Kapitola 1: Úvod

- Účel databázových systémů
- Pohled na data
- Modely dat
- Jazyk definice dat (Data Definition Language; DDL)
- Jazyk manipulace s daty (Data Manipulation Language; DML)
- Správa transakcí
- Správa ukládání dat
- Správce databáze
- Uživatelé databáze
- Přehled struktury systému

Systém pro správu databází (Database Management System; DBMS)

- Soubor s daty ve vzájemném vztahu
- Sada programů pro přístup k datům
- DBMS obsahuje informace o jednotlivých záznamech (?)
- DBMS poskytuje prostředí, které je pohodlné i účinné

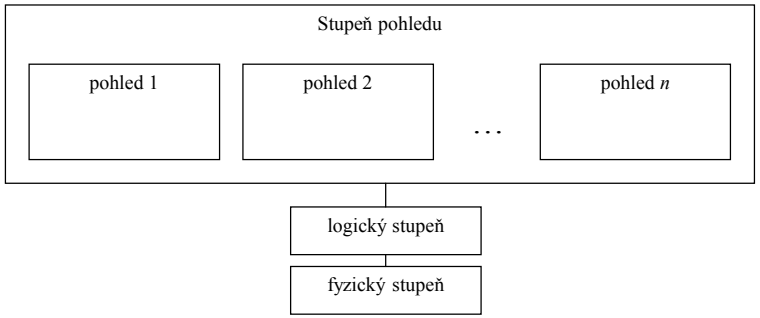
Účel databázových systémů

Systémy pro správu databází byly vyvinuty kvůli zvládnutí následujících problémů při zpracování souborů dat.

- Redundance a inkonsistence dat
- Problémy s přístupem k datům
- Izolace dat – více souborů a formátů
- Problémy s integritou
- Jedinečnost (atomicita) aktualizací
- Současný přístup více uživatelů
- Bezpečnostní problémy

Pohled na data

Architektura databázového systému:



Stupně abstrakce

- Fyzický stupeň: popisuje, jak je záznam (např. *customer*) uložen
- Logický stupeň: popisuje data uložená v databázi a vztahy mezi nimi.

```
type customer = record
        name: string;
        street: string;
        city: integer;
end;
```

- Stupeň pohledu: aplikační programy skrývají detaily typů dat. Pohledy také mohou skrývat informace (jako např. plat) kvůli bezpečnostním důvodům.

Instance a schéma

- Podobné typům a proměnným v programovacích jazycích
- Schéma – logická struktura databáze (např. množina zákazníků a účtů a vztahy mezi nimi)
- Instance – aktuální obsah databáze v určitém čase

Nezávislost dat

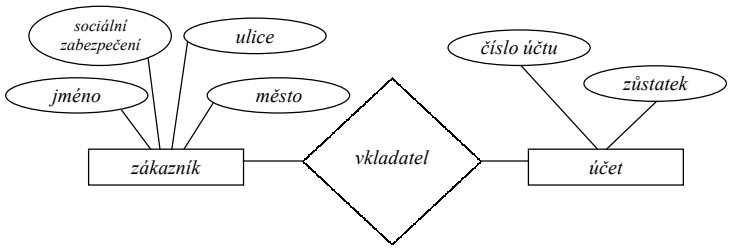
- Schopnost modifikovat definici schématu v jednom pohledu bez vlivu na definici schématu v dalším vyšším pohledu.
- Rozhraní mezi různými stupni a komponentami by měla být dobře definována, aby změny v některých částech neměly významný vliv na jiné části.
- Dva stupně nezávislosti dat:
  - Fyzická nezávislost dat
  - Logická nezávislost dat

Modely dat

- Sada nástrojů pro popis:
  - dat
  - vztahů mezi daty
  - sémantik dat
  - mezí dat (data constraints)
- Objektové logické modely
  - Model vztahu mezi entitami (entity-relationship model)
  - objektově orientovaný model
  - sémantický model
  - funkcionální model
- Záznamové logické modely
  - relační model (např. SQL/DS, DB2)
  - síťový model
  - hierarchický model (např. IMS)

Model vztahu mezi entitami (Entity-relationship model)

Příklad tohoto modelu



Relační model

Příklad dat v tabulce v relačním modelu:

jméno zákazníka	*sociální zabezpečení*	*ulice*	*město*	*číslo účtu*
Johnson	192-83-7465	Alma	Palo Alto	A-101
Smith	019-28-3746	North	Rye	A-215
Johnson	192-83-7465	Alma	Palo Alto	A-201
Jones	321-12-3123	Main	Harrison	A-217
Smith	019-28-3746	North	Rye	A-201

číslo účtu	*zůstatek*
A-101	500
A-201	900
A-215	700
A-217	750

Jazyk definice dat (Data definition language; DDL)

- Specifikace značení pro definování schématu databáze
- DDL kompilátor generuje množinu tabulek uložených v datovém slovníku (*Data dictionary*)
- Slovník dat obsahuje metadata (data o datech)
- Jazyk uložení a definice dat (*Data storage and definition language*) – speciální typ DDL, ve kterém je specifikována struktura uložení dat a metody použité databázovým systémem k přístupu k datům

Jazyk manipulace s daty (Data manipulation language; DML)

- Jazyk pro zpřístupnění a manipulaci s daty organizovanými příslušným datovým modelem
- Dvě třídy jazyků
  - Procedurální – uživatel specifikuje, která data jsou vyžadována a jak je získat
  - Neprocedurální – uživatel specifikuje, která data jsou vyžadována, ale ne, jak je získat

Správa transakcí

- *Transakce* je sada operací, které představují jednu logickou funkci v databázové aplikaci
- Správa transakcí zajišťuje, že databáze zůstává v konzistentním stavu nezávisle na selhání systému (např. výpadky energie a pády OS) a selhání transakcí.
- Správa souběžnosti (Concurrency-control manager) ovládá interakci mezi současně probíhajícími transakcemi, aby zajistila konzistenci databáze.

Správa ukládání dat

- Správce ukládání je programový modul, který poskytuje rozhraní mezi daty uloženými v databázi na nízké úrovni a aplikačními programy a dotazy poslanými systému.
- Správce ukládání je zodpovědný za následující úlohy:
  - interakce se správcem souborů
  - výkonné ukládání, získávání a aktualizace dat

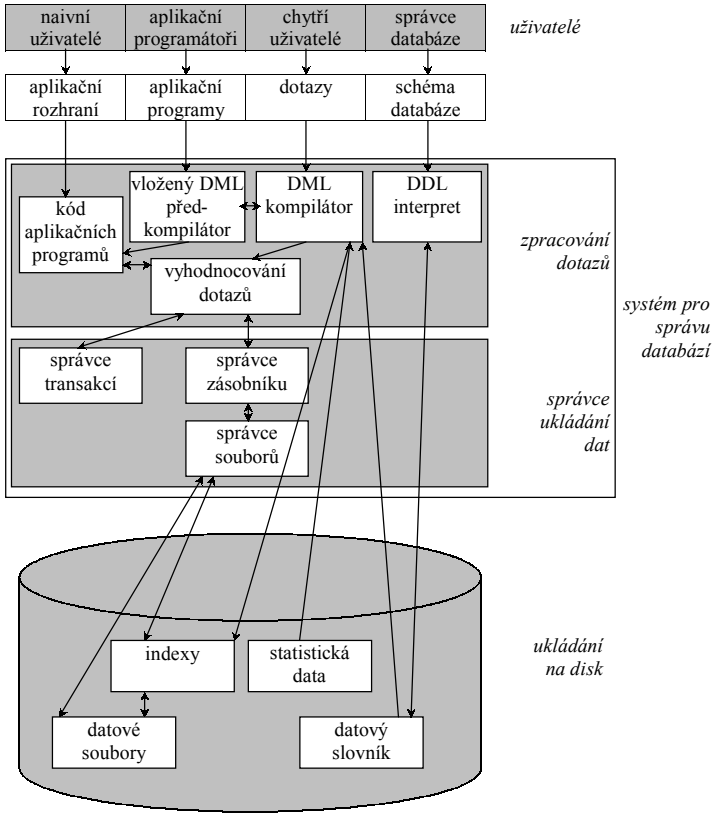
Správce databáze

- Koordinuje všechny aktivity v databázovém systému; má dobré znalosti o informačních prostředcích a potřebách podniku.
- Povinnosti správce databáze zahrnují:
  - Definice schématu
  - Definice struktury ukládání a metody přístupu
  - Modifikace schématu a fyzické organizace
  - Přiděluje uživatelům práva přístupu k databázi
  - Specifikuje meze integrity
  - Styčný bod s uživateli
  - Monitoruje výkon a zodpovídá za změny ve vybavení

Uživatelé databáze

- Uživatelé jsou rozděleni podle toho, do jaké míry pracují se systémem
- Aplikační programátoři – pracují se systémem pomocí volání DML
- Chytří uživatelé – formulují dotazy v databázovém dotazovacím jazyku
- Speciální uživatelé – píší speciální databázové aplikace, které nespadají do klasického zpracování dat
- Naivní uživatelé – spouští jeden z předem napsaných aplikačních programů

Přehled struktury systému



Kapitola 2: Model vztahu mezi entitami (Entity-Relationship model)

- Množiny entit
- Množiny vztahů
- Otázky designu (Design issues)
- Plánování mezí
- Klíče
- E-R diagram
- Rozšířené E-R rysy
- Design E-R databázového schématu
- Redukce E-R schématu na tabulky

Množiny entit

- *Databáze* může být modelována jako:
  - množina entit
  - vztahy mezi entitami
- *Entita* je objekt, který existuje a je odlišitelný od ostatních objektů.
  Např.: nějaká osoba, společnost, událost, rostlina
- *Množina entit* je skupina entit stejného typu, které sdílejí stejné vlastnosti.
  Např.: skupina všech osob, společností, stromů

Atributy

- Entita je reprezentována množinou atributů, to jsou popisné vlastnosti všech členů množiny entit.
  Např.:
  zákazník = (*jméno, sociální zabezpečení, ulice, město*)
  účet = (*číslo účtu, zůstatek*)
  tj. *entita* = (*atributy, …*)
- Doména – množina povolených hodnot pro každý atribut
- Typy atributů:
  - *Jednoduché* a *složené* atributy.
  - Atributy s jednoduchou hodnotou (*single-valued*) a s násobnou hodnotou (*multi-valued*)
  - *Nulové* atributy (např.: nemá telefon)
  - *Odvozené* atributy (např.: délka zaměstnání)

Množiny vztahů

- Vztah je spojitost mezi několika entitami
  Např.:

Hayes	*vkladatel*	A-102
entita *zákazník*	množina vztahů	entita *účet*

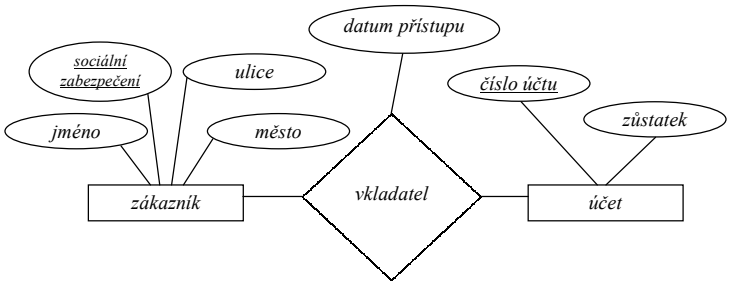
- Množina vztahů je matematická relace mezi $n \geq 2$ entitami, každá je braná z množiny entit

$$\{(e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

kde $(e_1, e_2, \ldots, e_n)$ je vztah, $e_1, e_2, \ldots, e_n$ jsou entity a $E_1, E_2, \ldots, E_n$ množiny entit
  - např.:

$$(\text{Hayes, A-102}) \in \textit{vkladatel}$$

- *Atribut* může být též vlastnost množiny vztahů. Vztah *vkladatel* mezi množinami entit *zákazník* a *účet* může mít atribut *datum přístupu*.

