

Navrhovanie databáz

Ján Mazák

FMFI UK Bratislava

Návrh databázových schém

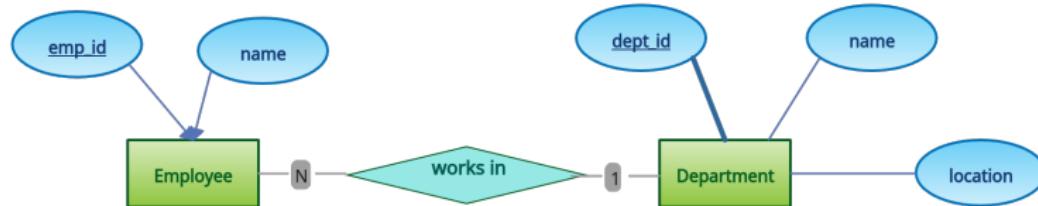
Databázová schéma: čo sú atribúty a aký majú dátový typ, aké relácie máme, vzťahy medzi nimi.

Pri návrhu databázovej schémy (voľne databázy) sa najmä snažíme vyhnúť známym nedostatkom. Neraz ide o kompromis medzi blízkosťou k ideálnemu návrhu a výkonom v praxi.

Relačný model, jazyk SQL a bežné DBMS poskytujú dostatok prostriedkov na zabezpečenie integrity dát.

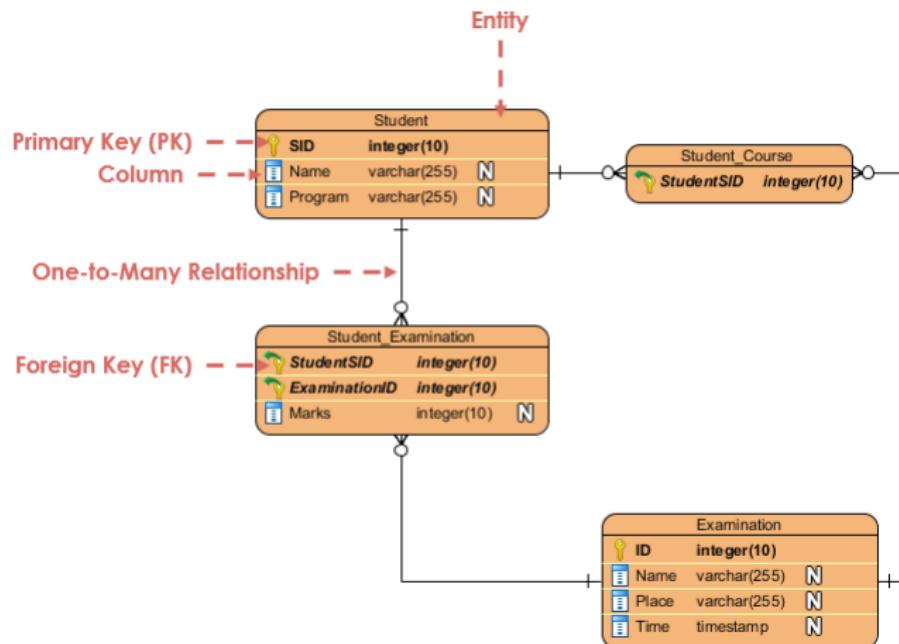
Entitno-relačný model (ERM)

Pri návrhu vychádzame zo zjednodušeného modelu sveta — *dátový model*. Tieto modely sa bežne kreslia v podobe **entitno-relačných diagramov** (ERD).



ERM možno vytvárať na rôznej úrovni podľa množstva detailov: konceptuálna, logická, fyzická.

Entitno-relačný model



Oproti predošlému obrázku pribudli dátové typy, sme bližšie k finálnej databázovej schéme.

ERM: konceptuálna úroveň (Conceptual Level)

- ▶ Identifikuje hlavné entity (napr. Čitateľ, Kniha).
- ▶ Definuje základné vzťahy medzi entitami (napr. *požičiava si*).
- ▶ Neobsahuje atribúty, dátové typy ani kl'úče.
- ▶ Slúži na (úvodnú) komunikáciu s „biznis používateľmi“ (manažment, klient).

ERM: logická úroveň (Logical Level)

Detailný návrh štruktúry dát nezávislý od DBMS.

- ▶ Entity sa stávajú tabuľkami.
- ▶ Definuje všetky atribúty (stĺpce) pre každú tabuľku.
- ▶ Určuje primárne klúče (PK) a cudzie klúče (FK).
- ▶ Špecifikuje kardinalitu vzťahov (1:N, M:N).
- ▶ Stále nerieši špecifické dátové typy (napr. VARCHAR vs TEXT).

ERM: fyzická úroveň (Physical Level)

Konkrétna implementácia pre špecifický databázový systém (DBMS).

