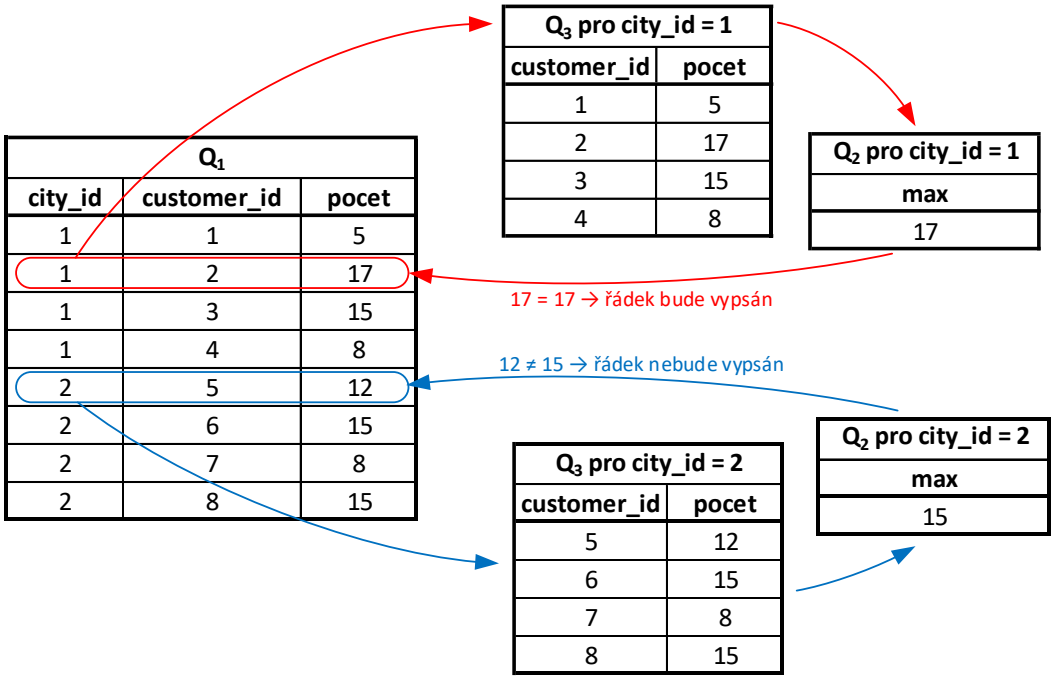
Pro lepší představu je průběh zpracování celého dotazu znázorněn na Obrázku 6. Vidíme, že např. pro město s ID = 1 máme zákazníka s ID = 2, který má 17 výpůjček a 17 je největší z počtů výpůjček pro zákazníky bydlící ve městě s ID = 1. Proto zákazník s ID = 2 bude pro město s ID = 1 vypsán. Naopak pro město s ID = 2 máme zákazníka s ID = 5, který má 12 výpůjček, ale největší z počtů výpůjček ve městě s ID = 2 je 17. Proto zákazník s ID = 5 pro město s ID = 2 vypsán nebude. Opět jen upozorňujeme, že jde pouze o logickou představu průběhu zpracování dotazů. Prakticky by bylo velmi neefektivní pro každý záznam z Q_1 opakovaně spouštět poddotazy. Databázový systém pravděpodobně nalezne efektivnější strategii.

Pro zpřehlednění zápisu můžeme nakonec použít také CTE výraz. Řešení je pak v podstatě velmi podobné řešení úlohy 14, pouze místo tabulky `film` používáme „fiktivní“ tabulku připravenou pomocí `WITH`:

```
WITH t AS
(
  SELECT c1.city_id, city, customer.customer_id, first_name, last_name,
     COUNT(rental.rental_id) AS pocet
  FROM
    city c1
    LEFT JOIN address ON c1.city_id = address.city_id
    LEFT JOIN customer ON address.address_id = customer.customer_id
    LEFT JOIN rental ON customer.customer_id = rental.customer_id
  GROUP BY c1.city_id, city, customer.customer_id, first_name, last_name
)
SELECT *
FROM t t1
WHERE pocet >= ALL(SELECT pocet FROM t t2 WHERE t1.city_id = t2.city_id)
```

34. U každého zákazníka vypište, kolikrát si půjčil nějaký film nejčastěji a který film to byl. Ignorujte zákazníky bez výpůjčky.

```
WITH t AS
```



Obrázek 6: Ukázka výpočtu výsledku úlohy 33

```
(
  SELECT customer.customer_id, customer.first_name, customer.last_name,
     film.film_id, film.title, COUNT(rental.rental_id) AS pocet
  FROM
    customer
    JOIN rental ON customer.customer_id = rental.customer_id
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
    JOIN film ON inventory.film_id = film.film_id
  GROUP BY customer.customer_id, customer.first_name, customer.last_name,
     film.film_id, film.title
)
SELECT DISTINCT first_name, last_name, title, pocet
FROM t t1
WHERE pocet = (SELECT MAX(pocet) FROM t t2 WHERE t1.customer_id = t2.
   customer_id)
```

Tentokrát si vysvětlíme rovnou řešení s `WITH`. Nejprve je nutné dát dohromady agregační dotaz, který ke každému zákazníkovi a filmu, který si půjčil, zjistí počet výpůjček. Výsledek tohoto dotazu si označíme jako `t`. Zbytek řešení je pak podobný jako v předchozí úloze, tzn. nalezneme takové zákazníky a filmy, kde je zjištěný počet v rámci stejného zákazníka největší. V rámci procvičení si zkuste uvedené řešení přepsat bez použití `WITH`.

35. Pro každou kategorii vypište filmy s nejmenším počtem výpůjček.

```
WITH t AS
(
```

```
  SELECT category.category_id, category.name, film.film_id, film.title,
     COUNT(rental.rental_id) AS pocet
  FROM
    category
    JOIN film_category ON category.category_id = film_category.category_id
    JOIN film ON film_category.film_id = film.film_id
    LEFT JOIN inventory ON film.film_id = inventory.film_id
    LEFT JOIN rental ON inventory.inventory_id = rental.inventory_id
  GROUP BY category.category_id, category.name, film.film_id, film.title
)
SELECT *
FROM t t1
WHERE pocet = (SELECT MIN(pocet) FROM t t2 WHERE t1.category_id = t2.
   category_id)
ORDER BY category_id
```

Základ řešení je opět podobný jako v předchozích úlohách. Nejprve musíme dát dohromady agregační dotaz, který pro každou kategorii a film v této kategorii zjistí počet výpůjček. Je zde pouze malá záludnost v tom, že nepožadujeme největší počty ale nejmenší počty. Nesmíme totiž zapomenout správně použít `LEFT JOIN`, tedy vnější spojení, pro připojení tabulek `inventory` a `rental`. Jinak bychom ignorovali filmy, ke kterým se nevztahuje žádná výpůjčka a to by nebylo správně. Klauzuli `ORDER BY` máme v řešení spíše z kosmetických důvodů, aby bylo na první pohled vidět požadované filmy pro jednotlivé kategorie.

36. Pro každou kategorii filmu vypište nejčastěji obsazované herce hrající ve filmu v dané kategorii.

```
WITH t AS
(
  SELECT category.category_id, category.name, actor.actor_id, actor.
     first_name, actor.last_name, COUNT(film.film_id) AS pocet
  FROM
    category
    JOIN film_category ON category.category_id = film_category.category_id
    JOIN film ON film_category.film_id = film.film_id
    JOIN film_actor ON film.film_id = film_actor.film_id
    JOIN actor ON film_actor.actor_id = actor.actor_id
  GROUP BY category.category_id, category.name, actor.actor_id, actor.
     first_name, actor.last_name
)
SELECT *
FROM t t1
WHERE pocet = (SELECT MAX(pocet) FROM t t2 WHERE t1.category_id = t2.
   category_id)
ORDER BY category_id
```

Stále pracujeme s podobným typem úlohy, kde je nutností dobře zvládat psaní agregačních dotazů. Začneme tedy dotazem, který pro každou filmovou kategorii zjistí, kolikrát v ní hrál určitý herec. Poté vybereme ty kategorie a herce, kde je zjištěný počet pro danou kategorii největší. Klauzule `ORDER BY` opět opticky seskupí jednotlivé kategorie podobně jako v předchozí úloze.