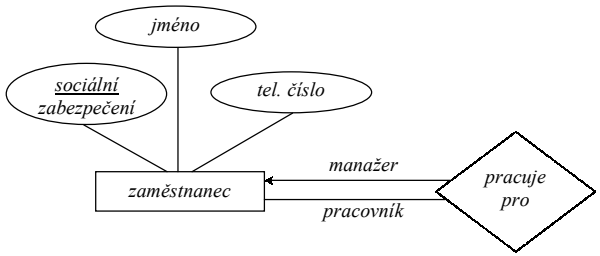


Stupeň množiny vztahů

- Ukazuje počet množin entit, které jsou součástí množiny vztahů.
- Množiny vztahů, které zahrnují 2 množiny entit, se nazývají *binární* (nebo stupně 2). Obecně, většina vztahů v databázovém systému je binární.
- Množiny vztahů můžou zahrnovat více než 2 množiny entit. Např. množiny entit *zákazník (customer)*, *půjčka (loan)* a *pobočka (branch)* můžou být spojeny ternární (stupně 3) množinou vztahů *CLB*.

Role

Množiny entit u vztahů nemusí být rozdílné



- Popisky *manažer* a *pracovník* jsou nazývány *role*; specifikují, jak na sebe entity typu *zaměstnanec* vzájemně působí přes množinu vztahů *pracuje pro*.
- Role jsou v E-R diagramech znázorněny popsáním čar, které spojují obdélníky s kosočtverci (aby se jednalo o role, musí být spojovací čáry alespoň 2).
- Popisky rolí jsou dobrovolné a jsou používány pro zvýraznění sémantiky vztahu

Otázky designu (Design issues)

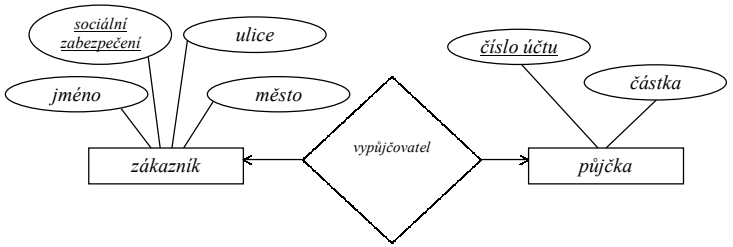
- Použití množin entit vs. atributů
  Výběr závisí zejména na struktuře podniku a na sémantikách spojených s atributem v otázce.
- Použití množin entit vs. množin vztahů
  Možným vodítkem může být sestrojit množinu vztahů pro popis akce, která se odehrává mezi entitami
- Binární vs. *n*-ární množiny vztahů
  Přestože je možné nahradit ne-binární (*n*-ární, pro n > 2) množinu vztahů množstvím různých binárních množin vztahů, *n*-ární ukazuje mnohem jasněji, že několik entit je součástí jednoduchého vztahu.

Mapping cardinalities

- Označuje počet entit, na které mohou být ostatní entity propojeny pomocí množiny vztahů.
- Nejužitečnější je v popisu binárních množin vztahů.
- Pro binární množinu vztahů musí být mapping cardinalities jeden z následujících typů:
  - jedna na jednu
  - jedna na mnoho
  - mnoho na jednu
  - mnoho na mnoho
- Mezi těmito typy rozlišujeme kreslením buď šipky (→) značící *jeden* nebo normální čáry (–) značící *mnoho* mezi množinou entit a vztahů.

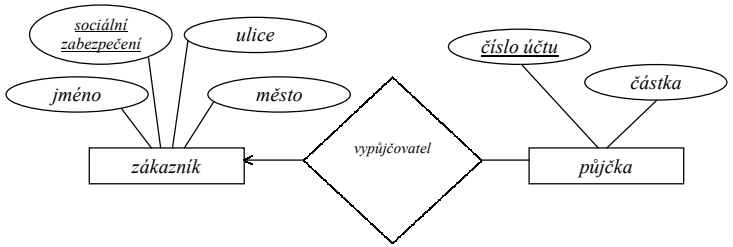
Vztah jedna na jednu (One-to-one)

- Zákazník je spojen s nejvýše jednou půjčkou vztahem *vypůjčovatel*.

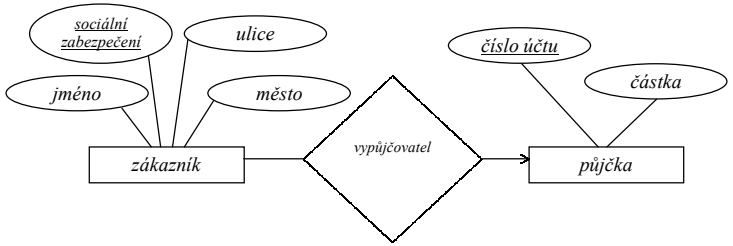


- Půjčka je spojena s nejvýše jedním zákazníkem vztahem *vypůjčovatel*.

Vztahy jedna na mnoho a mnoho na jednu (One-to-many a many-to-one)

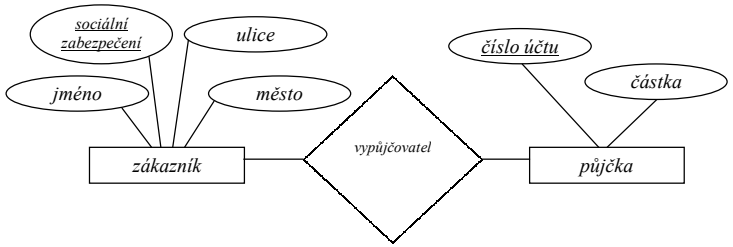


- Ve vztahu jedna na mnoho je půjčka spojena s nejvýše jedním zákazníkem a zákazník je spojen s žádnou nebo několika půjčkami vztahem *vypůjčovatel*.



- Ve vztahu mnoho na jednu je půjčka spojena s žádným nebo několika zákazníky a zákazník je spojen s nejvýše jednou půjčkou vztahem *vypůjčovatel*.

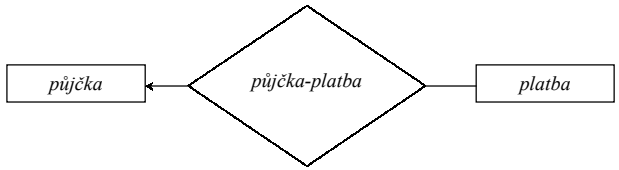
Vztah mnoho na mnoho (Many-to-many)



- Zákazník je spojen s žádnou nebo několika půjčkami vztahem vypůjčovatel
- Půjčka je spojena s žádným nebo několika zákazníky vztahem vypůjčovatel

Existenční závislost

- Závisí-li existence entity *x* na závislosti entity *y*, pak *x* se nazývá existenčně závislé (*existence dependent*) na *y*.
  - *y* je *dominantní entita* (v příkladu níže *půjčka*)
  - *x* je *podřízená entita* (v příkladu níže *platba*)



- Je-li entita *půjčka* smazána, pak všechny s ní spojené entity *platba* musí být smazány také.

Klíče

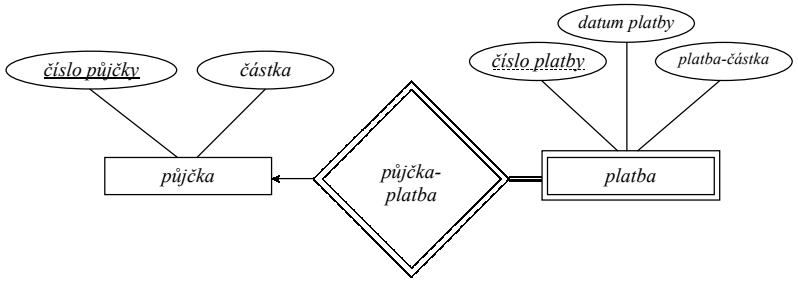
- *Super klíč* množiny entit je množina jednoho nebo více atributů, jejichž hodnoty jednoznačně určují entitu
- *Kandidátní klíč* množiny entit je minimální super klíč.
  - *sociální zabezpečení* je kandidátní klíč entity *zákazník*
  - *číslo účtu* je kandidátní klíč je kandidátní klíč entity *účet*
- Protože může existovat několik kandidátních klíčů, jeden z nich je vybrán jako *primární klíč*.
- Kombinace primárních klíčů zúčastněných množin entit určuje kandidátní klíč množiny vztahů.
  - při výběru *primárního klíče* musíme uznávat mapping cardinality a sémantiku množiny vztahů
  - (*sociální zabezpečení*, *číslo účtu*) je primární klíč množiny vztahů *vkladatel*

Komponenty E-R diagramu

- **Obdélníky** reprezentují množiny entit.
- **Elipsy** reprezentují atributy.
- **Kosočtverce** reprezentují množiny vztahů.
- **Čáry** spojují atributy s množinami entit a množiny entit s množinami vztahů.
- **Dvojité elipsy** reprezentují atributy s násobnou hodnotou.
- **Vyšrafované elipsy** označují odvozené atributy.
- Atributy primárního klíče jsou podtržené.

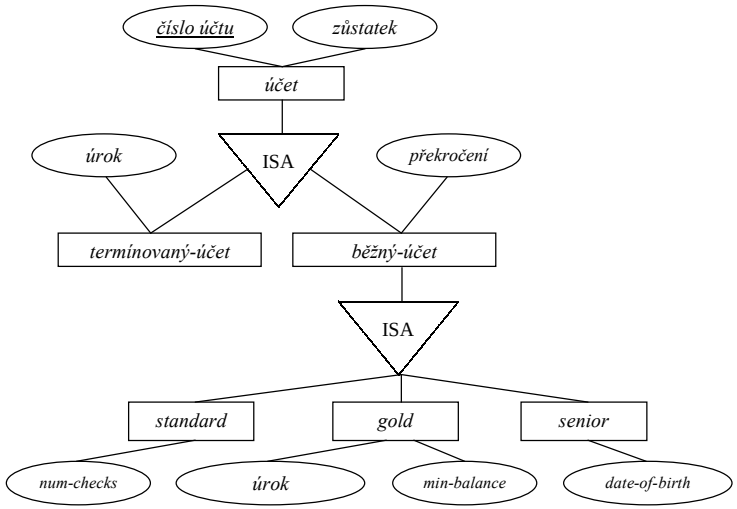
Slabé množiny entit

- Množina entit, která nemá primární klíč, se nazývá *slabá množina entit*.
- Existence slabé množiny entit závisí na existenci silné množiny entit; musí být spojena se silnou množinou vztahem jedna na mnoho.
- *Diskriminátor* (*parciální klíč*) slabé množiny entit je množina atributů, která se liší ve všech entitách slabé množiny.
- Primární klíč slabé množiny je tvořen primárním klíčem silné množiny, na níž je tato množina závislá a parciálním klíčem této slabé množiny.
- Slabé množiny eniti znázorňujeme dvojitým obdélníkem.
- Parciální klíč slabé množiny entit se podtrhává přerušovanou čárou.
- *číslo platby* – parciální klíč množiny entit *platba*
- Primární klíč pro množinu *platba* – (*číslo půjčky*, *číslo platby*)