- ▶ Presné názvy tabuľiek a stĺpcov.
- ▶ Definuje špecifické dátové typy (napr. TEXT, DATE).
- ▶ Zahŕňa obmedzenia (constraints) ako NOT NULL, UNIQUE.
- ▶ Rieši optimalizáciu výkonu: indexy, partície.
- ▶ Výsledkom je DDL skript (napr. CREATE TABLE...).

Entitno-relačný model

Reprezentujeme situáciu na pracovisku. Pre zamestnancov evidujeme meno, mzdu, oddelenie, na ktorom pracujú (len jedno), adresu tohto oddelenia, a zoznam projektov, na ktorých pracujú. Pre projekty potrebujeme vedieť názov, rozpočet, zoznam ich zamestnancov a vedúceho projektu.
Nakreslite entitno-relačný diagram:

- ▶ Entity sú obdĺžniky, hlavička je názov entity a pod ňou je zoznam atribútov s dátovým typom oddeleným dvojbodkou.
- ▶ Vzťah medzi entitami zobrazíme neorientovanou hranou medzi obdĺžnikmi, ku ktorému pripíšeme, či ide o one-to-one (1:1), one-to-many (1:N), many-to-many (M:N) (pozor na poradie; označenia 1, M, N sa bežne pripisujú ku koncom hrán).

Entitno-relačný model v SQL

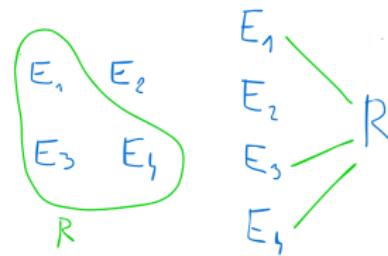
Prepis ERM do databázovej schémy v SQL:

- ▶ Entite zodpovedá tabuľka; každý jej atribút je stĺpec tejto tabuľky.
- ▶ Vzťahy 1:N — pridáme stĺpec do príslušnej tabuľky, napr. číslo oddelenia v tabuľke zamestnancov.
- ▶ Vzťahy 1:1 — tiež pridáme stĺpec (možno si vybrať kam).
- ▶ Vzťahy M:N — zvyčajne cez extra *tabuľku* s dvomi stĺpcami pre dvojice entít; atribút, ktorého dátovým typom je pole, je takmer vždy v rozpore s dobrou praxou.

Vzťah M:N možno „povýsiť“ na samostatnú entitu, vzniknú tak dva nové vzťahy 1:N. To umožňuje pridať ku vzťahu doplňujúce údaje, napr. pre zapísané predmety študentov doplníme semester štúdia a čas zápisu. Tieto „vzťahové“ entity neraz majú podivné názvy (napr. emp_project_assignment).

ERM: ternárne vzťahy (pre 3 entity)

V ERM možno reprezentovať aj vzťahy viac ako 2 entít. Stačí z ternárneho vzťahu R spraviť samostatnú entitu a pridať jej väzby na entity, ktorých sa vzťah týka.



Pozrite si príklad: (link)

ERM: ternárne vzťahy (pre 3 entity)

Všeobecne: hypergraf vieme reprezentovať bipartitným grafom. Tento postup je však dosť neprirodzený a viacnásobné vzťahy je jednoduchšie priamo reprezentovať v databázovom relačnom modeli.

Pre ternárny vzťah vieme rovno pridať reláciu a stačí správne popísat násobnosť priradenia (napr. že pre každú dvojicu AB existuje najviac jedna hodnota C tak, že ABC sú v relácii).

Terminológia: čo je to **atribút**?

- ▶ V „matematickom“ databázovom relačnom modeli je to **„atomický“ element**, nad ktorým vytvárame relácie (ako množiny atribútov).
- ▶ V SQL (čo je praktická implementácia „matematického“ modelu) je to **stĺpec tabuľky**; stĺpce jednotlivých tabuliek sú však nezávislé a na priblíženie sa matematickému modelu slúžia cudzie klúče.
- ▶ V ERM je to **vlastnosť entity** (a nesúvisí so vzťahmi medzi entitami, hoci jedno aj druhé musíme v SQL reprezentovať stĺpcami).

ERM: vzťah vs. atribút entity

Neraz si vyberáme medzi **vzťahom medzi entitami** a **atribútom entity**. Príklady:

- ▶ ulica — relácia s fixným zoznamom existujúcich ulíc a odkazy na ňu / textový atribút bez väzby na čokoľvek iné
- ▶ rodinný stav — tabuľka s „vopred daným“ zoznamom stavov / textový atribút, kde si každý vyplní, čo chce (resp. obmedzíme cez CHECK)

ERM: vzťah vs. atribút entity

Výhody vytvorenia samostatnej entity:

- ▶ Existuje jednoznačný zoznam prípustných hodnôt, využiteľný v UI a klientskych programoch.
- ▶ Spoľahlivé vyhľadávanie (nemusíme premýšľať, či niekto nepoužil iný reťazec na popis tej istej veci).
- ▶ Databáza automaticky a transparentne stráži konzistentnosť (napr. preklepy); možno „pekné“ reportovať chyby (na rozdiel od CHECK).
- ▶ Rozšíriteľnosť: nové tabuľky sa môžu cez cudzí klúč odkazovať na entity; nemusíme zdvojovať CHECK a strážiť, že je identický.