37. Pro každého zákazníka vypište jeho nejoblíbenějšího herce, tj. herce, který hrál v nejvíce různých filmech, které si zákazník půjčil. Ignorujte zákazníky bez výpůjčky.

```
WITH t AS (
  SELECT
    customer.customer_id, customer.first_name AS c_first_name, customer.
       last_name AS c_last_name,
    actor.actor_id, actor.first_name AS a_first_name, actor.last_name AS
       a_last_name,
  COUNT(DISTINCT film.film_id) AS pocet
  FROM
    customer
    JOIN rental ON customer.customer_id = rental.customer_id
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
    JOIN film ON inventory.film_id = film.film_id
    JOIN film_actor ON film.film_id = film_actor.film_id
    JOIN actor ON film_actor.actor_id = actor.actor_id
  GROUP BY customer.customer_id, customer.first_name, customer.last_name,
     actor.actor_id, actor.first_name, actor.last_name
)
SELECT *
FROM t t1
WHERE pocet = (SELECT MAX(pocet) FROM t t2 WHERE t1.customer_id = t2.
   customer_id)
ORDER BY customer_id
```

Pro procvičení také tato úloha řeší stejný typ problému jako předchozí úlohy. Nejprve je nutné umět spočítat pro každého zákazníka, v kolika různých filmech, které si zákazník půjčil, hrál určitý herec. Pozor, v tomto případě je nezbytné použít agregační funkci `COUNT(DISTINCT)`, jinak by nastal problém v situaci, kdy by si zákazník půjčil nějaký film vícekrát. Použití klauzule `ORDER BY` má stejný význam jako v předchozích dvou úlohách.

6 Příkazy pro modifikaci a definici dat

Až do této chvíle jsme v naší databázi neprováděli žádné úpravy. Používali jsme pouze jediný příkaz SELECT, jehož možnosti jsou sice obrovské, ale samotná data a struktura databáze zůstávají nedotčeny. Dnes si naopak ukážeme příkazy spadající do kategorie DML (Data Manipulation Language) pro úpravu obsahu tabulek a příkazy spadající do kategorie DDL (Data Definition Language) pro úpravu struktury tabulek.

Toto cvičení se bude svou strukturou mírně odlišovat od těch předchozích z několika důvodů. Některé úlohy se budou skládat z více bodů, které je nutné řešit v přesně daném pořadí. Dále, pokud nebude přímo uvedeno, můžete úlohy řešit i více SQL příkazy, které budete spouštět postupně.

Než se pustíme do řešení úloh, poznamenejme, že zatímco syntaxe příkazu SELECT bývá napříč různými SŘBD téměř stejná (je dodržován standard ANSI SQL), u příkazů v kategoriích DML a DDL se často objevují mírné rozdíly. V této sbírce budeme ukazovat syntaxi pro Microsoft SQL Server. Principiálně se ale řešení úloh pro jiné relační SŘBD nebude lišit.

1. (a) Vložte do databáze nového herce se jménem Arnold Schwarzenegger. Pro časové razítko poslední aktualizace záznamu (`last_update`) ponechte výchozí hodnotu (tzn. tuto hodnotu nenastavujte).

```
INSERT INTO actor (first_name, last_name)
VALUES ('Arnold', 'Schwarzenegger');
```

Jistě jste pochopili, že příkaz INSERT vloží do tabulky `actor` nový řádek (odborně záznam). Ačkoli je seznam atributů za názvem tabulky v SQL nepovinný, měli bychom jej vždy explicitně uvádět z několika důvodů:

- Ne všechny atributy jsou v tabulce povinné, některé atributy mohou být nastaveny na výchozí hodnotu (jako v tomto případě atribut `last_update`) a nesmíme explicitně zadávat hodnotu pro automaticky generovaná ID (zde `actor_id`).
- Vynecháním závorky se spoléháte na určité pořadí sloupců (atributů) v tabulce. Může se ale velmi snadno stát, že v jiné databázi bude pořadí sloupců jiné. Pokud se např. budete spoléhat na pořadí sloupců ve vaší lokální testovací databázi, pak po nasazení do produkční databáze může nastat velký průšvih – data se budou zapisovat tam, kam nemají!
- Poslední důvod je tak trochu psychologický. Explicitním uváděním atributů si po čase lépe zapamatujete strukturu tabulek. Jinými slovy, lenost se zde nevyplácí.

(b) Vložte do databáze film Terminátor. Popis a délku filmu zjistěte např. v databázi CSFD[3]. Jazyk filmu nastavte na Angličtinu, standardní dobu výpůjčky na 3 dny a cenu na 1,99. Ostatní atributy budou nevyplněné nebo budou nastaveny na výchozí hodnotu.

```
INSERT INTO film (title, description, language_id, rental_duration,
    rental_rate, length)
VALUES ('Terminátor', 'Z roku 2029 je do Los Angeles roku 1984
   vyslán zabijácký stroj podobný člověku...', 1, 3, 1.99, 107);
```

[3] https://www.csfd.cz/film/1249-terminator/videa/#play-video-157711308

V této úloze si ještě jednou vyzkoušíme obyčejný INSERT. Všimněme si, že explicitním uvedením seznamu atributů si výrazně usnadníme práci – v tabulce `film` je totiž celkem 14 atributů, zatímco my jsme vyplnili pouze 6.

(c) Upravte obsah databáze tak, aby herec Arnold Schwarzenegger hrál ve filmu Terminátor. ID herce a filmu si předem zjistěte vhodnými dotazy.

Nejprve je tedy nutné zjistit si ID, která byla přiřazena námi vytvořeným záznamům. Můžeme použít například tyto dva jednoduché dotazy:

```
SELECT film_id
FROM film
WHERE title = 'Terminátor';

SELECT actor_id
FROM actor
WHERE last_name = 'Schwarzenegger';
```

Dejme tomu, že dotazy vrátí postupně hodnoty `$x` a `$y`. Toto značení budeme používat i v následujících úlohách k označení „fiktivních“ proměnných. Nejde tedy o součást syntaxe SQL – bude to pouze pomocné značení pro naše účely, abychom se vyhnuli konkrétním konstantám, které může mít každý v databázi trochu jinak.

Poznámka pro zvědavější studenty: V databázích, které umožňují automaticky generovat ID, obvykle existují speciální funkce pro zjištění ID, které bylo vygenerováno naposled. Zjišťovat toto poslední ID pomocí dotazů jako `SELECT MAX(id) FROM tabulka` nemusí vést ke správnému výsledku. Na Microsoft SQL Serveru můžeme např. použít dotaz `SELECT @@IDENTITY`, který vrátí poslední vygenerované ID pro jakoukoli tabulku, nebo `SELECT IDENT_CURRENT('název tabulky')`, který vrátí poslední vygenerované ID pro určitou konkrétní tabulku. Tyto funkce zde uvádíme spíše pro úplnost – můžou se Vám hodit, až budete vyvíjet nějakou reálnou aplikaci informačního systému.

Mělo by být jasné, že přiřazení herce k filmu provedeme vložením záznamu do tabulky `film_actor`, a to následujícím příkazem:

```
INSERT INTO film_actor (film_id, actor_id) VALUES ($x, $y);
```

(d) Zařaďte film Terminátor do kategorií „Action“ a „Sci-Fi“. ID příslušných záznamů zjistěte předem vhodnými dotazy.

Princip řešení této úlohy je obdobný jako u předchozí úlohy. ID filmu, které si označíme jako `$x` zjistíme např. dotazem:

```
SELECT film_id
FROM film
WHERE title = 'Terminátor';
```

Pro zjištění kategorií „Action“ a „Sci-Fi“ bude nejsnazší zobrazit si celou tabulku `category`. Dejme tomu, že ID příslušných kategorií budou `$y` a `$z`.

```
SELECT *
FROM category;
```

Film pak k daným kategoriím přiřadíme příkazy:

```
INSERT INTO film_category (film_id, category_id) VALUES ($x, $y);
INSERT INTO film_category (film_id, category_id) VALUES ($x, $z);
```

(e) Zařaďte film Terminátor do kategorie „Comedy“. Úlohu vyřešte jedním příkazem za použití poddotazů tak, abyste se vyhnuli ručnímu zápisu konstant pro ID filmu a kategorie. Potřebná ID dohledejte podle názvů (`film.title` a `category.name`).

```
INSERT INTO film_category (film_id, category_id) VALUES
(
  (SELECT film_id FROM film WHERE title = 'Terminátor'),
  (SELECT category_id FROM category WHERE name = 'Comedy')
);
```

Zatímco u předchozích úloh bylo řešení zjednodušené tím, že jednotlivá ID jsme si mohli předem zjistit libovolným způsobem, zde je požadováno vyřešit úlohu jedním příkazem (tj. na jedno stisknutí F5). Nemělo by být překvapením, že DML příkazy, jako je např. INSERT, se velmi často kombinují se SELECT dotazy. V tomto případě tedy místo konstant pro ID filmu a ID kategorie použijeme poddotazy. Je samozřejmě nutné, aby oba tyto poddotazy vrátily přesně jedenu hodnotu – tj., abychom v databázi např. neměli dva filmy se jménem „Terminátor“.