Specializace

- Tvoříme podskupiny v množině entit, které jsou různé od ostatních entit v množině (proces seshora dolů)
- Tyto podskupiny se stávají množinami entit nižší úrovně, které mají atributy nebo jsou součástí množin vztahů, které se nepromítají do množiny vztahů vyšší úrovně.
- Znázorňujeme trojúhelníkovou komponentou označenou ISA (*termínovaný vklad* „je (*is an*)“ *účet*)

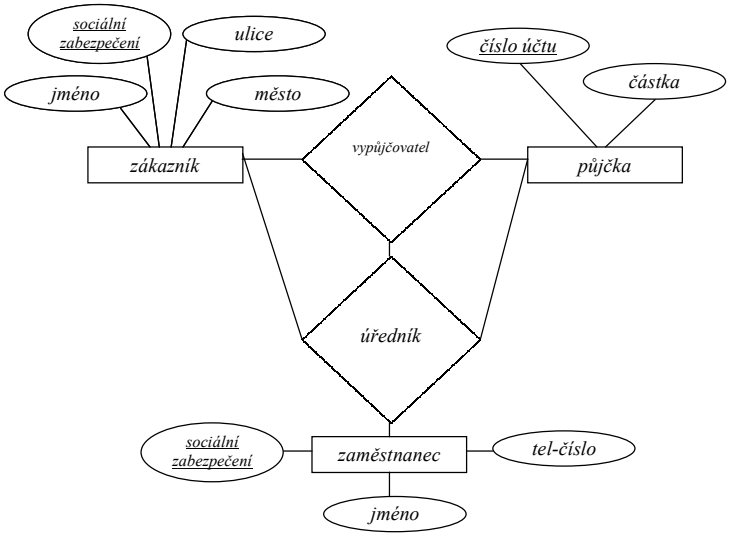


Generalizace

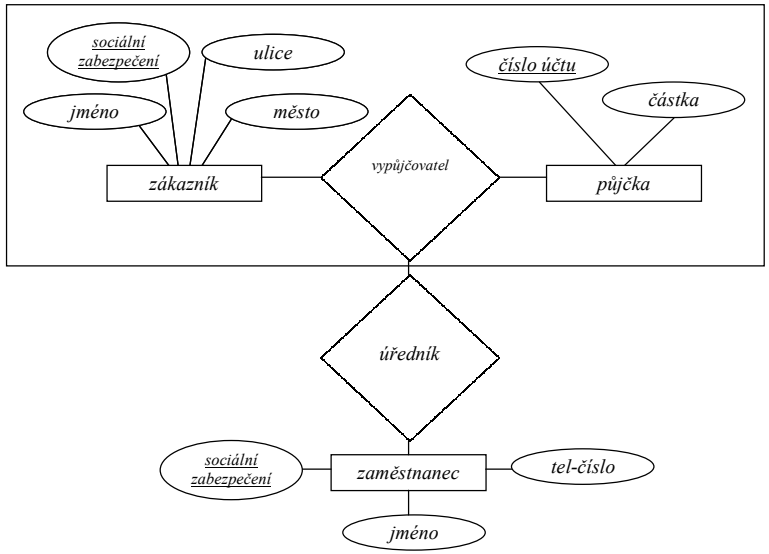
- Kombinujeme několik množin entit, které sdílejí stejné rysy do množiny entit vyšší úrovně (proces zezdola nahoru)
- Specializace a generalizace jsou jednoduše vzájemně inverzní; jsou reprezentovány E-R diagramem stejným způsobem.
- **Dědičnost atributů** – entita nižší úrovně dědí všechny atributy a účasti ve vztazích z množiny entit vyšší úrovně, ke které je připojena.

Agregace

- Věřitelé (*loan customers*) mohou být doporučeni úředníkem (*loan-officer*).



- Vztah určí, že *vypůjčovatel* a *úředník* reprezentují stejnou informaci.
- Tuto redundanci eliminujeme agregací
  - Se vztahem zacházíme jako s abstraktní entitou
  - Umožňuje vztahy mezi vztahy
  - Abstrakce vztahu do nové entity
- Následující diagram reprezentuje:
  - Zákazník si vezme půjčku
  - Zaměstnanec může být úředník pro pár *zákazník-půjčka*



Rozhodnutí o designu E-R

- Použití atributu nebo množiny entit pro reprezentaci objektu.
- Je koncept reálného světa nejlépe vyjádřen množinou entit nebo množinou vztahů?
- Použití ternárního vztahu vs. páru bínárních vztahů.
- Použití silných nebo slabých množin entit.
- Použití generalizace – přispívá k modularitě designu.
- Použití agregace – můžeme zacházet s agregovanou množinou entit jako s jednotkou bez koncentrace na její detaily a vnitřní strukturu.

Redukce E-R schématu na tabulky

- Primární klíče umožňují vyjádřit množiny entit a vztahů jako tabulky reprezentující obsah databáze.
- Databáze, která odpovídá E-R diagramu, může být reprezentována jako kolekce tabulek.
- Pro každou množinu entit a vztahů je jedinečná tabulka, která je spojená se jménem odpovídající množiny entit nebo vztahů.
- Každá tabulka má počet sloupců odpovídající atributům, které mají jedinečná jména.
- Převod E-R diagramu na tabulku je základ pro odvození designu relační databáze z E-R diagramu.

Reprezentace množin entit tabulkami

- Silná množina entit se převede na tabulku se stejnými atributy.

jméno	*sociální zabezpečení*	*ulice*	*město*
Jones	321-12-3123	Main	Harrison
Smith	019-28-3746	North	Rye
Hayes	677-89-9011	Main	Harrison

Tabulka *zákazník*

- Slabá množina entit se převede na tabulku, která zahrnuje sloupec pro primární klíč identifikační silné množiny entit.

číslo půjčky	*číslo platby*	*datum platby*	*částka platby*
L-17	5	10.5.1996	50
L-23	11	17.5.1996	75
L-15	22	23.5.1996	300

Tabulka *platba*

Reprezentace množin vztahů tabulkami

- Množina vztahů mnoho na mnoho je reprezentována jako tabulka se sloupci pro primární klíče dvou účastnících se množin entit a popisné atributy množiny vztahů.

sociální zabezpečení	*číslo účtu*	*datum přístupu*
…	…	…

Tabulka *vkladatel*

- Tabulka odpovídající množině vztahů spojující slabou množinu entit s její identifikační silnou množinou je redundantní. Tabulka *platba* již obsahuje informace, které by se objevily v tabulce *půjčka-platba* (tj. sloupce *číslo půjčky* a *číslo platby*)

Reprezentace generalizace tabulkami

- Metoda č. 1: Sestrojíme tabulku pro generalizovanou entitu *účet*. Sestrojíme tabulku pro každou množinu entit, která je generalizovaná (zahrneme primární klíč generalizované množiny)

tabulka	atributy tabulky
účet	*číslo účtu, zůstatek, typ účtu*
termínovaný účet	*číslo účtu, invest-rate*
běžný účet	*číslo účtu, překročení*

- Metoda č. 2: Sestrojíme tabulku pro každou množinu entit, která je generalizovaná

tabulka	atributy tabulky
termínovaný účet	*číslo účtu, zůstatek, invest-rate*
běžný účet	*číslo účtu, zůstatek, překročení*

Metoda č. 2 netvoří žádnou tabulku pro generalizovanou entitu *účet*.

Vztahy odpovídající agregaci

zákazník

jméno	*sociální zabezpečení*	*ulice*	*město*

půjčka

číslo půjčky	*částka*

vypůjčovatel

sociální zabezpečení	*číslo půjčky*

zaměstnanec

sociální zabezpečení	*jméno*	*tel. číslo*

úředník

sociální zabezpečení (zam.)	*sociální zabezpečení (zák.)*	*číslo půjčky*

Kapitola 3: Relační model

- Struktura relačních databází
- Relační algebra
- Relační kalkul n-tic (*Tuple relational calculus*)
- Doménový relační kalkul
- Rozšířené operace relační algebry
- Modifikace databáze
- Pohledy

Základní struktura

- Daná množina A_1, A_2, …, A_n *relace r* je podmnožina $A_1 \times A_2 \times \ldots \times A_n$. Tedy relace je *množina n-tic* (a_1, a_2, …, a_n), kde $a_i \in A_i$
- Příklad: je-li
  jméno = {Jones, Smith, Curry, Lindsay}
  ulice = {Main, North, Park}
  město = {Harrison, Rye, Pittsfield}
  pak *r* = {(Jones, Main, Harrison), (Smith, North, Rye), (Curry, North, Rye), (Lindsay, Park, Pittsfield)} je relace přes *jméno* × *ulice* × *město*

Relační schéma

- A_1, A_2, …, A_n jsou *atributy*
- $R = (A_1, A_2, \ldots, A_n)$ je *relační schéma*
  zákazník-schéma = (*jméno*, *ulice*, *město*)
- $r(R)$ je *relace* na relačním schématu R
  zákazník (*zákazník-schéma*)

Instance relace

- Současné hodnoty (*instance relace*) jsou specifikovány tabulkou.
- Element *t* relace *r* je *n-tice* reprezentovaná *řádkem* v tabulce.

jméno	*ulice*	*město*
Jones	Main	Harrison
Smith	North	Rye
Curry	North	Rye
Lindsay	Park	Pittsfield

zákazník

Klíče

- Nechť $K \subseteq R$
- K je superklíč schématu R, jestliže hodnoty K jsou dostatečné pro identifikaci jedinečné n-tice každé možné relace $r(R)$. Např.: {*jméno*, *ulice*} a {*jméno*} jsou superklíče schématu *zákazník*, jestliže dva zákazníci nemají shodné jméno.
- K je *kandidátní klíč*, jestliže K je minimální. Např.: {*jméno*} je kandidátní klíč schématu *zákazník*, jestliže dva zákazníci nemají shodné jméno a žádná jeho podmnožina není superklíč.

Odvozování klíčů z E-R množin

- **Silná množina entit.** Primární klíč množiny se stává primárním klíčem relace.
- **Slabá množina entit.** Primární klíč relace je sjednocení primárního klíče silné množiny entit a parciálního klíče slabé množiny.
- **Množina vztahů.** Sjednocení primárních klíčů příbuzných množin entit se stává superklíčem relace. Pro binární vztahy mnoho na mnoho je superklíč též primární klíč. Pro binární vztahy mnoho na jednu se stává primární klíč množiny entit „mnoho“ primárním klíčem relace. Pro vztahy jedna na jednu může být primární klíč relace z každé množiny entit.

Dotazovací jazyky

- Jazyk, kterým uživatel žádá informace z databáze.
- Kategorie jazyků:
  - Procedurální
  - Neprocedurální
- Čisté jazyky („*pure*“):
  - Relační algebra
  - Relační kalkul n-tic
  - Doménový relační kalkul
- Čisté jazyky jsou základem dotazovacích jazyků, které se běžně používají.

Relační algebra

- Procedurální jazyk
- 6 základních operátorů
  - výběr
  - projekce
  - sjednocení
  - rozdíl množin
  - kartézský součin
  - přejmenování
- Operátory berou jednu nebo více relací jako vstup a dávají novou relaci jako výsledek.