ERM: vzťah vs. atribút entity

Nevýhody vytvorenia samostatnej entity:

- ▶ Možno príliš obmedzujúce, napr. fixný zoznam krstných mien či prípustných hodnôt pre položku „hobby“.
- ▶ Čažkosti so zmenami zoznamu povolených hodnôt: Kto bude mať oprávnenie meniť či pridávať hodnoty? Ako sa zmena možných hodnôt distribuuje ostatným, aby nepracovali s neplatným zoznamom?
- ▶ Potenciálne priveľa entít (napr. adresu možno rozbiť na entity ulica, číslo domu, mesto, ...) a zahltenie stovkami tabuliek vedúce k ľudským chybám alebo nízkemu výkonu.

Redundancia

<i>Zamestnanec</i>	<i>Pracovisko</i>	<i>Adresa pracoviska</i>
Adam	SAV	Patrónka
Adam	KAGDM	Mlynská dolina
Cyril	KAGDM	Mlynská dolina

Redundancy: Informácia o adrese pracoviska je v tabuľke viacnásobne.

Redundancia — anomálie pri updatovaní

Zamestnanec	Pracovisko	Adresa pracoviska
Adam	SAV	Patrónka
Adam	KAGDM	Mlynská dolina → Staré grunty
Cyril	KAGDM	Mlynská dolina

UPDATE anomalies: keď upravíme adresu pracoviska v jednom zázname, bude to v rozporu s ostatnými záznamami.

Redundancia — anomálie pri mazaní

Zamestnanec	Pracovisko	Adresa pracoviska
Adam	SAV	Patrónka
Adam	KAGDM	Mlynská dolina
Cyril	KAGDM	Mlynská dolina

DELETE anomaly: keď zmažeme všetkých zamestnancov pracoviska, stratíme informáciu o jeho adrese.

Redundancia — anomálie pri vkladaní

Zamestnanec	Pracovisko	Adresa pracoviska
Adam	SAV	Patrónka
Adam	KAGDM	Mlynská dolina
Cyril	KAGDM	Mlynská dolina

INSERT anomaly: nemožno zmysluplnie pridať adresu pracoviska bez pridania zamestnancov (resp. NULL).

Redundancia — ako sa jej vyhnúť?

Riešenie: dekompozícia relácií.

Zamestnanec	Pracovisko	Pracovisko	Adresa prac.
Adam	SAV	SAV	Patrónka
Adam	KAGDM	KAGDM	Mlynská d.
Cyril	KAGDM		

- ▶ žiadna redundancia
- ▶ žiadne anomálie (INSERT, UPDATE, DELETE)

Naplnenie relácií v dekompozícii získame projekciou z pôvodnej relácie. Pôvodné dátá získame pomocou operácie JOIN.

Funkčné závislosti

Ako vieme, kedy je dekompozícia dostatočná?

Matematický model: funkčné závislosti.

Uvažujme reláciu R a dve množiny atribútov X, Y .

Funkčná závislosť $X \rightarrow Y$ platí práve vtedy,

keď pre každé naplnenie relácie R platí:

ak sa dva záznamy zhodujú na všetkých atribútoch z X ,
zhodujú sa aj na všetkých atribútoch z Y .

Príklad: $R = (\text{emp_id}, \text{emp_name}, \text{dept_id}, \text{dept_name})$;
FZ:

$\{\text{dept_id}\} \rightarrow \{\text{dept_name}\}$

$\{\text{emp_id}\} \rightarrow \{\text{emp_name}, \text{dept_id}\}$

Funkčné závislosti

Príklad: neplatí $X \rightarrow Y$, ale môže platiť $Y \rightarrow X$:

X	Y
1	2
1	3
2	4
3	5
3	6

(Pozor: platnosť FZ je cez všetky naplnenia tabuľky, čiže z jedného naplnenia nevieme rozhodnúť o platnosti FZ.)

Funkčné závislosti

Príklad: ktoré z možných netriviálnych FZ
 $(C \rightarrow B, AC \rightarrow B\dots)$ určite neplatia?

A	B	C
1	1	1
1	2	3
2	5	3
3	5	3

Funkčné závislosti

Pridanie funkčnej závislosti spraví z relácie funkciu. Táto funkcia je definovaná naplnením tabuľky (nie matematicky — to by sme pravú stranu FZ vôbec nemuseli evidovať v databáze, ak by sa dala vždy vypočítať z hodnoty naľavo).