(f) Upravte cenu výpůjčky filmu Terminátor na hodnotu 2,99. Upravte současně také atribut `last_update` podle aktuálního časového razítka.

```
UPDATE film
SET rental_rate = 2.99, last_update = CURRENT_TIMESTAMP
WHERE title = 'Terminátor';
```

Příkaz UPDATE je dalším zástupcem příkazů DML. Tento příkaz mění hodnotu jednoho nebo více atributů pro vybrané řádky. Klauzule `WHERE` zde funguje obdobně jako u příkazu `SELECT`. Pro získání aktuálního data a času můžeme použít standardní vestavěnou funkci `CURRENT_TIMESTAMP`.

Někoho by možná napadlo sestavit podmínku na základě ID filmu a poddotazu:

```
UPDATE film
SET rental_rate = 2.99, last_update = CURRENT_TIMESTAMP
WHERE film_id = (SELECT film_id FROM film WHERE title = 'Terminátor');
```

Toto řešení by bylo samozřejmě správně, ale výsledek by byl stejný jako v předchozím případě. Malý rozdíl v obou zápisech ale bude – víte, jaký?

Nakonec jedno velké upozornění! Klauzule `WHERE` není u příkazu UPDATE povinná. Pokud ji neuvedeme nebo zapomeneme, aktualizují se hodnoty v celé tabulce. A to může být v reálné produkční databázi velký problém!

2. (a) Vytvořte zaměstnance s Vaším jménem a Vaší adresou (údaje o adrese samozřejmě mohou být smyšlené). Uživatelské jméno bude Váš login a budete zařazeni do skladu s ID = 2. Potřebné konstanty pro cizí klíče zjistěte předem vhodnými dotazy.

Tuto úlohu zde zařazujeme zejména pro procvičení příkazu INSERT. Abychom mohli vytvořit zaměstnance, musíme mu nejprve vytvořit adresu. S tím pak souvisí to, že adresa se musí vztahovat k nějakému městu a to musí být umístěno v nějakém státě. Ve výsledku tedy budeme vkládat záznamy do tabulek `staff`, `address`, nejspíše do `city` a možná i do `country`.

Úloha by tedy měla být řešena v této posloupnosti:

i. Nejprve zjistíme, zda se v databázi nachází záznam o našem státu (Česká republika i Slovensko v databázi jsou). Můžeme si tedy pomoct například tímto dotazem:

```
SELECT *
FROM country
ORDER BY country;
```

Zjištěné ID státu si označme jako `$x`.

ii. Dále zjistíme, zda se v daném státě nachází naše město:

```
SELECT *
FROM city
WHERE country_id = $x
ORDER BY city;
```

iii. Pokud ne (tj. pokud nejste z Olomouce), je potřeba vložit záznam:

```
INSERT INTO city (city, country_id)
VALUES ('Ostrava', $x);
```

iv. Pokud jsme vkládali záznam, je potřeba si zapamatovat ID města – proměnná `$y`.

```
SELECT *
FROM city
WHERE city = 'Ostrava';
```

v. Teprve teď můžeme vložit adresu:

```
INSERT INTO address (address, district, city_id, phone)
VALUES ('Testova␣123', 'Okres␣Ostrava', $y, '+420␣601␣001␣001');
```

vi. Zjistíme ID vložené adresy (`$z`):

```
SELECT *
FROM address
WHERE address = 'Testova␣123'
```

vii. Nakonec můžeme vložit samotného zaměstnance:

```
INSERT INTO staff (first_name, last_name, address_id, store_id,
   username)
VALUES ('Jan', 'Novak', $z, 2, 'nov001');
```

(b) Vytvořte v databázi adresu naší univerzity.

Vzhledem k tomu, že záznam pro město Ostrava už v databázi pravděpodobně máte (po vyřešení předchozího bodu), pouze si do proměnné `$x` zapamatujme ID našeho města.

```
SELECT *
FROM city
WHERE city = 'Ostrava';
```

Za pomocí tohoto ID vložíme adresu:

```
INSERT INTO address (address, district, city_id, phone)
VALUES ('17.␣listopadu␣2172/15', 'Okres_Ostrava', $x, '+420␣597␣326␣
   001');
```

(c) Vytvořte nový sklad na adrese naší univerzity. V novém skladu budete Vy manažerem.
Pomocí jednoduchých dotazů si nejprve zjistěme naše ID a ID adresy naší univerzity (proměnné `$x` a `$y`).

```
SELECT *
FROM staff;

SELECT *
FROM address
WHERE address = '17. listopadu 2172/15';
```

Následující INSERT by pro nás neměl být problém:

```
INSERT INTO store (manager_staff_id, address_id)
VALUES ($x, $);
```

(d) Pro každý film, který půjčovna vlastní alespoň v jedné kopii, přesuňte do nového skladu (viz předchozí bod) jeho kopii s nejvyšším ID.

Nejprve si jako obvykle zjistíme ID skladu. Označme si jej jako `$s`. Jelikož víme, že skladů je málo, bude stačit např. tento triviální dotaz:

```
SELECT *
FROM store
```

Co bude určitě složitější – jak vybrat příslušné kopie. Základem je tedy umět dát dohromady dotaz, který vrátí ID posledních kopií filmů (myšleno podle `inventory_id`):

```
SELECT i1.inventory_id
FROM inventory i1
WHERE i1.inventory_id >= ALL(
  SELECT i2.inventory_id
  FROM inventory i2
  WHERE i1.film_id = i2.film_id
)
```

Pokud konstrukce dotazu není jasná, vraťte se k úloze 14 na straně 71.

Dotaz pak můžeme velmi snadno začlenit do podmínky `WHERE` příkazu UPDATE. Řešení úlohy by tedy mohlo vypadat např. takto:

```
UPDATE inventory
SET store_id = $s
WHERE inventory_id IN (
  SELECT i1.inventory_id
  FROM inventory i1
  WHERE i1.inventory_id >= ALL(
    SELECT i2.inventory_id
    FROM inventory i2
    WHERE i1.film_id = i2.film_id
  )
)
```

Aktualizujeme tedy všechny záznamy, jejichž ID spadá (konstrukce `IN`) do nějaké množiny, kterou vrací poddotaz.

Předchozí řešení je správně, nicméně zápis můžeme dále zjednodušit tím, že logiku dotazu integrujeme přímo do samotného UPDATE:

```
UPDATE inventory
SET store_id = $s
WHERE inventory_id >= ALL(
  SELECT i2.inventory_id
  FROM inventory i2
  WHERE inventory.film_id = i2.film_id
)
```

Nakonec si spíše pro zajímavost ukažme syntaxi UPDATE, který může obsahovat klauzuli `FROM`. Syntaxe je poměrně užitečná, nicméně bohužel specifická pro Microsoft SQL Server:

```
UPDATE i1
SET store_id =$s
FROM inventory i1
WHERE i1.inventory_id >= ALL(
  SELECT i2.inventory_id
  FROM inventory i2
  WHERE i1.film_id = i2.film_id
)
```

Klauzule `UPDATE` a `SET` pochází z příkazu UPDATE. Zbytek (od klauzule `FROM`) pak připomíná spíše klasický dotaz SELECT. Všimněme si, že pokud tabulce za `FROM` přiřadíme alias, musíme tento alias použít za klíčovým slovem `UPDATE` místo názvu původní tabulky.

3. Navyšte cenu výpůjčky všech filmů, ve kterých hraje herec ZERO CAGE, o 10 %. Úlohu vyřešte jedním příkazem bez zápisu konstanty pro ID herce (herec bude identifikován svým jménem).

```
UPDATE film
SET rental_rate = rental_rate * 1.1
WHERE film_id IN (
  SELECT film_id
  FROM
    film_actor
    JOIN actor ON film_actor.actor_id = actor.actor_id
  WHERE first_name = 'ZERO' AND last_name = 'CAGE'
);
```

Nejprve si samozřejmě musíme správně sestavit dotaz, který vrací ID filmů, kde hraje herec ZERO CAGE. Nad tímto dotazem pak postavíme příkaz UPDATE, který u vybraných filmů vynásobí cenu výpůjčky konstantou 1,1. Mělo by být zřejmé, že toto násobení představuje ono zdražení o 10 %.

Všimněte si, že se v klauzuli `SET` můžeme bez problémů odkazovat na původní hodnoty záznamu (tj. např. na původní `rental_rate`). Pro zjištění původního `rental_rate`, které chceme vynásobit 1,1, rozhodně není nutné psát poddotaz.