Operace výběr

- Značení: $\sigma_P(r)$
- Definováno jako:

$$\sigma_P(r) = \{t \mid t \in r \text{ and } P(t)\}$$

Kde P je výraz v propozičním kalkulu, skládající se z podmínek ve formě:

<atribut> = <atribut> nebo <konstanta>
$\neq$
$>$
$\geq$
$<$
$\leq$

„spojené pomocí“: $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

Příklad výběru

- Relace *r*:

A	*B*	*C*	*D*
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A = B \wedge D > 5}(r)$ – vybere řádky, kde A = B a zároveň D > 5

A	*B*	*C*	*D*
α	α	1	7
β	β	23	10

Operace projekce

- Značení: $\Pi_{A_1, A_2, \ldots, A_k}(r)$

kde A_1, A_2 jsou jména atributů a *r* je jméno relace.

- Výsledek je definován jako relace *k* sloupců, které dostaneme smazáním sloupců, které nejsou vyjmenovány
- Duplicitní řádky jsou z výsledku odstraněny

Příklad projekce

- Relace *r*:

A	*B*	*C*
α	10	1
α	20	1
β	30	1
β	40	2

- $\Pi_{A, C}(r)$

A	*C*
α	1
~~α~~	~~1~~
β	1
β	2

=

A	*C*
α	1
β	1
β	2

Operace sjednocení

- Značení: $r \cup s$
- Definováno jako:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- Aby bylo sjednocení $r \cup s$ platné,
  - *r* a *s* musí mít *stejnou aritu* (stejný počet atributů)
  - Domény atributů musí být kompatibilní (např. 2. sloupec *r* obsahuje hodnoty stejného typu jako 2. sloupec *s*)

Příklad sjednocení

- Relace *r*, *s*:

A	*B*
α	1
α	2
β	1

r

A	*B*
α	2
β	3

s

- $r \cup s$

A	*B*
α	1
α	2
β	1
β	3

Operace rozdíl množin

- Značení: $r - s$
- Definováno jako:

$$r - s = \{t \mid t \subset r \text{ and } t \not\subset s\}$$

- Rozdíly množin musí být brány mezi kompatibilními relacemi.
  - r a s musí mít stejnou aritu
  - domény atributů relací r a s musí být kompatibilní

Příklad rozdílu množin

- Relace r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r - s$

A	B
α	1
β	1

Operace kartézského součinu

- Značení: $r \times s$
- Definováno jako:

$$r \times s = \{t\, q \mid t \in r \text{ and } q \in s\}$$

- Předpokládá se, že atributy relací $r(R)$ a $s(S)$ jsou disjunktní (tj. $R \cap S = \emptyset$).
- Pokud atributy relací $r(R)$ a $s(S)$ nejsou disjunktní, musí se přejmenovat.

Příklad kartézského součinu

- Relace r, s:

A	B
α	1
β	2

r

C	D	E
α	10	+
β	10	+
β	20	-
γ	10	-

s

- $r \times s$

A	B	C	D	E
α	1	α	10	+
α	1	β	10	+
α	1	β	20	-
α	1	γ	10	-
β	2	α	10	+
β	2	β	10	+
β	2	β	20	-
β	2	γ	10	-

Skládání operací

- Můžeme tvořit výrazy skládáním operací
- Příklad: $\sigma_{A=C}(r \times s)$
- $r \times s$
  - Značení: $r \times s$
  - Nechť r a s jsou relace na schématech R a S. Výsledek je relace na schématu $R \cup S$, kterou získáme uvažováním každého páru n-tic t_r z r a t_s z s.
  - Mají-li t_r a t_s stejné hodnoty na každém z atributů v $R \cap S$, je k výsledku přidána n-tice t, kde
    - t má stejnou hodnotu jako t_r na r
    - t má stejnou hodnotu jako t_s na s

Příklad:
$R = (A, B, C, D)$
$S = (E, B, D)$

- Výsledné schéma = (A, B, C, D, E)
- $r \times s$ je definováno jako:

$$\Pi_{r.A,r.B,r.C,r.D,s.E}(\sigma_{r.B=s.B \wedge r.D=s.D}(r \times s))$$

Příklad přirozeného spojení (Natural join operation)

- Relace *r*, *s*:

A	*B*	*C*	*D*
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	*D*	*E*
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- $r \times s$

A	*B*	*C*	*D*	*E*
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Operace dělení

- Značení: $r \div s$
- Odpovídá dotazům, které obsahují frázi „pro všechny“.
- Nechť *r* a *s* jsou relace na schématech *R* a *S*, kde
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  Výsledek operace $r \div s$ je relace na schématu $R - S = (A_1, \ldots, A_m)$

  $$r \div s = \{t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s(tu \in r)\}$$
- Vlastnost
  - Nechť $q = r \div s$
  - Pak *q* je největší relace, pro kterou platí: $q \times s \subseteq r$
- Definice pomocí podmínek základních operací algebry. Nechť $r(R)$ a $s(S)$ jsou relace a nechť $S \subseteq R$

  $$r \div s = \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$
- Vysvětlení:
  - $\Pi_{R-S,S}(r)$ jednoduše přeřadí atributy *r*
  - $\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ dává ty n-tice *t* v $\Pi_{R-S}(r)$, že pro nějakou n-tici $u \in s$, $tu \notin r$

Příklad dělení

- Relace *r*, *s*:

A	*B*
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
δ	6
ε	1
ε	2

r

B
1
2

s

- $r \div s$

A
α
ε

Jiný příklad dělení

- Relace *r*, *s*:

A	*B*	*C*	*D*	*E*
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

r

D	*E*
a	1
b	1

s

- $r \div s$

A	*B*	*C*
α	a	γ
γ	a	γ

Operace přiřazení

- Operace přiřazení (←) představuje vhodný způsob, jak vyjádřit komplexní dotazy; dotazy se píší jako sekvenční program skládající se ze skupiny přiřazení následovaných výrazem, jehož hodnota je zobrazena jako výsledek dotazu.
- Přiřazení musí být vždy prováděno na dočasné relační proměnné.
- Příklad: Napišme $r \div s$ jako
  $temp1 \leftarrow \Pi_{R\text{-}S}(r)$
  $temp2 \leftarrow \Pi_{R\text{-}S}((temp1 \times s) - \Pi_{R\text{-}S,S}(r))$
  $result = temp1 - temp2$
  - Výsledek vpravo od ← je přiřazen do relační proměnné vlevo od ←.

Příklady dotazů

- Najděte všechny zákazníky, kteří mají účet aspoň v pobočkách „Downtown“ a „Uptown“.
  - Dotaz 1
    $\Pi_{CN}(\sigma_{BN = \text{„Downtown“}}(vkladatel \times účet)) \cap \Pi_{CN}(\sigma_{BN = \text{„Uptown“}}(vkladatel \times účet))$
    kde *CN* značí *jméno-zákazníka*, *BN* značí *jméno-pobočky* a *BC* značí *město-pobočky*
  - Dotaz 2
    $\Pi_{CN, BN}(vkladatel \times účet) \div \rho_{temp(BN)}(\{(\text{„}Downtown\text{“}), (\text{„}Uptown\text{“})\})$
- Najděte všechny zákazníky, kteří mají účet ve všech pobočkách v Brooklynu.
  $\Pi_{CN, BN}(vkladatel \times účet) \div \Pi_{BN}(\sigma_{BC = \text{„Brooklyn“}}(pobočka))$

Relační kalkul n-tic (Tuple relational calculus)

- Neprocedurální dotazovací jazyk, kde každý dotaz je ve tvaru
  $\{t \mid P(t)\}$
- Je to množina všech n-tic *t* takových, že predikát *P* je pravdivý pro *t*
- *t* je proměnná n-tice (*tuple variable*); *t*[*A*] značí hodnotu n-tice *t* na atributu *A*
- $t \in r$ značí, že n-tice *t* je v relaci *r*
- *P* je výraz (*formula*) podobný tomu stejnému v predikátovém kalkulu

Výraz v predikátovém kalkulu

- Množina atributů a konstant
- Množina operátorů porovnání: (např.: <, ≤, =, ≠, >, ≥)
- Množina spojek: and (∧), or (∨), not (¬)
- Implikace (⇒): $x \Rightarrow y$, jestliže *x* je pravdivé, pak *y* také
  $x \Rightarrow y \equiv \neg x \vee y$
- Množina kvantifikátorů:
  - $\exists t \in r\,(Q(t)) \equiv$ existuje n-tice *t* v relaci *r* tak, že platí *Q*(*t*)
  - $\forall t \in r\,(Q(t)) \equiv$ pro všechny n-tice *t* v relaci *r* platí *Q*(*t*)

Příklad (banka)

pobočka (pobočka-jméno, pobočka-město, aktivum)
zákazník (zákazník-jméno, zákazník-ulice, zákazník-město)
účet (pobočka-jméno, číslo-účtu, částka)
půjčka (pobočka-jméno, půjčka-číslo, částka)
vkladatel (zákazník-jméno, číslo-účtu)
půjčovatel (zákazník-jméno, půjčka-číslo)

Příklady dotazů

- Najděte *pobočka-jméno, půjčka-číslo* a *částka* pro půjčky přes $1200
  $\{t \mid t \in půjčka \wedge t[částka] > 1200\}$
- Najděte číslo půjčky pro každou půjčku s částkou větší než $1200
  $\{t \mid \exists s \in půjčka\,(t[půjčka\text{-}číslo] = s[půjčka\text{-}číslo] \wedge s[částka] > 1200)\}$

Poznámka: Relace na schématu [*zákazník-jméno*] je implicitně definována dotazem

- Najděte jména všech zákazníků, kteří mají v bance půjčku, účet nebo obojí
  $\{t \mid \exists s \in půjčovatel(t[zákazník\text{-}jméno] = s[zákazník\text{-}jméno]) \vee$
  $\exists u \in vkladatel(t[zákazník\text{-}jméno] = u[zákazník\text{-}jméno])\}$
- Najděte jména všech zákazníků, kteří mají v bance půjčku a účet
  $\{t \mid \exists s \in půjčovatel(t[zákazník\text{-}jméno] = s[zákazník\text{-}jméno]) \wedge$
  $\exists u \in vkladatel(t[zákazník\text{-}jméno] = u[zákazník\text{-}jméno])\}$
- Najděte jména všech zákazníků, kteří mají půjčku v pobočce Perryridge
  $\{t \mid \exists s \in půjčovatel(t[zákazník\text{-}jméno] = s[zákazník\text{-}jméno] \wedge$
  $\exists u \in půjčka(u[pobočka\text{-}jméno] = \text{„Perryridge“} \wedge u[půjčka\text{-}číslo] = s[půjčka\text{-}číslo]))\}$
- Najděte jména všech zákazníků, kteří mají půjčku v pobočce Perryridge, ale nemají účet v žádné pobočce banky.
  $\{t \mid \exists s \in půjčovatel(t[zákazník\text{-}jméno] = s[zákazník\text{-}jméno] \wedge$
  $\exists u \in půjčka(u[pobočka\text{-}jméno] = \text{„Perryridge“} \wedge u[půjčka\text{-}číslo] = s[půjčka\text{-}číslo]) \wedge$
  $\exists v \in vkladatel\,(v[zákazník\text{-}jméno] = t[zákazník\text{-}jméno])\}$

- Najděte jména všech zákazníků, kteří mají půjčku v pobočce Perryridge a města, ze kterých jsou

$$\{t \mid \exists\, s \in \textit{půjčka}(s[\textit{pobočka-jméno}] = \text{„Perryridge“} \wedge$$

$$\exists u \in \textit{půjčovatel}\ (u[\textit{půjčka-číslo}] = s[\textit{půjčka-číslo}] \wedge$$

$$t[\textit{zákazník-jméno}] = u[\textit{zákazník-jméno}] \wedge$$

$$\exists v \in \textit{zákazník}\ (u[\textit{zákazník-jméno}] = v[\textit{zákazník-jméno}] \wedge$$

$$t[\textit{zákazník-město}] = v[\textit{zákazník-město}]))))\}$$

- Najděte jména všech zákazníků, kteří mají účet ve všech pobočkách umístěných v Brooklynu:

$$\{t \mid \forall s \in \textit{pobočka}\ (s[\textit{pobočka-město}] = \text{„Brooklyn“} \Rightarrow$$

$$\exists\, u \in \textit{účet}\ (s[\textit{pobočka-jméno}] = u[\textit{pobočka-jméno}] \wedge$$

$$\exists\, s \in \textit{vkladatel}\ (t[\textit{zákazník-jméno}] = s[\textit{zákazník-jméno}] \wedge$$

$$s[\textit{číslo-účtu}] = u[\textit{číslo-účtu}]))))\}$$

Bezpečnost výrazů

- V kalkulu n-tic je možné psát výrazy, které generují nekonečné relace
- Např. $\{t \mid \neg t \in r\}$ vede k nekonečné relaci, je-li doména jakéhokoli atributu relace r nekonečná
- Abychom tomuto zamezili, omezíme množinu použitelných výrazů na *bezpečné* výrazy
- Výraz $\{t \mid P(t)\}$ v relačním kalkulu n-tic je *bezpečný*, jestliže se každá komponenta t objeví v jedné z relací, n-tic nebo konstant v P.

Doménový relační kalkul

- Neprocedurální dotazovací jazyk, který je v síle ekvivalentní relačnímu kalkulu n-tic.
- Každý dotaz je výraz ve tvaru:

$$\{< x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

  - $x_1, x_2, \ldots, x_n$ reprezentují doménové proměnné
  - P reprezentuje formuli podobnou té stejné z predikátového kalkulu

Příklady dotazů

- Najděte *pobočka-jméno (b)*, *půjčka-číslo (l)* a *částka (a)* pro půjčky přes $1200:

$$\{< b, l, a > \mid < b, l, a > \in \textit{půjčka} \wedge a > 1200\}$$

- Najděte jména všech zákazníků *(c)*, kteří mají půjčku přes $1200:

$$\{< c > \mid \exists\, b, l, a\ (< c, l > \in \textit{půjčovatel} \wedge < b, l, a > \in \textit{půjčka} \wedge a > 1200)\}$$

Pozn.: musíme uvést tolik proměnných, s kolika relacemi pracujeme (3 proměnné: *b, l, a*; 3 relace: *půjčovatel, půjčka, 1200*)

- Najděte jména všech zákazníků, kteří mají půjčku u pobočky Perryridge a částku půjčky:

$$\{< c, a > \mid \exists\, l\ (< c, l > \in \textit{půjčovatel} \wedge \exists\, b\ (< b, l, a > \in \textit{půjčka} \wedge b = \text{„Perryridge“}))\}$$

- Najděte jména všech zákazníků, kteří mají půjčku, účet nebo obojí u pobočky Perryridge:

$$\{< c > \mid \exists\, l\ (< c, l > \in \textit{půjčovatel} \wedge \exists\, b, a\ (< b, l, a > \exists\, \textit{půjčka} \wedge b = \text{„Perryridge“})$$

$$\vee \exists\, a\ (< c, a > \in \textit{půjčovatel} \wedge \exists\, b, n\ (< b, a, n > \in \textit{účet} \wedge b = \text{„Perryridge“}))\}$$

- Najděte jména všech zákazníků, kteří mají účet ve všech pobočkách umístěných v Brooklynu:

$$\{< c > \mid \forall\, x, y, z\ (< x, y, z > \in \textit{pobočka} \wedge y = \text{„Brooklyn“}) \Rightarrow$$

$$\exists\, a, b\ (< x, a, b > \in \textit{účet} \wedge < c, a > \in \textit{vkladatel})\}$$

Bezpečnost výrazů

$$\{< x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

je bezpečný výraz, platí-li následující:

- Všechny hodnoty, které se objeví v n-ticích výrazu, jsou hodnoty z $dom(P)$, tj. hodnoty, které se objeví buď v P nebo n-ticích relace zmíněné v P.
- Pro každou podformuli „existuje“ tvaru $\exists\, x\ (P_1(x))$ je tato podfurmule pravdivá jen tehdy, je-li hodnota x v $dom(P)$, takže je $P_1(x)$ pravdivá.
- Pro každou podformuli „pro všechny“ tvaru $\forall\, x\ (P_1(x))$ je tato podformule pravdivá jen tehdy, je-li $P_1(x)$ pravdivá pro všechny hodnoty x z $dom(P)$.

Rozšířené operace relační algebry

- Zobecněná projekce
- Vnější spojení (Outer join)
- Souhrnné funkce

Zobecněná projekce

- Rozšiřuje operaci projekce tak, že umožňuje použití aritmetických funkcí v seznamu projekce.

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

- E je jakýkoliv výraz relační algebry
- Každá z F_1, F_2, …, F_n jsou aritmetické výrazy zahrnující konstanty a atributy ve schématu E.
- Daná relace *credit-info (name, limit, balance*) najde, kolik může každá osoba utratit:

$$\Pi_{name,\ limit - balance}(credit\text{-}info)$$

Vnější spojení (Outer join)

- Rozšíření operace spojení, která zabraňuje ztrátě informací.
- Spočítá operaci spojení a přidá n-tice z jedné relace, které neodpovídají n-ticím v druhé relaci k výsledkům operace spojení.
- Používá hodnotu *null*:
  - *null* značí, že hodnota je neznámá nebo neexistuje
  - všechna porovnávání zahrnující *null* mají z definice hodnotu **false**.

Příklad

- Relace *půjčka*

pobočka-jméno	*půjčka-číslo*	*částka*
Downtown	L-170	3000
Redwood	L-230	4000
Perryridge	L-260	1700

- Relace *půjčovatel*

zákazník-jméno	*půjčka-číslo*
Jones	L-170
Smith	L-230
Hayes	L-260

- *půjčka* × *půjčovatel*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith

- *půjčka* × *půjčovatel*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*	*půjčka-číslo*
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230
Perryridge	L-260	1700	*null*	*null*

- *půjčka* × *půjčovatel*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
null	L-155	*null*	Hayes

- *půjčka* × *půjčovatel*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
Perryridge	L-260	1700	*null*
null	L-155	*null*	Hayes

Souhrnné funkce

- Souhrnný operátor ς vezme kolekci hodnot a vrátí jednoduchou hodnotu jako výsledek.

  avg: průměrná hodnota
  min: minimální hodnota
  max: maximální hodnota
  sum: suma hodnot
  count: počet hodnot

$$G_1, G_2, \ldots, G_n\ \varsigma\ F_1 A_1, F_2 A_2, \ldots, F_m A_m\ (E)$$

  - E je jakýkoliv výraz relační algebry
  - G_1, G_2, …, G_n je seznam atributů ke spojení
  - F_i je souhrnná funkce
  - A_i je jméno atributu

Příklad

- Relace *r*

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

- $\text{sum}_C(r)$: 27
- Relace *účet* seskupená podle *pobočka-jméno*:

pobočka-jméno	*číslo-účtu*	*zůstatek*
Perryridge	A-102	400
Perryridge	A-201	900
Brightoh	A-217	750
Brightoh	A-215	750
Redwood	A-222	700

- $_{pobočka\text{-}jméno}\varsigma_{\text{sum }zůstatek}(účet)$

pobočka-jméno	*suma-zůstatek*
Perryridge	1300
Brightoh	1500
Redwood	700

Modifikace databáze

- Obsah databáze lze měnit následujícími operacemi:
  - Mazání
  - Vložení
  - Aktualizace
- Tyto operace jsou vyjadřovány operátorem přiřazení.

Mazání

- Požadavek na mazání je vyjádřen podobně jako dotaz, ale n-tice nejsou poslány uživateli, ale smazány z databáze.
- Lze mazat pouze celé n-tice, ne pouze hodnoty ve zvláštních atributech.
- V relační algebře je mazání vyjádřeno takto:

$$r \leftarrow r - E$$

kde r je relace a E je dotaz relační algebry.

Příklady mazání

- Smaže všechny záznamy o účtech v pobočce Perryridge.

$$account \leftarrow account - \varsigma_{pobočka\text{-}jméno = \text{„Perryridge“}}(účet)$$

- Smaže všechny záznamy o půjčkách s částkou v rozmezí 0–50.

$$půjčka \leftarrow půjčka - \varsigma_{částka \geq 0 \text{ and } částka \leq 50}(půjčka)$$

- Smaže všechny účty v pobočkách umístěných v Needhamu.

$$r_1 \leftarrow \sigma_{pobočka\text{-}město = \text{„Needham“}}(účet \times pobočka)$$
$$r_2 \leftarrow \Pi_{pobočka\text{-}jméno,\ číslo\text{-}účtu,\ zůstatek}(r_1)$$
$$r_3 \leftarrow \Pi_{zákazník\text{-}jméno,\ číslo\text{-}účtu}(r_2 \times vkladatel)$$
$$účet \leftarrow účet - r_2$$
$$vkladatel \leftarrow vkladatel - r_3$$

Vložení

- Abychom přidali data do relace, buď:
  - specifikujeme n-tici, která má být přidána
  - napíšeme dotaz, jehož výsledek je množina n-tic, která má být přidána
- V relační algebře je vložení vyjádřeno takto:

$$r \leftarrow r \cup E$$

kde r je relace a E je dotaz relační algebry.
- Vložení jednoduché n-tice je vyjádřeno tak, že E může být konstantní relace obsahující jednu n-tici.

Příklady vložení

- Vložte informaci do databáze: Smith má $1200 na účtu A-973 v pobočce Perryridge.

$$účet \leftarrow účet \cup \{(\text{„Perryridge“}, A\text{-}973, 1200)\}$$
$$vkladatel \leftarrow vkladatel \cup \{(\text{„Smith“}, A\text{-}973)\}$$

- Jako dárek poskytněte všem zákazníkům půjček v pobočce Perryridge $200 termínovaný účet. Nechť číslo půjčky slouží jako číslo účtu pro nový termínovaný účet.

$$r_1 \leftarrow (\sigma_{pobočka\text{-}jméno = \text{„Perryridge“}}(půjčovatel \times půjčka))$$
$$účet \leftarrow účet \cup \Pi_{pobočka\text{-}jméno,\ půjčka\text{-}číslo,\ 200}(r_1)$$
$$vkladatel \leftarrow vkladatel \cup \Pi_{zákazník\text{-}jméno,\ půjčka\text{-}číslo}(r_1)$$

Aktualizace

- Mechanismus jak změnit hodnoty v n-tici, aniž by musely být změněny *všechny*
- K tomu se používá operátor zobecněné projekce

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(r)$$

  - F_i je i-tý atribut relace r, jestliže i-tý atribut není aktualizován nebo má-li být aktualizován
  - F_i je výraz obsahující pouze konstanty a atributy r, který dává novou hodnotu atributu

Příklady aktualizace

- Zvyšte všechny zůstatky o 5 %.

$$účet \leftarrow \Pi_{BN,\ AN,\ BAL \leftarrow BAL * 1.05}(účet)$$

kde *BAL, BN* a *AN* jsou *zůstatek, pobočka-jméno* a *číslo-účtu*.
- Vyplaťte všechny účty se zůstatkem přes $10,000 6% úrokem, ostatní 5% úrokem.

$$účet \leftarrow \Pi_{BN,\ AN,\ BAL \leftarrow BAL * 1.06}(\sigma_{BAL > 10000}(účet)) \cup$$
$$\Pi_{BN,\ AN,\ BAL \leftarrow BAL * 1.05}(\sigma_{BAL \leq 10000}(účet))$$

Pohledy

- V některých případech není vhodné, aby všichni uživatelé viděli celý logický model databáze (tj. všechny relace aktuálně uložené v databázi).
- Představme si osobu, která potřebuje vědět o počtu půjček zákazníkům, ale nemusí vidět částku půjčky. Tato osoba by měla vidět relaci popsanou v relační algebře takto:

$$\Pi_{zákazník\text{-}jméno,\ půjčka\text{-}číslo}(půjčovatel \times půjčka)$$

- Jakákoli relace, která není součástí koncepčního modelu, ale je viditelná uživateli jako „virtuální relace“, se nazývá *pohled*.