4. Všem filmům, jejichž původní jazyk (`original_language`) je Mandarínština (Mandarin), nastavte původní jazyk na `NULL`. Vyhněte se zápisu ID pro daný jazyk.

```
UPDATE film
SET original_language_id = NULL
WHERE original_language_id =
  (SELECT language_id FROM language WHERE name = 'Mandarin');
```

Úlohu zde zařazujeme pouze pro procvičení příkazu UPDATE s použitím poddotazu. Poddotazem tedy zjistíme ID jazyku Mandarínština. Pro filmy v tomto jazyce pak provedeme nastavení `original_language_id` na `NULL`.

5. Pro každý film, ve kterém hraje herec GROUCHO SINATRA, vložte do tabulky `inventory` jednu novou kopii. Všechny tyto nové kopie budou umístěny ve skladu s ID = 2. Datum poslední aktualizace záznamu ponechejte na výchozí hodnotě. Úlohu opět vyřešte jedním příkazem bez zápisu konstanty pro ID herce (herec bude identifikován svým jménem).

```
INSERT INTO inventory (film_id, store_id)
SELECT film_id, 2
FROM
  actor
  JOIN film_actor ON actor.actor_id = film_actor.actor_id
WHERE first_name = 'GROUCHO' AND last_name = 'SINATRA';
```

Zatímco u předchozích úloh jsme u příkazu INSERT psali klauzuli `VALUES` (tj. vyjmenovávali jsme konkrétní hodnoty), tato úloha ukazuje, že příkaz INSRET lze napsat také za pomocí dotazu SELECT. Takto můžeme velmi snadno vložit do tabulky velké množství záznamů najednou. Tato úloha také vyvrací častou mylnou představu studentů, že INSERT vždy vkládá jediný záznam.

6. Odstraňte z databáze jazyk Mandarínština (Mandarin). Úlohu řešte až po vyřešení příkladu 4.

```
DELETE FROM language
WHERE name = 'Mandarin';
```

Zde se dostáváme k poslednímu z příkazů DML, a tím je DELETE. Jeho syntaxe je podobná příkazu UPDATE – základem je opět správně nastavit podmínku `WHERE`. Také tady platí, že `WHERE` je možno vynechat, čímž dojde ke smazání obsahu tabulky (neplést se smazáním celé tabulky jako takové).

Pro shrnutí – k úpravě dat v databázi existují 3 standardní DML příkazy:

- `INSERT` – vkládá nové řádky do tabulky,
- `UPDATE` – aktualizuje existující záznamy, tj. mění hodnoty jejich atributů, a
- `DELETE` – maže řádky z tabulky.

Pro zajímavost, Microsoft SQL Server má například ještě navíc příkaz `MERGE`, který si ale zde ukazovat nebudeme.

7. Odstraňte z databáze film Terminátor (úlohu řešte až po vyřešení příkladu 1). Je možné tuto úlohu řešit pouze odebráním příslušného záznamu z tabulky `film`?

Samotný příkaz pro odstranění filmu „Terminátor“ vypadá velice jednoduše:

```
DELETE
FROM film
WHERE title = 'Terminátor';
```

Pokud jste však řešili úlohu 1, příkaz nepůjde spustit, resp. obdržíte chybové hlášení. Mělo by být zřejmé, že problém je v tom, že existují záznamy, které se na daný film odkazují prostřednictvím cizího klíče. Jde v tomto případě o záznamy v tabulkách `film_actor` a `film_category`. Nejprve je tedy nutné smazat příslušné záznamy v těchto dvou tabulkách pomocí následujících dvou příkazů:

```
DELETE
FROM film_actor
WHERE film_id = (SELECT film_id FROM film WHERE title = 'Terminátor');


DELETE
FROM film_category
WHERE film_id = (SELECT film_id FROM film WHERE title = 'Terminátor');
```

Teprve potom bude možné spustit DELETE nad tabulkou `film`. V úloze 20 si však ukážeme, že existuje tzv. *kaskádové mazání*, kdy budou podřízené záznamy mazány automaticky.

8. Odstraňte z databáze všechny neaktivní zákazníky.

   V této úloze řešíme podobný problém jako v úloze předchozí. Zákazníka nemůžeme smazat, dokud se na něj odkazují nějaké výpůjčky nebo platby. Výpůjčky nemůžeme smazat, dokud se na ně odkazují platby (platba se odkazuje vždy na zákazníka a většinou i na výpůjčku – viz model databáze na straně 4).

   Musíme tedy začít odstraněním plateb, které se na neaktivní zákazníky odkazují buď skrz výpůjčku nebo přímo (podmínka `OR` v následujícím příkazu):

```
DELETE
FROM payment
WHERE
  rental_id IN
  (
    SELECT rental.rental_id
    FROM customer JOIN rental ON customer.customer_id = rental.customer_id
    WHERE customer.active = 0
  )
  OR customer_id IN
  (
    SELECT customer_id
    FROM customer
    WHERE active = 0
  );
```

   Následně můžeme odstranit výpůjčky:

```
DELETE
FROM rental
WHERE customer_id IN (SELECT customer_id FROM customer WHERE active = 0);
```

   Teprve v tuto chvíli se na neaktivní zákazníky neodkazuje žádný záznam, takže provedeme DELETE nad tabulkou `customer`:

```
DELETE
FROM customer
WHERE active = 0;
```

9. Přidejte do tabulky `film` nepovinný celočíselný atribut `inventory_count`. Nastavte tento atribut pro všechny filmy tak, aby odpovídal počtu kopií daného filmu (počtu odpovídajících záznamů v tabulce `inventory`).

   Až do této chvíle jsme nijak neměnili strukturu databáze. Měnili jsme sice obsah tabulek, ale samotná struktura (tabulky, jejich sloupce, vazby a další) zůstávala stále stejná. Pro úpravu struktury slouží příkazy v kategorii DDL.

   ```
   ALTER TABLE film
   ADD inventory_count INT;
   ```

   Příkaz ALTER TABLE je typickým zástupcem DDL příkazů. V tomto případě pomocí klauzule `ADD` říkáme, že chceme přidat nový sloupec s určitým jménem a datovým typem – v tomto případě `INT`, který představuje celé číslo v určitém rozsahu. Výčet datových typů zde uvádět nebudeme – ten můžete najít např. v dokumentaci používaného SŘBD[4].

   Druhá část úlohy je zde zejména k procvičení příkazu UPDATE a také k připomenutí agregačních funkcí:

   ```
   UPDATE film
   SET inventory_count = (
     SELECT COUNT(*)
     FROM inventory
     WHERE inventory.film_id = film.film_id
   )
   ```

10. Upravte atribut `name` v tabulce `category` tak, aby bylo možné zadat až 50 znaků.

    ```
    ALTER TABLE category
    ALTER COLUMN name VARCHAR(50);
    ```

    Jde o poměrně běžnou úpravu, kterou je potřeba provést, když si zákazník stěžuje, že se mu do políčka nevejde potřebný text. Pro úpravu definice sloupce (datový typ, rozsah datového typu, povinnost apod.) slouží klauzule `ALTER COLUMN`. Zde poznamenejme, že např. v SŘBD Oracle se místo `ALTER COLUMN` píše `MODIFY`.

11. Přidejte do tabulky `customer` povinný textový atribut `phone` o maximální délce 20 znaků. Telefon nastavte podle atributu `phone`, který je součástí adresy zákazníka.

    ```
    ALTER TABLE customer
    ADD phone VARCHAR(20) NOT NULL;
    ```

    Povinný atribut specifikujeme tak, že za datový typ (v tomto případě řetězec s proměnnou délkou a s uvedením maximálního počtu znaků) uvedeme `NOT NULL`. Pokud se však příkaz pokusíme spustit, zjistíme, že SŘBD hlásí chybu. Problém je v tom, že atribut má

[4] https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql

být povinný ale zároveň není jasné, jak má být nastavena jeho hodnota pro záznamy, které se v tabulce už vyskytují (a není jich málo).

Existují dvě možná řešení: (1) nastavit atributu výchozí hodnotu, což si ukážeme později (viz úloha 12), nebo (2) vytvořit atribut jako nepovinný, nastavit příkazem UPDATE jeho hodnotu pro všechny záznamy a nakonec atribut změnit na povinný. Zkusme si tedy druhou z možností.

Nejprve přidejme nepovinný atribut `phone`. Nepovinný atribut specifikujeme tak, že za datový typ místo `NOT NULL` uvedeme `NULL` nebo neuvedeme nic:

```
ALTER TABLE customer
ADD phone VARCHAR(20);
```

Dále aktualizujme hodnotu tohoto atributu pro všechny záznamy:

```
UPDATE customer
SET phone = (
  SELECT phone
  FROM address
  WHERE address.address_id = customer.address_id
)
```

Nakonec můžeme atribut upravit tak, aby byl povinný:

```
ALTER TABLE customer
ALTER COLUMN phone VARCHAR(20) NOT NULL;
```

12. Přidejte do tabulky `rental` povinný atribut `create_date`, jehož výchozí hodnota bude aktuální časové razítko.

    Zde si tedy ukážeme druhou z možností, jak do neprázdné tabulky přidat povinný atribut. Přidáme atribut s definicí výchozí hodnoty.