Definice pohledu

- Pohled je definován statementem **create view**, který má tvar
  create view *v* as <výraz dotazu>
  kde <výraz dotazu> je jakýkoliv správný výraz relační algebry. Jméno pohledu je reprezentováno proměnnou *v*.
- Jakmile je pohled definován, jeho jméno může být použito pro odkazování na virtuální relaci, která pohled generuje.
- Definice pohledu není to samé jako vytvoření nové relace vyhodnocením dotazovacího výrazu. Definice pohledu způsobuje uložení výrazu, aby byl substituován do dotazů použitím tohoto pohledu.

Příklady pohledů

- Představme si pohled (pojmenovaný *všichni-zákazníci*) skládající se z poboček a jejich zákazníků.

  create view *všichni-zákazníci* **as**
  $\Pi_{\textit{pobočka-jméno, zákazník-jméno}}(\textit{vkladatel} \times \textit{účet})$
  $\cup\ \Pi_{\textit{pobočka-jméno, zákazník-jméno}}(\textit{půjčovatel} \times \textit{účet})$
- Nyní můžeme najít všechny zákazníky pobočky Perryridge napsáním:
  $\Pi_{\textit{zákazník-jméno}}(\sigma_{\textit{pobočka-jméno = „Perryridge“}}(\textit{všichni-zákazníci}))$

Aktualizace přes pohledy

- Modifikace databáze s použitím pohledů musí být překládány na modifikace aktuálních relací v databázi.
- Představme si osobu, která potřebuje vidět všechna data o půjčkách v relaci *půjčka* s výjimkou *částky*. Pohled, který ji poskytneme (*pobočka-půjčka*), je definován takto:

  create view *pobočka-půjčka* **as** $\Pi_{\textit{pobočka-jméno, půjčka-číslo}}(\textit{půjčka})$

  Umožníme-li, aby se jméno pohledu objevilo kdekoliv, kde je akceptováno jméno relace, osoba může psát:

  $\textit{pobočka-půjčka} \leftarrow \textit{pobočka-půjčka} \cup \{(\text{„Perryridge“, L-37})\}$
- Předchozí vkládání musí být reprezentováno vkládáním do aktuální relace *půjčka*, ze které byl pohled *pobočka-půjčka* vytvořen.
- Vkládání do *půjčka* vyžaduje hodnotu pro *částka*. Vkládání se může chovat takto:
  - odmítne vkládání a vrátí chybové hlášení
  - vloží n-tici („Perryridge“, L-37, *null*) do relace *půjčka*

Pohledy definované použitím jiných pohledů

- Pouze jeden pohled může být použit při definování jiného.
- Relace pohledu v_1 *závisí přímo na* relaci pohledu v_2, jestliže v_2 je použit ve výrazu definujícím v_1.
- Relace pohledu v_1 *závisí na* relaci pohledu v_2, jestliže v grafu závislostí je cesta z v_2 do v_1.
- Relace pohledu *v* je *rekurzivní*, jestliže závisí sama na sobě.

Expanze pohledu

- Cesta, jak definovat význam pohledů definovaných podmínkami ostatních pohledů.
- Nechť pohled v_1 je definován výrazem e_1, který může sám obsahovat použití relací pohledů.
- Expanze pohledu výrazu opakuje následující nahrazovací krok:
  repeat
    najdi jakoukoli relaci v_i v e_1
    nahraď relaci pohledu v_i výrazem definujícím v_i
  until v e_1 nejsou přítomny další relace pohledů
- Až nebudou definice pohledů rekurzivní, tato smyčka skončí.

Kapitola 4: SQL

- Základní struktura
- Množinové operace
- Souhrnné funkce
- Nulové hodnoty
- Vnořené poddotazy (Nested subqueries)
- Odvozené relace
- Pohledy
- Modifikace databáze
- Spojené relace
- Jazyk definice dat (Data definition language)
- Vložený SQL

Základní struktura

- SQL je založen na množinových a relačních operacích s několika modifikacemi a vylepšeními
- Typický SQL dotaz má tvar:
  select A_1, A_2, …, A_n
  from r_1, r_2, …, r_m
  where P
  - A_i reprezentuje atributy
  - r_i reprezentuje relace
  - P je predikát
- Tento dotaz je ekvivalentní následujícímu výrazu relační algebry:
  $$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$
- Výsledek každého SQL dotazu je relace.

Klauzule select

- Klauzule **select** odpovídá operaci projekce v relační algebře. Je používána k vypsání požadovaných atributů ve výsledku dotazu.
- Najděte jména všech poboček v relaci *půjčka*
  select *pobočka-jméno*
  from *půjčka*
  V syntaxi „čisté“ relační algebry se tento dotaz zapíše takto:
  $$\Pi_{pobočka\text{-}jméno}(půjčka)$$
- Hvězdička značí „všechny atributy“
  select *
  from *půjčka*
- SQL umožňuje duplikáty v relacích i ve výsledcích dotazů.
- Pro eliminaci duplikátů vložte klíčové slovo **distinct** za **select**.
- Najděte jména všech poboček v relaci *půjčka* a odstraňte duplikáty
  select distinct *pobočka-jméno*
  from *půjčka*

- Klíčové slovo **all** specifikuje, že duplikáty odstraněny nebudou.
  select all *pobočka-jméno*
  from *půjčka*
- Klauzule **select** může obsahovat aritmetické výrazy s operátory +, -, * a / na konstantách nebo atributech n-tic.
- Dotaz
  select *pobočka-jméno, půjčka-číslo, částka* * 100
  from *půjčka*
  vrátí relaci shodnou s relací *půjčka*, jen atribut *částka* je vynásoben číslem 100

Klauzule where

- Klauzule **where** odpovídá operaci výběr v relační algebře. Skládá se z predikátu zahrnujícího atributy relací, které jsou v klauzuli **from**.
- Najděte všechna čísla půjček pro půjčky v pobočce Perryridge s částkami většími než $1200.
  select *půjčka-číslo*
  from *půjčka*
  where *pobočka-jméno* = „Perryridge“ **and** *částka* > 1200
- SQL používá logické spojky **and**, **or** a **not**. Umožňuje tak použití aritmetických výrazů jako operandů pro operátory porovnání.
- SQL zahrnuje operátor porovnání **between** pro zjednodušení klauzule **where**, který specifikuje hodnotu z určitého intervalu.
- Najděte čísla půjček s částkami mezi $90,000 a $100,000 (tj. ≥$90,000 a ≤$100,000)
  select *půjčka-číslo*
  from *půjčka*
  where *částka* **between** 90000 **and** 100000

Klauzule from

- Klauzule **from** odpovídá kartézskému součinu z relační algebry. Vypíše relace, které mají být procházeny při vyhodnocování výrazu.
- Najděte kartézský součin *půjčovatel* × *půjčka*
  select *
  from *půjčovatel, půjčka*
- Najděte jméno a číslo půjčky všech zákazníků, kteří mají půjčku v pobočce Perryridge.
  select distinct *zákazník-jméno, půjčovatel.půjčka-číslo*
  from *půjčovatel, půjčka*
  where *půjčovatel.půjčka*-číslo = *půjčka.půjčka-číslo* **and** pobočka-*jméno* = „Perryridge“

Operace přejmenování

- Mechanismus pro přejmenovávání relací je v SQL řešen pomocí klauzule **as**:
  staré-jméno **as** *nové-jméno*
- Najděte jméno a číslo půjčky všech zákazníků, kteří mají půjčku v pobočce Perryridge; nahraďte jméno sloupce *půjčka-číslo* jménem *půjčka-id*.
  select distinct *zákazník-jméno, půjčovatel.půjčka číslo* **as** *půjčka-id*
  from *půjčovatel, půjčka*
  where *půjčovatel.půjčka-číslo = půjčka.půjčka-číslo*
  and *pobočka-jméno* = „Perryridge“

Proměnné n-tic (Tuple variables)

- Proměnné n-tic jsou definovány v klauzuli **from** pomocí klauzule **as**.
- Najděte jména zákazníků a čísla jejich půjček pro všechny zákazníky, kteří mají půjčku ve stejné pobočce.
  select distinct *zákazník-jméno, T.půjčka-číslo*
  from *půjčovatel* **as** *T*, *půjčka* **as** *S*
  where *T.půjčka-číslo = S.půjčka-číslo*
- Najděte jména všech poboček, které mají větší aktiva než nějaká pobočka v Brooklynu.
  select distinct *T.pobočka-jméno*
  from *pobočka* **as** *T*, *pobočka* **as** *S*
  where *T.aktiva > S.aktiva* **and** *S.pobočka-jméno* = „Brooklyn“

Operace s řetězci

- SQL zahrnuje operátor pro porovnávání řetězců znaků. Vzorky jsou popsány použitím dvou speciálních znaků:
  - procento (%). Znak % vyhovuje jakémukoliv podřetězci.
  - podtržítko (_). Znak _ vyhovuje jakémukoliv znaku.
- Najděte jména všech zákazníků, jejichž ulice obsahuje podřetězec ‚Main‘.
  select *zákazník-jméno*
  from *zákazník*
  where *zákazník-ulice* **like** „%Main%“
- Co vyhovuje jménu „Main%“?
  like „Main\%“ **escape** „\“

Uspořádání zobrazení n-tic

- Výpis všech zákazníků, kteří mají půjčku v pobočce Perryridge seřazený podle abecedy
  select distinct *zákazník-jméno*
  from *půjčovatel, půjčka*
  where *půjčovatel.půjčka-číslo = půjčka.půjčka-číslo* **and**
  pobočka-jméno = „Perryridge“
  order by *zákazník-jméno*
- Můžeme ještě specifikovat **desc** pro sestupné pořadí nebo **asc** pro vzestupné pořadí; vzestupné pořadí je implicitní.
- SQL musí provést třídění, aby vyhověl požadavku **order by**. Protože třídění velkého množství n-tic může být drahé, je vhodné třídit pouze v nezbytných případech.

Duplikáty

- V relaci s duplikáty může SQL definovat, kolik kopií n-tic se objeví ve výsledku.
- *Více-množinové* verze některých operátorů relační algebry – dané více-množinové relace r_1 a r_2:
  - Je-li c_1 kopií n-tice t_1 v r_1 a t_1 vyhovuje výběru σ_θ, pak je c_1 kopií n-tice t_1 v $\sigma_\theta(r_1)$
  - Pro každou kopii n-tice t_1 v r_1 je kopie n-tice $\Pi_A(t_1)$ v $\Pi_A(r_1)$, kde $\Pi_A(t_1)$ značí projekci jednoduché n-tice t_1.
  - Je-li c_1 kopií n-tice t_1 v r_1 a c_2 kopií n-tice t_2 v r_2, pak je $c_1 \times c_2$ kopií n-tice $t_1.t_2$ v $r_1 \times r_2$.
- Předpokládejme, že relace r_1 se schématem (*A, B*) a r_2 se schématem © jsou následující více-množiny:
  $r_1 = \{(1, a), (2, a)\}$ $r_2 = \{(2), (3), (3)\}$
- Pak $\Pi_B(r_1)$ bude $\{(a), (a)\}$, zatímco $\Pi_B(r_1) \times r_2$ bude
  $\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$
- SQL sémantika duplikátů:
  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P
  je ekvivalentní *více-množinové* verzi výrazu:
  $\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$

Množinové operace

- Množinové operace **union**, **intersect** a **except** operují na relacích a odpovídají operacím relační algebry $\cup$, $\cap$ a –.
- Každá z těchto operací automaticky eliminuje duplikáty; chceme-li i duplikáty, použijeme více-množinové verze **union all**, **intersect all** a **except all**. Předpokládejme, že se n-tice objeví *m*-krát v *r* a *n*-krát v *s*, pak se objeví:
  - *(m + n)*-krát v *r* **union all** *s*
  - *min(m, n)*-krát v *r* **intersect all** *s*
  - *max(0, m – n)*-krát v *r* **except all** *s*

- Najděte všechny zákazníky, kteří mají půjčku, účet nebo obojí:
  (**select** *zákazník-jméno* **from** *vkladatel*)
  union
  (**select** *zákazník-jméno* **from** *půjčovatel*)
- Najděte všechny zákazníky, kteří mají půjčku i účet:
  (**select** *zákazník-jméno* **from** *vkladatel*)
  intersect
  (**select** *zákazník-jméno* **from** *půjčovatel*)
- Najděte všechny zákazníky, kteří mají účet, ale ne půjčku:
  (**select** *zákazník-jméno* **from** *vkladatel*)
  except
  (**select** *zákazník-jméno* **from** *půjčovatel*)

Souhrnné funkce

Tyto funkce operují na více-množinových hodnotách sloupců relace a vracejí hodnotu
avg: průměrná hodnota
min: minimální hodnota
max: maximální hodnota
sum: suma hodnot
count: počet hodnot

- Najděte průměrnou hodnotu zůstatku účtu v pobočce Perryridge.
  select avg (*zůstatek*)
  from *účet*
  where *pobočka-jméno* = „Perryridge“
- Najděte počet n-tic v relaci *zákazník*.
  select count (*)
  from *zákazník*
- Najděte počet vkladatelů v bance.
  select count (**distinct** *zákazník-jméno*)
  from *vkladatel*

Souhrnné funkce – Group by

- Najděte počet vkladatelů v každé pobočce.
  select *pobočka-jméno*, **count** (**distinct** *zákazník-jméno*)
  from *vkladatel, účet*
  where *vkladatel.číslo-účtu* = *účet.číslo-účtu*
  group by *pobočka-jméno*

  Poznámka: Atributy v klauzuli **select** mimo souhrnné funkce se musí objevit v seznamu **group by**.

Souhrnné funkce – klauzule having

- Najděte jména všech poboček, jejichž průměrný zůstatek účtu je větší než $1,200.
  select *pobočka-jméno*, **avg** (*zůstatek*)
  from *účet*
  group by *pobočka-jméno*
  having avg (*zůstatek*) > 1200

  Poznámka: predikáty v klauzuli **having** jsou po zformování skupin

Nulové hodnoty

- N-tice můžou mít nulovou hodnotu, značenou *null*, ta znamená neznámou hodnotu nebo hodnotu, která neexistuje.
- Výsledek jakékoliv aritmetické operace zahrnující *null* je *null*.
- Všechna porovnávání zahrnující *null* vrací *false*. Precizněji:
  - Jakékoliv porovnání s *null* vrací *unknown*
  - (*true* or *unknown*) = *true*, (*false* or *unknown*) = *unknown*, (*unknown* or *unknown*) = *unknown*, (*true* and *unknown*) = *unknown*, (*false* and *unknown*) = *false*, (*unknown* and *unknown*) = *unknown*
  - Výsledek klauzule **where** je brán jako *false*, je-li *unknown*
  - „*P* **is unknown**“ se vyhodnotí jako *true*, jestliže predikát *P* je vyhodnocen jako *unknown*
- Najděte všechna čísla půjček, které se objeví v relaci *půjčka* s nulovou hodnotou *částka*.
  select *půjčka-číslo*
  from *půjčka*
  where *částka* **is null**
- Součet všech částek půjček
  select sum (*částka*)
  from *půjčka*

  Tyto výrazy ignorují nulové částky; výsledek je nula, jestliže zde není žádná nenulová částka.
- Všechny souhrnné operace s výjimkou **count(*)** ignorují n-tice s nulovými hodnotami na souhrnných atributech.

Vnořené poddotazy (Nested subqueries)

- SQL poskytuje mechanismus pro vnořování poddotazů.
- Poddotaz je výraz **select-from-where**, který je vnořen do jiného dotazu.
- Obvyklé použití poddotazů je provádění testů na členství v množině, porovnávání množin a kardinalitu množin.

Členství v množině

- F **in** $r \Leftrightarrow \exists\, t \in r\ (t = \mathrm{F})$

(5 **in** {0, 4, 5}) = true

(5 **in** {0, 4, 6}) = false

(5 **not in** {0, 4, 6}) = true

Příklady dotazů

- Najděte všechny zákazníky, kteří mají v bance účet i půjčku.

```
select distinct zákazník-jméno
from půjčovatel
where zákazník-jméno in (select zákazník-jméno
                         from vkladatel)
```

- Najděte všechny zákazníky, kteří mají v bance půjčku, ale ne účet.

```
select distinct zákazník-jméno
from půjčovatel
where zákazník-jméno not in (select zákazník-jméno
                             from vkladatel)
```

- Najděte všechny zákazníky, kteří mají účet i půjčku v pobočce Perryridge.

```
select distinct zákazník-jméno
from půjčovatel, půjčka
where půjčovatel.půjčka-číslo = půjčka.půjčka-číslo and
      pobočka-jméno = „Perryridge“ and
      (pobočka-jméno, zákazník-jméno) in
          (select pobočka-jméno, zákazník-jméno
          from vkladatel, účet
          where vkladatel.číslo-účtu = účet.číslo-účtu)
```

Porovnávání množin

- Najděte všechny pobočky, které mají větší aktiva než nějaká pobočka v Brooklynu.

```
select distinct T.pobočka-jméno
from pobočka as T, pobočka as S
where T.aktiva > S.aktiva and S.pobočka-město = „Brooklyn“
```


Klauzule some

- F <comp> **some** $r \Leftrightarrow \exists\, t\ (t \in r \wedge$ [F <comp> t])

kde <comp> může být: <, ≤, >, ≥, =, ≠

(5 < **some** {0, 5, 6}) = true (čteme: 5 je menší než nějaká n-tice z relace

(5 < **some** {0, 5}) = false

(5 = **some** {0, 5}) = true

(5 ≠ **some** {0, 5}) = true (neboť 0 ≠ 5)

- (= **some**) ≡ **in**
- Ale: (≠ **some**) ≠ **not in**

Příklad dotazu

- Najděte pobočky, které mají větší aktiva než nějaká pobočka v Brooklynu.

```
select pobočka-jméno
from pobočka
where aktiva > some
      (select aktiva
      from pobočka
      where pobočka-město = „Brooklyn“)
```

Klauzule all

- F <comp> **all** $r \Leftrightarrow \forall\, t\ (t \in r \wedge$ [F <comp> t])

(5 < **all** {0, 5, 6}) = false

(5 < **all** {6, 10}) = true

(5 = **all** {4, 5}) = false

(5 ≠ **all** {4, 6}) = true (neboť 5 ≠ 4 **and** 5 ≠ 6)

- (≠ **all**) ≡ **not in**
- Ale: (= **all**) ≠ **in**

Příklad dotazu

- Najděte pobočky, které mají větší aktiva než všechny pobočky v Brooklynu.

```
select pobočka-jméno
from pobočka
where aktiva > all
        (select aktiva
        from pobočka
        where pobočka-město = „Brooklyn“)
```

Test na prázdné relace

- Konstrukce **exists** vrací hodnotu **true**, je-li poddotaz v argumentu neprázdný.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

Příklad dotazu

- Najděte všechny zákazníky, kteří mají účet ve všech pobočkách v Brooklynu.

```
select dictinct S.zákazník-jméno
from vkladatel as S
where not exists (
        (select pobočka-jméno
        from pobočka
        where pobočka-město = „Brooklyn“)
        except
        (select R.pobočka-jméno
        from vkladatel as T, účet as R
        where T.číslo-účtu = R.číslo-účtu and
                S.zákazník-jméno = T.zákazník-jméno))
```

- Poznámka: $X - Y = \emptyset \Leftrightarrow X \subseteq Y$

Test na nepřítomnost duplikátních n-tic

- Konstrukce **unique** testuje, zda poddotaz má ve svém výsledku nějaké duplikátní n-tice.
- Najděte všechny zákazníky, kteří mají pouze jeden účet v pobočce Perryridge.

```
select T.zákazník-jméno
from vkladatel as T
where unique (
        select R.zákazník-jméno
        from účet, vkladatel as R
        where T.zákazník-jméno = R.zákazník-jméno and
                R.číslo-účtu = účet.číslo-účtu and
                účet.pobočka-jméno = „Perryridge“)
```


Příklad dotazu

- Najděte všechny zákazníky, kteří mají alespoň dva účty v pobočce Perryridge.

```
select T.zákazník-jméno
from vkladatel as T
where not unique (
        select R.zákazník-jméno
        from účet, vkladatel as R
        where T.zákazník-jméno = R.zákazník-jméno and
                R.číslo-účtu = účet.číslo-účtu and
                účet.pobočka-jméno = „Perryridge“)
```

Odvozené relace

- Najděte průměrný zůstatek účtu v těch pobočkách, kde je průměrný zůstatek větší než $1200.

```
select pobočka-jméno, prům-zůstatek
from (select pobočka-jméno, avg (zůstatek)
        from účet
        group by pobočka-jméno)
        as result (pobočka-jméno, prům-zůstatek)
where prům-zůstatek > 1200
```

Poznámka: nepotřebujeme použít klausuli **having**, jestliže v klauzuli **from** spočítáme dočasnou relaci *result* a její atributy můžeme použít přímo v klauzuli **where**.