    Výchozí hodnotu můžeme specifikovat jednoduše tak, že za datový typ a `NOT NULL` přidáme klíčové slovo `DEFAULT`. Pro získání aktuálního data a času použijeme funkci `CURRENT_TIMESTAMP`:

    ```
    ALTER TABLE rental
    ADD create_date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP;
    ```

    Uvedené řešení je sice v pořádku, ale z určitého důvodu, který si ukážeme hned v následující úloze, by se dalo ještě trochu vylepšit. Specifikace výchozí hodnoty je totiž jedním z tzv. integritních omezení (constraints) a každé integritní omezení by mělo mít nějaký název. Pokud název neuvedeme, SŘBD nějaký název sám zvolí, přičemž půjde o částečně náhodnou kombinaci znaků a čísel. Naučme se integritní omezení explicitně pojmenovávat, a to tak, že uvedeme klíčové slovo `CONSTRAINT` následované názvem (např. `DF_rental_create_date`):

    ```
    ALTER TABLE rental
    ADD create_date DATETIME NOT NULL
      CONSTRAINT DF_rental_create_date DEFAULT CURRENT_TIMESTAMP;
    ```

13. Atribut `rental.create_date` vytvořený v předchozí úloze smažte.

    Také v této úloze pracujeme s příkazem ALTER TABLE, u kterého tentokrát použijeme klauzuli `DROP COLUMN`. Následující příkaz je sice syntakticky správně, ale po spuštění obdržíme chybu.

```
ALTER TABLE rental
DROP COLUMN create_date;
```

Problém je v tom, že na sloupec je navázáno integritní omezení, které jsme pojmenovali jako `DF_rental_create_date`. Před odstraněním sloupce je nejprve nutné toto integritní omezení smazat pomocí následujícího příkazu:

```
ALTER TABLE rental
DROP CONSTRAINT DF_rental_create_date;
```

A právě zde vidíme, proč je vhodné integritní omezení pojmenovávat. V opačném případě bychom totiž museli nejprve zjistit, jaký název integritnímu omezení systém vygeneroval.

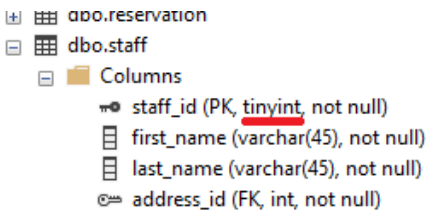
14. Přidejte do tabulky `film` nepovinný atribut `creator_staff_id`, který bude cizím klíčem do tabulky `staff`. Cizí klíč pojmenujte jako `fk_film_staff`.

    Tuto úlohu můžeme v principu vyřešit dvěma způsoby. Buď nejprve přidáme sloupec a pak z něj uděláme cizí klíč, nebo přidáme sloupec už s nastavením cizího klíče rovnou jedním příkazem.

    Vyzkoušejme si tedy první možnost a přidejme sloupec `creator_staff_id`:

    ```
    ALTER TABLE film
    ADD creator_staff_id TINYINT NULL;
    ```

    Zde stojí za povšimnutí, že jsme jako datový typ použili `TINYINT`, což je celé číslo s menším rozsahem než `INT`. Je tomu tak proto, že cizí klíč musí mít vždy naprosto stejný datový typ (včetně rozsahu, pokud bychom uvažovali např. `VARCHAR`) jako odpovídající primární klíč. Pohledem např. do seznamu sloupců tabulky `staff` v nástroji Microsoft SQL Server Management Studio (panel Object Explorer) se můžeme přesvědčit, že primární klíč `staff_id` je skutečně datového typu `TINYINT` (viz Obrázek 7).

    Obrázek 7: Zjištění datového typu primárního klíče `staff_id` v tabulce `staff`

    Nyní z nového sloupce dalším příkazem vytvoříme cizí klíč:

    ```
    ALTER TABLE film
    ADD FOREIGN KEY (creator_staff_id) REFERENCES staff (staff_id);
    ```

    Použijeme tedy opět ALTER TABLE, jelikož upravujeme strukturu tabulky. Za klauzulí `ADD FOREIGN KEY` pak v závorce uvedeme, které atributy tvoří cizí klíč, a za `REFERENCES`, na jakou tabulku a jaké atributy se tento cizí klíč odkazuje. Počet atributů v obou závorkách musí být stejný. Jestliže se zde odkazujeme na jednoduchý primární klíč (tj. složený z jednoho atributu), je cizí klíč také jednoduchý. Jsou ale situace, kdy se musíme odkázat na složený primární klíč – pak i cizí klíč bude složený.

Cizí klíč je další z integritních omezení (constraints), které bychom měli pojmenovávat. To znamená, že bychom si raději měli zapamatovat následující zápis, kde cizímu klíči nastavíme také název (`FK_film_staff`):

```
ALTER TABLE film
ADD CONSTRAINT FK_film_staff FOREIGN KEY (creator_staff_id) REFERENCES
    staff (staff_id);
```

Jak jme již dříve uvedli, sloupec můžeme přidat také rovnou už s tím, že z něj uděláme cizí klíč:

```
ALTER TABLE film
ADD creator_staff_id TINYINT NULL CONSTRAINT FK_film_staff FOREIGN KEY
    REFERENCES staff (staff_id);
```

Nakonec si ukažme pravděpodobně nejúspornější řešení celé úlohy:

```
ALTER TABLE film
ADD creator_staff_id TINYINT NULL REFERENCES staff;
```

15. Nastavte kontrolu atributu `staff.email` tak, aby hodnota e-mailu vždy obsahovala znak „@“ následovaný znakem „.“.

```
ALTER TABLE staff
ADD CONSTRAINT check_email CHECK (email LIKE '%@%.%');
```

Dostáváme se k dalšímu z integritních omezení – `CHECK`. Tímto integritním omezením můžeme specifikovat logickou podmínku, která musí platit pro každý záznam v určité tabulce. Pokud bychom se pokusili přidat nebo upravit libovolný záznam tak, že porušíme danou podmínku (např. e-mail nebude obsahovat zavináč), skončí daný příkaz chybou. Stejně tak by mělo být jasné, že nelze vytvořit integritní omezení `CHECK` nad tabulkou s neprázdným obsahem, kde některé záznamy specifikovanou podmínku nesplňují.

16. Kontrolu nastavenou v předchozí úloze zrušte.

```
ALTER TABLE staff
DROP CONSTRAINT check_email;
```

Pokud jsme si integritní omezení vhodně pojmenovali, nebude s jeho odstraněním žádný problém. Postupujeme stejně, jako bychom mazali cizí klíč nebo výchozí hodnotu.

17. Nastavte kontrolu výpůjček tak, aby datum vrácení byl vždy větší než je datum výpůjčky.

```
ALTER TABLE rental
ADD CONSTRAINT check_dates CHECK (return_date > rental_date)
```

Řešení úlohy je obdobné jako v úloze 15. Pouze si zde ukazujeme, že podmínka může pracovat zároveň s více atributy z dané tabulky. Mohlo by nás napadnout, zda podmínka může obsahovat poddotaz. V tomto případě bohužel nemůže. Pro složitější kontroly lze využít databázové triggery, kterými se však v předmětu DS1 zabývat nebudeme.

18. Vytvořte novou tabulku `reservation`, tj. tabulku rezervací, s celočíselným automaticky generovaným primárním klíčem `reservation_id`. Tabulka bude dále obsahovat následující atributy: povinné datum rezervace `reservation_date` s výchozí hodnotou na aktuální datum, povinné datum konce rezervace `end_date`, povinné ID zákazníka `customer_id` jako cizí klíč do tabulky `customer`, povinné ID filmu `film_id` jako cizí klíč do tabulky `film` a nepovinné ID zaměstnance `staff_id` jako cizí klíč do tabulky `staff`.

```
CREATE TABLE reservation
(
  reservatoin_id TINYINT IDENTITY PRIMARY KEY NOT NULL,
  reservation_date DATE NOT NULL,
  end_date DATE NOT NULL,
  customer_id INT CONSTRAINT fk_reservation_customer FOREIGN KEY
    REFERENCES customer (customer_id),
  film_id INT CONSTRAINT fk_reservation_film FOREIGN KEY
    REFERENCES film (film_id),
  staff_id TINYINT CONSTRAINT fk_reservation_staff FOREIGN KEY
    REFERENCES staff (staff_id)
);
```

U této úlohy se konečně dostáváme k velmi důležitému příkazu CREATE TABLE, kterým vytvoříme úplně novou tabulku. V závorce za `CREATE TABLE` uvádíme seznam sloupců včetně jejich datového typu a případně dalších integritních omezení. Syntaxe u jednotlivých sloupců je obdobná, jako bychom sloupce přidávali příkazem `ALTER TABLE ... ADD`. Jedinou novinkou je zde specifikace primárního klíče pomocí `PRIMARY KEY`, kterému předchází klíčové slovo `IDENTITY`. Tím říkáme, že primární klíč bude generovaný automaticky.

Upozorňujeme, že tento způsob definice automaticky generovaného klíče je specifický pro Mirosoft SQL Server. V jiných systémech se používají například jiná klíčová slova (`AUTO_INCREMENT` pro MySQL nebo `COUNTER` pro Micosoft Access) nebo se používají tzv. sekvence (Oracle, PostgreSQL nebo nově i Microsoft SQL Server).

19. Vložte do tabulky vytvořené v předchozí úloze dva libovolné záznamy. Poté druhý ze záznamů smažte. Jaké ID bude přiděleno dalšímu vkládanému záznamu?

Vložme si tedy podle zadání nejprve dva libovolné záznamy:

```
INSERT INTO reservation (reservation_date, end_date, customer_id, film_id,
    staff_id)
VALUES ('2020-10-02', '2020-10-05', 25, 10, 1);

INSERT INTO reservation (reservation_date, end_date, customer_id, film_id,
    staff_id)
VALUES ('2020-10-02', '2020-10-15', 56, 78, 2);
```

Musíme se samozřejmě pouze ujistit, že použité konstanty cizích klíčů skutečně odkazují na existující záznamy, jinak příkazy skončí chybou.

Nyní se pro jistotu podívejme, s jakými ID se záznamy vložily:

```
SELECT *
FROM reservation
```

Dejme tomu, že ID druhého záznamu bude `$x` (s největší pravděpodobností `$x = 2`). Zkusme tedy tento záznam smazat:

```
DELETE FROM reservation
WHERE reservatoin_id = $x
```

Nakonec vložíme další nový záznam a zjistíme, jaké ID mu bylo přiřazeno:

```
INSERT INTO reservation (reservation_date, end_date, customer_id, film_id,
    staff_id)
VALUES ('2020-10-20', '2020-10-21', 5, 64, 2);

SELECT *
FROM reservation
```

Někoho možná překvapí, že ID posledního vloženého záznamu není `$x`, ale číslo o 1 větší. Je tomu tak proto, že automatický generátor nikdy nepřidělí novému záznamu ID, které již bylo někdy v minulosti použito. Dokážete uhodnout, jaký význam toto chování má?

20. (a) Vytvořte tabulku `review` s atributy `film_id` a `customer_id`, které budou představovat cizí klíče do tabulek `film`, resp. `customer`. Oba tyto atributy budou dohromady tvořit složený primární klíč. Součástí tabulky bude dále povinný atribut `stars`, který bude nabývat celých čísel v intervalu $\langle 1, 5 \rangle$, a nepovinný atribut `actor_id`, který bude cizím klíčem do tabulky `actor`. Zajistěte, aby se při smazání zákazníka nebo filmu automaticky smazaly také odpovídající záznamy v tabulce `review`. Dále zajistěte, aby při smazání herce došlo u odpovídajících záznamů v `review` k nastavení `actor_id` na `NULL`.

```
CREATE TABLE review
(
  film_id INT NOT NULL
    CONSTRAINT fk_review_film
    FOREIGN KEY REFERENCES film (film_id) ON DELETE CASCADE,
  customer_id INT NOT NULL
    CONSTRAINT fk_review_customer
    FOREIGN KEY REFERENCES customer (customer_id) ON DELETE CASCADE,
  stars TINYINT NOT NULL
    CONSTRAINT ch_review_stars
    CHECK (stars BETWEEN 1 AND 5),
  actor_id INT NULL
    CONSTRAINT fk_review_actor
    FOREIGN KEY REFERENCES actor (actor_id) ON DELETE SET NULL,
  PRIMARY KEY (film_id, customer_id)
)
```

V této úloze si znova ukážeme vytvoření nové tabulky. Oproti úloze 18 je zde rozdíl v tom, že primární klíč není složen z jednoho, ale z více atributů – konkrétně tedy z atributů `film_id` a `customer_id`. V takovém případě nemůžeme klíčové slovo `PRIMARY KEY` uvést rovnou za atribut (popř. někoho by mohlo napadnout uvést `PRIMARY KEY` za oba atributy – to je syntakticky špatně), ale musíme jej uvést samostatně pod seznam sloupců.

Dále se po nás v úloze chce specifické chování jednotlivých vazeb. Součástí vazby totiž může být jeden z následujících tří modifikátorů, které určují, co se má stát v případě, že dojde k odstranění záznamu z odkazované tabulky:

i. `ON DELETE NO ACTION` je výchozí volba, tj. pokud je mazaný záznam odkazovaný jiným záznamem, mazání nebude povoleno (viz např. úloha 7).

ii. `ON DELETE CASCADE` říká, že záznamy, které se na smazaný záznam odkazují, budou automaticky smazány také – tzv. kaskádové mazání.

iii. `ON DELETE SET NULL` říká, že cizí klíč odkazující se na mazaný záznam bude nastaven na `NULL`. Tato volba má samozřejmě smysl jen v případě, že daný cizí klíč není povinným atributem.

V našem případě jsme tedy pro cizí klíče `film_id` a `customer_id` modifikátorem `ON DELETE CASCADE` nastavili, že pokud bude odstraněn nějaký film nebo zákazník, pak všechna hodnocení (záznamy v tabulce `review`), které se na daný film nebo zákazníka odkazují, budou automaticky smazána také. Modifikátorem `ON DELETE SET NULL` jsme určili, že při smazání herce bude všem odpovídajícím hodnocením nastavena hodnota `actor_id` na `NULL`.

Ukažme si nakonec obecnější a univerzálnější zápis příkazu CREATE TABLE. Ten vypadá tak, že nejprve specifikujeme jednotlivé sloupce a teprve potom samostatně uvedeme jednotlivá integritní omezení. Následující obecnější zápis je tedy ekvivalentní předchozímu zápisu:

```
CREATE TABLE review
(
  film_id INT NOT NULL,
  customer_id INT NOT NULL,
  stars TINYINT NOT NULL,
  actor_id INT NULL,
  PRIMARY KEY (film_id, customer_id),
  CONSTRAINT fk_review_film
    FOREIGN KEY (film_id) REFERENCES film (film_id) ON DELETE CASCADE
      ,
  CONSTRAINT fk_review_customer
    FOREIGN KEY (customer_id) REFERENCES customer (customer_id) ON
      DELETE CASCADE,
  CONSTRAINT fk_review_actor
    FOREIGN KEY (actor_id) REFERENCES actor (actor_id) ON DELETE SET
      NULL,
  CONSTRAINT ch_review_stars CHECK (stars BETWEEN 1 AND 5)
)
```

(b) Vložte do tabulky `review` dva záznamy:

- Hodnocení filmu ARMY FLINTSTONES zákazníka BRIAN WYMAN – 4 hvězdy, bez uvedení herce.
- Hodnocení filmu ARSENIC INDEPENDENCE zákazníka CHERYL MURPHY – 5 hvězd s uvedením herce EMILY DEE.

Pojďme nejprve vyřešit tu „otravnější“ část, tj. zjistit si ID pro uvedené filmy, herce a zákazníky. Konkrétní dotazy zde pro jejich jednoduchost psát nebudeme, konstanty by měly být následující:

film	ARMY FLINTSTONES	film_id	= 40
film	ARSENIC INDEPENDENCE	film_id	= 41
zákazník	BRIAN WYMAN	customer_id	= 318
zákazník	CHERYL MURPHY	customer_id	= 59
herec	EMILY DEE	actor_id	= 148

Příkazy INSERT by pak měly vypadat takto:

```
INSERT INTO review (film_id, customer_id, stars, actor_id)
VALUES (40, 318, 4, NULL);

INSERT INTO review (film_id, customer_id, stars, actor_id)
VALUES (41, 59, 5, 148);
```

Pozn.: Zákazník BRIAN WYMAN a herec EMILY DEE jsou v této úloze zvoleni záměrně, jelikož se na ně neodkazují žádné jiné záznamy (kromě našich záznamů v `review`).

(c) Odstraňte z databáze zákazníka BRIAN WYMAN a herce EMILY DEE. Následně si prohlédněte obsah tabulky `review`.