Pohledy

- Poskytují mechanismus, jak skrýt nějaká data před nějakými uživateli. Pro vytvoření pohledu použijeme příkaz:

create view *v* **as** <výraz dotazu>

kde:
- <výraz dotazu> je jakýkoliv dovolený výraz
- *v* reprezentuje jméno pohledu

Příklady dotazů

- Pohled skládající se z poboček a jejich zákazníků

```
create view všichni-zákazníci as
(select pobočka-jméno, zákazník-jméno
from vkladatel, účet
where vkladatel.číslo-účtu = účet.číslo-účtu)
union
(select pobočka-jméno, zákazník-jméno
from půjčovatel, účet
where půjčovatel.půjčka-číslo = půjčka.půjčka-číslo)
```

Modifikace databáze – mazání

- Smažte všechny záznamy o účtech z pobočky Perryridge
  delete from *účet*
  where *pobočka-jméno* = „Perryridge“
- Smažte všechny účty z každé pobočky v Needhamu
  delete from *účet*
  where *pobočka-jméno* **in** (
  select *pobočka-jméno*
  from *pobočka*
  where *pobočka-město* = „Needham“)
  delete from *vkladatel*
  where *číslo-účtu* **in** (
  select *číslo-účtu*
  from *pobočka, účet*
  where *pobočka-město* = „Needham“
  and *pobočka.pobočka-jméno* = *účet.pobočka-jméno*)
- Smažte záznamy o všech účtech se zůstatkem pod průměrem banky
  delete from *účet*
  where *zůstatek* < (
  select avg (*zůstatek*)
  from *účet*)
  – Problém: když smažeme n-tice z *vklad*, průměrný zůstatek se změní
  – Řešení použité v SQL:
  * Nejprve spočítáme průměrný zůstatek a najdeme všechny n-tice ke smazání
  * Potom smažeme všechny n-tice nalezené výše

Modifikace databáze – vkládání

- Vložení nové n-tice do relace *účet*
  insert into *účet*
  values („Perryridge“, A-9732, 1200)
  nebo ekvivalentně
  insert into *účet* (*pobočka-jméno, zůstatek, číslo-účtu*)
  values („Perryridge“, 1200, A-9732)
- Přidání nové n-tice do relace *účet* se *zůstatkem* nastaveným na nulu.
  insert into *účet*
  values („Perryridge“, A-9732, *null*)

- Jako dárek poskytněte všem zákazníkům půjček v pobočce Perryridge $200 termínovaný účet. Nechť číslo půjčky slouží jako číslo účtu pro nový termínovaný účet.
  insert into *účet*
  select *pobočka-jméno, půjčka-číslo,* 200
  from *půjčka*
  where *pobočka-jméno* = „Perryridge“
  insert into *vkladatel*
  select *zákazník-jméno, půjčka-číslo*
  from *půjčka, půjčovatel*
  where *pobočka-jméno* = „Perryridge“
  and *půjčka.číslo-účtu* = *půjčovatel.číslo-účtu*

Modifikace databáze – aktualizace

- Všechny účty se zůstatkem přes $10,000 o 6 %, ostatní o 5 %.
  – Napíšeme dva výrazy **update**:
  update *účet*
  set *zůstatek* = *zůstatek* * 1.06
  where *zůstatek* > 10000

  update *účet*
  set *zůstatek* = *zůstatek* * 1.05
  where *zůstatek* ≤ 10000
  – pořadí je důležité
  – lépe lze vyřešit pomocí výrazu **case**

Aktualizace pohledu

- Vytvoříme pohled na všechna data v relaci *půjčka*, zakryjte atribut *částka*
  create view *pobočka-půjčka* **as**
  select *pobočka-jméno, půjčka-číslo*
  from *půjčka*
- Přidáme novou n-tici do pohledu
  insert into *pobočka-půjčka*
  values („Perryridge“, „L-307“)

Toto vkládání musí být reprezentováno vložením n-tice („Perryridge“, „L-307“, *null*) do relace *půjčka*.

- Aktualizaci komplexnějších pohledů je náročné nebo nemožné přeložit a proto nejsou povoleny.

Spojené relace (Joined relations)

- Operace spojení vezme dvě relace a jako výsledek vrátí jednu relaci.
- Tyto operace jsou typicky používány jako poddotazy v kaluzuli **from**.
- Podmínka spojení (Join condition) – definuje, které n-tice ve dvou relacích si odpovídají a které atributy jsou ve výsledku spojení.
- Typ spojení (Join type) – definuje kolik je n-tic v každé relaci, které nedpovídají žádné n-tici v jiné relaci.

Typy spojení
vnitřní spojení
levé vnější spojení
pravé vnější spojení
plné vnější spojení

Podmínky spojení
přirozené
na <predikát>
použitím (A_1, A_2, …, A_n)

Příklady

- Relace *půjčka*

pobočka-jméno	*půjčka-číslo*	*částka*
Downtown	L-170	3000
Redwood	L-230	4000
Perryridge	L-260	1700

- Relace *půjčovatel*

zákazník-jméno	*půjčka-číslo*
Jones	L-170
Smith	L-230
Hayes	L-155

- *půjčka* **inner join** *půjčovatel* **on** *půjčka.půjčka-číslo = půjčovatel.půjčka-číslo*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*	*půjčka-číslo*
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230

- *půjčka* **left outer join** *půjčovatel* **on** *půjčka.půjčka-číslo = půjčovatel.půjčka-číslo*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*	*půjčka-číslo*
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230
Perryridge	L-260	1700	*null*	*null*

- *půjčka* **natural join** *půjčovatel*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith

- *půjčka* **natural right outer join** *půjčovatel*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
null	L-155	*null*	Hayes

- *půjčka* **full outer join** *půjčovatel* **using** (*půjčka-číslo*)

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
Perryridge	L-260	1700	*null*
null	L-155	*null*	Hayes

- Najděte všechny zákazníky, kteří mají v bance buď účet nebo půjčku (ale ne obojí).

select *zákazník-jméno*
from (*vkladatel* **natural full outer join** *půjčovatel*)
where *číslo-účtu* **is** *null* **or** *půjčka-číslo* **is** *null*

Jazyk definice dat (Data definition language; DDL)

Umožňuje specifikaci nejen množin atributů, ale také informaci o každé relaci, zahrnující:

- Schéma každé relace.
- Doménu hodnot spojenou s každým atributem.
- Omezení integrity.
- Množinu indexů, které budou udržovány pro každou relaci.
- Bezpečnostní a autorizační informace o každé relaci.
- Fyzickou strukturu uložení na disk pro každou relaci.

Doménové typy v SQL

- **char(n)**. Řetězec znaků s pevnou délkou *n*.
- **varchar(n)**. Řetězec znaků s proměnlivou délkou (maximálně *n*).
- **int**. Celé číslo; závislé na implementaci.
- **smallint**. Krátké celé číslo; závislé na implementaci.
- **numeric(p,d)**. Číslo s pevnou desetinnou čárkou s přesností na *p* míst s *d* místy vpravo od desetinné tečky.
- **real, double precision**. Čísla s plovoucí desetinnou čárkou; závislé na implementaci.
- **float(n)**. Číslo s plovoucí desetinnou čárkou s přesností nejméně na *n* míst.
- **date**. Datumy obsahující (4bitový) rok, měsíc a den.
- **time**. Čas v hodinách, minutách a sekundách.
  - Ve všech doménových typech jsou povoleny nulové hodnoty. Deklarování atributu **not null** je zakazuje.
  - V SQL-92 vytvoří konstrukce **create domain** uživatelské doménové typy
    create domain *osoba-jméno* **char**(20) **not null**

Konstrukce create table

- SQL relace je definována použitím příkazu **create table**:
  create table *r* (A_1 D_1, A_2 D_2, …, A_n D_n,
  <omezení-integrity$_1$>,
  …,
  <omezení-integrity$_k$>)
  - *r* je jméno relace
  - každé A_i je jméno atributu ve schématu relace *r*
  - D_i je datový typ hodnot v doméně atributu A_i
- Příklad:

```
create table pobočka
        (pobočka-jméno      char(15) not null,
        pobočka-město       char(30),
        aktiva              integer)
```

Omezení integrity v konstrukci create table

- **not null**
- **primary key** (A_1, …, A_n)
- **check** (*P*), kde *P* je predikát

Příklad: Deklarace *pobočka-jméno* jako primárního klíče pro *pobočka* a zajištění, že hodnota atributu *aktiva* bude nezáporná.

```
create table pobočka
        (pobočka-jméno      char(15) not null,
        pobočka-město       char(30),
        aktiva              integer,
        primary key (pobočka-jméno),
        check (aktiva >= 0))
```

- Deklarace atributu jako **primary key** zajišťuje v SQL-92 automaticky i **not null**.

Konstrukce drop table a alter table

- Příkaz **drop table** smaže všechny informace o dané relaci z databáze.
- Příkaz **alter table** přidá atributy k existující relaci. Ke všem n-ticím v relaci přiřadí *null* jako hodnotu po nové atributy. Tvar příkazu je následující
  alter table *r* **add** *A D*
  kde *A* je jméno atributu, který je přidán do relace *r* a *D* je jeho doména.
- Příkaz **alter table** je často používán též k odstranění (drop) atributů z relace:
  alter table *r* **drop** *A*

kde *A* je jméno atributu relace *r*.

Zabudovaný SQL

- Standard SQL definuje zabudování SQL do řady programovacích jazyků jako jsou Pascal, PL/I, Fortran, C a Cobol.
- Jazyk, ve kterém jsou zabudovány SQL dotazy, je nazýván *hostitelský* jazyk a SQL struktury povolené v hostitelském jazyce vytvářejí *zabudovaný* SQL.
- Výraz EXEC SQL je používán pro identifikaci vloženého SQL dotazu pro preprocesor
  EXEC SQL <vložený SQL výraz> END EXEC

Příklad dotazu

Z hostitelského jazyka najděte jména a čísla účtu s více než *částka* dolary na nějakém účtu.

- Specifikujte dotaz v SQL a deklarujete pro ni *kurzor*

```
EXEC SQL
    declare c cursor for
    select zákazník-jméno, číslo-účtu
    from vkladatel, účet
    where vkladatel.číslo-účtu = účet.číslo-účtu
        and účet.zůstatek > :částka
END-EXEC
```

Zabudovaný SQL (pokračování)

- Výraz **open** způsobí vyhodnocení dotazu
  EXEC SQL **open** *c* END-EXEC
- Výraz **fetch** způsobí, že hodnoty jedné n-tice budou vloženy do proměnných hostitelského jazyka.
  EXEC SQL **fetch** *c* **into** :*cn* :*an* END-EXEC
- Výraz **close** zavře databázový systém, takže se smaže dočasná relace, která obsahuje výsledek dotazu
  EXEC SQL **close** *c* END-EXEC

Dynamický SQL

- Umožňuje programům konstruovat SQL dotazy za běhu.
- Příklad použití dynamického SQL z programu v jazyce C.

```
char *sqlprog = „update účet set zůstatek = zůstatek * 1.05
                where číslo-účtu = ?“
EXEC SQL prepare dynprog from :sqlprog;
char účet[10] = „A-101“;
EXEC SQL execute dynprog using :účet;
```

- Dynamický SQL program obsahuje znak ?, který je místem pro hodnotu, která je poskytována před spuštěním SQL programu.

Další rysy SQL

- *Jazyky čtvrté generace* –speciální jazyky asistující aplikačním programátorům při tvorbě uživatelského rozhraní a formátování výstupu
- SQL sezení (SQL session) – poskytuje abstrakci klienta a serveru
  - klient se *připojí* k SQL serveru, zahájí sezení
  - spustí sérii výrazů
  - *odpojí* sezení
  - může odevzdat nebo vrátit práci uskutečněnou v sezení
- SQL prostředí obsahuje několik komponent (identifikátor uživatele a schéma, které identifikuje, které z několika schémat sezení používá).