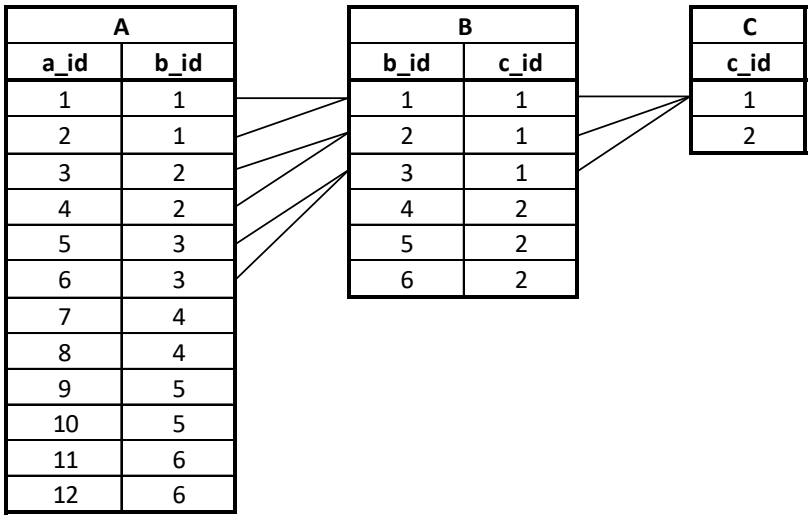
```
DELETE FROM customer
WHERE customer_id = 318

DELETE FROM actor
WHERE actor_id = 148;
```

Samotné příkazy pro smazání příslušných záznamů nikoho nepřekvapí. Nicméně měli bychom si uvědomit rozdíl oproti úloze 7, kde smazání nebylo možné, dokud se na záznam odkazovaly jiné záznamy. Tím, že jsme však v tomto případě nastavili vazbám modifikátory `ON DELETE CASCADE` (mezi hodnocením a zákazníkem) a `ON DELETE SET NULL` (mezi hodnocením a hercem), se spuštěním uvedených příkazů není problém. Hodnocení, které provedl zákazník s ID 318 (BRIAN WYMAN) bude automaticky smazáno, a u hodnocení herce s ID 148 (EMILY DEE) dojde k nastavení atributu `actor_id` na `NULL`. Přesvědčte se o tom.

Zkusme si kaskádové mazání pro jistotu demonstrovat ještě na jednom příkladě a vysvětlit si, odkud se vlastně vzal název „kaskádové“. Mějme tabulky `A(a_id, b_id)`, `B(b_id, c_id)` a `C(c_id)`, kde `A.a_id`, `B.b_id` a `C.c_id` jsou primární klíče v jednotlivých tabulkách a `A.b_id`, `B.c_id` cizí klíče. Pro cizí klíče `A.b_id` a `B.c_id` bude nastaveno kaskádové mazání (tj. `ON DELETE CASCADE`). Ukázkový obsah tabulek je znázorněn na Obrázku 8. Pokud z tabulky `C` odstraníme první záznam (tj. $\texttt{c\_id} = 1$), automaticky odstraníme také související záznamy z tabulky `B` ($\texttt{b\_id} \in \{1, 2, 3\}$) a dále související záznamy z tabulky `A` ($\texttt{a\_id} \in \{1, 2, 3, 4, 5, 6\}$). Vidíme, že záznamy se budou mazat „kaskádově“.

Z příkladu na Obrázku 8 plyne jedno upozornění. Kaskádové mazání je „dobrý sluha, ale zlý pán“. Můžeme si sice usnadnit práci tím, že nebudeme muset myslet na ruční mazání souvisejících záznamů. Na druhou stranu, jak na obrázku vidíme, neopatrným nastavením kaskádového mazání můžeme velmi snadno nechtěně smazat obsah velké části databáze.



A	
a_id	b_id
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4
9	5
10	5
11	6
12	6

B	
b_id	c_id
1	1
2	1
3	1
4	2
5	2
6	2

C
c_id
1
2

Obrázek 8: Ukázka kaskádového mazání

21. Zálohujte obsah tabulky `film` do nové tabulky `film_backup`. Nová tabulka bude svou strukturou totožná s tabulkou `film` s tím rozdílem, že nebude obsahovat nastavení primárních ani cizích klíčů. Jinými slovy, atributy jako `film_id`, `language_id` budou běžné celočíselné (neklíčové) atributy.

   U této úlohy si ukážeme dvě řešení. Jedno zdlouhavé, které byste už teď měli umět dát dohromady, a jedno překvapivě velice jednoduché. Začněme tedy tím prvním tak, že příkazem CREATE TABLE vytvoříme tabulku `film_backup` se stejnou strukturou jako má tabulka `film`:

```
CREATE TABLE film_backup
(
  film_id INT,
  title VARCHAR(255),
  description TEXT,
  release_year VARCHAR(4),
  language_id TINYINT,
  original_language_id TINYINT,
  rental_duration TINYINT,
  rental_rate DECIMAL(4, 2),
  length SMALLINT,
  replacement_cost DECIMAL(5, 2),
  rating VARCHAR(10),
  special_features VARCHAR(255),
  last_update DATETIME
);
```

   Poté pomocí příkazu INSERT INTO, kde místo `VALUES` použijeme `SELECT`, zkopírujme všechny filmy do nové tabulky:

```
INSERT INTO film_backup (
  film_id, title, description, release_year, language_id,
  original_language_id, rental_duration, rental_rate,
  length, replacement_cost, rating, special_features, last_update)
SELECT
  film_id, title, description, release_year, language_id,
```

```
  original_language_id, rental_duration, rental_rate,
  length, replacement_cost, rating, special_features, last_update
FROM film
```

A nyní druhé, úspornější, řešení:

```
SELECT * INTO film_backup
FROM film
```

Tímto triviálním a velmi užitečným příkazem můžeme z výsledku libovolného dotazu rovnou vytvořit novou tabulku. Bohužel, syntaxe příkazu je specifická pro Microsoft SQL Server, nicméně v jiných databázích existují jiné obdobné konstrukce.

22. Odstraňte tabulky `review` a `film_backup` vytvořené v předchozích dvou úlohách.

    Odstranění kompletně celé tabulky z databáze provádíme příkazem DROP TABLE. Odstranění tabulek `film_backup` a `review` bude tedy vypadat následovně:

    ```
    DROP TABLE film_backup;
    DROP TABLE review;
    ```

    Můžeme si všimnout, že pro smazání tabulky není nutné nejprve mazat obsažená integritní omezení (např. výchozí hodnoty, cizí klíče apod.). Na druhou stranu, tabulku se nám nepovede smazat, pokud je odkazována cizím klíčem z jiné tabulky, jak si ukážeme v poslední úloze 24.

    Pro shrnutí, v tuto chvíli už tedy umíme všechny DDL operace s tabulkou:

    - `CREATE TABLE tabulka` – vytvoří novou tabulku,
    - `ALTER TABLE tabulka` – upraví strukturu tabulky, kde
      - `ADD sloupec` – přidá sloupec,
      - `ALTER COLUMN sloupec` – upraví sloupec,
      - `DROP COLUMN sloupec` – odstraní sloupec,
      - `ADD CONSTRAINT` – přidá integritní omezení (`DEFAULT`, `CHECK`, `FOREIGN KEY`, popř. `PRIMARY KEY`),
      - `DROP CONSTRAINT` – odstraní integritní omezení,
    - `DROP TABLE tabulka` – odstraní tabulku.

    Měli bychom si uvědomovat rozdíly mezi DML příkazy `INSERT`, `UPDATE`, `DELETE` a DDL příkazy `CREATE`, `ALTER`, `DROP`. Zatímco první skupina příkazů manipuluje s obsahem tabulky, druhá skupina zasahuje do struktury tabulky.

23. Vytvořte tabulku `rating` s atributy `rating_id` – celočíselný automaticky generovaný primární klíč, `name` – povinný řetězec o maximálně 10-ti znacích a `description` – nepovinný řetězec s neomezeným počtem znaků. Vložte do nové tabulky záznamy pro unikátní hodnoty atributu `rating` v tabulce `film`. Vytvořte v tabulce `film` povinný atribut `rating_id`, který bude cizím klíčem do nově vytvořené tabulky `rating`. Hodnoty tohoto atributu budou nastaveny tak, aby odpovídaly skutečnosti dle atributu `rating`. Původní atribut `rating` nakonec smažte.

    Tato úloha demonstruje řešení poměrně častého problému. Atribut `rating` v tabulce `film` obsahuje několik málo unikátních hodnot – filmových kategorií. Abychom ke každé

kategorii mohli přidat popis, rozhodli jsme se zavést do databáze samostatný číselník kategorií a místo původního atributu `film.rating` evidovat cizí klíč `film.rating_id` odkazující se do tohoto číselníku.

Začněme tedy tak, že vytvořme tabulku představující nový číselník:

```
CREATE TABLE rating
(
  rating_id TINYINT NOT NULL IDENTITY PRIMARY KEY,
  name VARCHAR(10) NOT NULL,
  description TEXT NULL
);
```

Nyní pomocí příkazu `INSERT` s klauzulí `SELECT` do číselníku jedním příkazem hromadně vložíme všechny kategorie. Za `SELECT` nesmíme zapomenout uvést `DISTINCT`, jinak by nová tabulka `rating` obsahovala duplicitní kategorie.

```
INSERT INTO rating (name)
SELECT DISTINCT rating
FROM film;
```

Dále do tabulky `film` přidáme cizí klíč odkazující se novou tabulku `rating`. Cizí klíč bude prozatím nepovinný, naplníme jej posléze.

```
ALTER TABLE film
ADD rating_id TINYINT NULL CONSTRAINT fk_film_rating FOREIGN KEY
   REFERENCES rating (rating_id);
```

Nyní tedy nastavíme hodnotu cizího klíče tak, že pro každý film vyhledáme odpovídající kategorii – tj. takovou, jejíž `rating.name = film.rating`:

```
UPDATE film
SET rating_id = (
  SELECT rating_id
  FROM rating
  WHERE rating.name = film.rating
);
```

V tuto chvíli je tedy cizí klíč nastaven pro všechny filmy a atribut `rating_id` můžeme změnit na povinný:

```
ALTER TABLE film
ALTER COLUMN rating_id TINYINT NOT NULL;
```

Nakonec odstraníme původní atribut `film.rating`:

```
ALTER TABLE film
DROP COLUMN rating;
```

Spuštění tohoto příkazu však pravděpodobně povede k chybě a to proto, že na tento atribut jsou navázána dvě integritní omezení. Jak zjistíme, která integritní omezení to jsou? Pravděpodobně nejrychlejší způsob je přečíst si chybové hlášení, které by mělo vypadat jako na Obrázku 9.

Zkusme se na chybová hlášení nedívat jako na sprosté nadávky, ale jako na užitečné rady. Jde o integritní omezení `DF__film__rating__59063A47` (výchozí hodnota) a `CHECK_special_rating` (kontrola přípustných hodnot). Integritní omezení tedy smažeme následujícími příkazy:

```
Msg 5074, Level 16, State 1, Line 1
The object 'DF__film__rating__59063A47' is dependent on column 'rating'.
Msg 5074, Level 16, State 1, Line 1
The object 'CHECK_special_rating' is dependent on column 'rating'.
Msg 4922, Level 16, State 9, Line 1
ALTER TABLE DROP COLUMN rating failed because one or more objects access this column.
```

Obrázek 9: Chybové hlášení při pokusu o smazání sloupce `film.rating`

```
ALTER TABLE film
DROP CONSTRAINT DF__film__rating__59063A47;

ALTER TABLE film
DROP CONSTRAINT CHECK_special_rating;
```

Nyní se můžeme vrátit k odstranění sloupce `film.rating` a úloha je tímto hotová.

24. Odstraňte z databáze všechny tabulky.

    V této úplně poslední úloze se naučíme po sobě uklidit, procvičíme si příkazy `DROP TABLE` a `DROP CONSTRAINT` a zmíníme se o tzv. systémovém katalogu. Úlohu můžeme vyřešit obecně dvěma způsoby: (1) odstranit tabulky v takovém pořadí, abychom nikdy nemazali tabulku, na kterou se odkazuje cizí klíč nebo (2) odstranit z databáze nejprve všechny cizí klíče a následně všechny tabulky.

    Začneme první možností a postupujme podle E-R diagramu databáze na Obrázku 1 (strana 5). Tabulky, které jistě nebudou odkazovány cizími klíči jsou např. `film_actor` a `film_category`. Začneme tedy:

    ```
    DROP TABLE film_actor;
    DROP TABLE film_category;
    ```

    Následně můžeme odstranit herce a kategorie, protože ty už v tuto chvíli nejsou odkazovány žádným cizím klíčem:

    ```
    DROP TABLE actor;
    DROP TABLE category;
    ```

    Kompletní řešení zde ukazovat nebudeme – za pomocí E-R diagramu jistě každý rozumí, jakým způsobem postupovat dál. Každopádně se po čase dostaneme do situace, kdy pouhá volba správného pořadí nebude stačit. Jde např. o tabulky `store` a `staff`, kdy se jedna odkazuje na druhou a naopak (`store.manager_staff_id` a `staff.store_id`). U podobných cyklů nezbývá nic jiného než nejprve odebrat cizí klíče. V tomto případě jde tedy o použití následujících příkazů:

    ```
    ALTER TABLE staff
    DROP CONSTRAINT fk_staff_store;


    ALTER TABLE store
    DROP CONSTRAINT fk_store_staff;
    ```

    Pro ujasnění dodáváme, že odebrání cizího klíče neznamená smazání atributu. Pouze odebereme určitou speciální vlastnost atributu – tj. atribut bude dál existovat, ale nebude cizím klíčem (jeho hodnota nebude nijak kontrolována ani omezována).

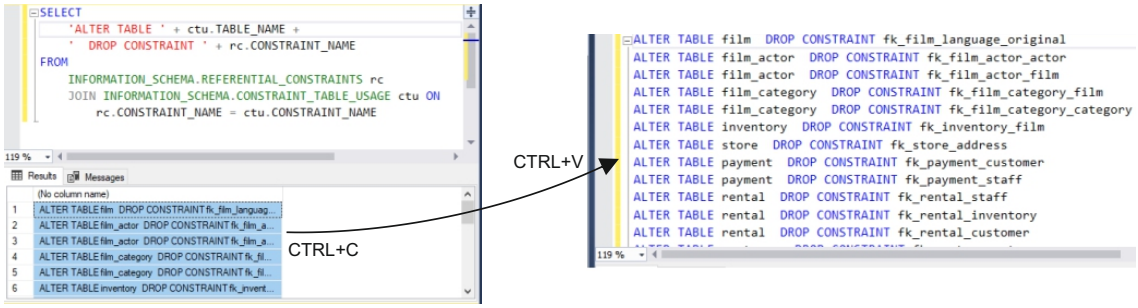
Vraťme se nyní ke druhé možnosti – tzn. odebrat nejprve všechny cizí klíče a pak všechny tabulky. Tady samozřejmě nastává praktický problém – kde zjistíme seznam všech cizích klíčů? Pokud nechceme ručně „proklikávat“ strom s tabulkami (Object Explorer v Microsoft SQL Server Management Studio), můžeme si pomoct tzv. systémovým katalogem. Možnost použít tento katalog zde uvádíme spíše pro zajímavost – nejde o látku, která by byla součástí předmětu DS1. Systémový katalog je kolekce jakýchsi fiktivních tabulek, které obsahují metadata o samotných tabulkách.

Následující dotaz nad tabulkami `INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS` a `INFORMATION_SCHEMA.CONSTRAINT_TABLE_USAGE`, které jsou součástí systémového katalogu, vrátí názvy tabulek a názvy cizích klíčů. Pozor, katalog vypadá v každém SŘBD trochu jinak – řešení tedy bude specifické pro Microsoft SQL Server:

```
SELECT ctu.TABLE_NAME, rc.CONSTRAINT_NAME
FROM
  INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS rc
  JOIN INFORMATION_SCHEMA.CONSTRAINT_TABLE_USAGE ctu ON
    rc.CONSTRAINT_NAME = ctu.CONSTRAINT_NAME
```

Vhodnou úpravou klauzule `SELECT` si můžeme nechat rovnou vypsat samotné příkazy `ALTER TABLE ... DROP CONSTRAINT`, které pak můžeme jednoduše zkopírovat z výsledku, vložit jako skript a spustit (viz Obrázek 10).

```
SELECT 'ALTER␣TABLE␣' + ctu.TABLE_NAME + '␣␣DROP␣CONSTRAINT␣' + rc.
   CONSTRAINT_NAME
FROM
  INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS rc
  JOIN INFORMATION_SCHEMA.CONSTRAINT_TABLE_USAGE ctu ON
    rc.CONSTRAINT_NAME = ctu.CONSTRAINT_NAME
```



Obrázek 10: Ukázka využití systémového katalogu

Nakonec si podobně můžeme nechat vypsat příkazy pro smazání všech tabulek, které teď již nejsou odkazovány žádnými cizími klíči. Využijeme další z tabulek systémového katalogu – `INFORMATION_SCHEMA.TABLES`:

```
SELECT 'DROP␣TABLE␣' + TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE␣TABLE'
```