Pre tabuľku s dvomi atribútmi x , y to presne zodpovedá matematickým pojmom: *binárna relácia* reprezentuje *funkciu*, ak pre každú hodnotu x máme jedinú hodnotu y . (Technicky, ide o čiastočnú funkciu, nemusí byť definovaná na všetkých prípustných hodnotách x , jej definičný obor je daný konkrétnym naplnením tabuľky.)

Funkčné závislosti

Aké FZ platia v relácii s atribútmi idFaktúry,
idObjednávky, idZákazníka, idProduktu, Cena, Množstvo?

Funkčné závislosti

Aké FZ platia v relácii s atribútmi idFaktúry,
idObjednávky, idZákazníka, idProduktu, Cena, Množstvo?

$$F \rightarrow O$$

$$O \rightarrow Z$$

$$OP \rightarrow M$$

$$P \rightarrow C \quad ??? \quad (\text{zľavy pre veľkoodberateľov?})$$

$$O \rightarrow F \quad ??? \quad (\text{čo s opravnými faktúrami?})$$

Ak chceme nájsť všetky platné FZ, môžeme postupne pre každý atribút A hľadať minimálne množiny X také, že $X \rightarrow A$.

Funkčné závislosti

FZ nemožno určiť z naplnenia databázy (pri inom by nemuseli platiť), musia byť preto súčasťou návrhu.

Funkčné závislosti treba vnímať so všetkými ich dôsledkami.
Predpokladajme, že platí

$$A \rightarrow C, \quad B \rightarrow D, \quad CD \rightarrow E.$$

Potom aj

$$AB \rightarrow D$$

$$A \rightarrow A$$

$$AB \rightarrow E$$

...

Funkčné závislosti

Čo znamenajú nasledujúce FZ?

$$A \rightarrow \emptyset$$

$$\emptyset \rightarrow A$$

Funkčné závislosti

Čo znamenajú nasledujúce FZ?

$$A \rightarrow \emptyset$$

$$\emptyset \rightarrow A$$

Prvá vždy triviálne platí (nehovorí nič), podľa druhej je A konštantá (hodnota A vo všetkých riadkoch musí byť rovnaká).

Funkčné závislosti

Pravidlá pre odvdzovanie — Armstrongove axiómy.

1. Ak $X \rightarrow Y$, tak $XZ \rightarrow YZ$.
2. Ak $Y \subseteq X$, tak $X \rightarrow Y$.
3. Ak $X \rightarrow Y$ a $Y \rightarrow Z$, tak $X \rightarrow Z$.

Ih plnosť vyplýva priamo z definície FZ.

Odvodiť sa dajú ďalšie pravidlá, napr.

$$A_1 A_2 \dots A_n \rightarrow B_1 B_2 \dots B_n \leftrightarrow \forall i \ A_1 A_2 \dots A_n \rightarrow B_i.$$

Množinu všetkých FZ, ktoré možno odvodiť z danej množiny FZ M , nazývame **uzáver množiny FZ M** .

(Jeho veľkosť môže byť exponenciálna vzhľadom na veľkosť M , čo spôsobuje algoritmické problémy.)

Funkčné závislosti

Uzáver množiny atribútov $\{A_1, A_2 \dots, A_n\}$ vzhľadom na nejakú množinu FZ M : maximálna množina atribútov B taká, že platí $A_1 A_2 \dots A_n \rightarrow B$.

Uzáver možno vypočítať v polynomiálnom čase tak, že začneme pôvodnou množinou atribútov, a v každom kroku prejdeme všetky FZ z M a pre každú z nich pridáme do uzáveru pravú stranu, ak sa tam už nachádza ľavá strana. Keď sme nič nepridali počas celého prechodu M , končíme.

Uzáver možno využiť na výpočet všetkých FZ platných v danej relácii (vyskúšame všetky možné podmnožiny atribútov ako ľavé strany a na pravú stranu pripíšeme uzáver ľavej strany).

Funkčné závislosti a klúče

Množina atribútov X je v nejakej relácii s atribútmi R nadklúč, ak platí $X \rightarrow R$. Nadklúč minimálny vzhľadom na inkluziu sa nazýva klúč. (V angličtine sa pre dvojicu pojmov nadklúč/klúč používa superkey/key alebo key/candidate key.)

Aké klúče má relácia
(emp_id, emp_name, dept_id, dept_name)?

Reprezentácia FZ v SQL: UNIQUE

Pre závislosť $AB \rightarrow C$ spravíme tabuľku

```
CREATE TABLE t (
    A <data_type_of_A>,
    B <data_type_of_B>,
    C <data_type_of_C>,
    UNIQUE(A, B)
);
```

Ked'že sa do nej nedajú vložiť záznamy (a_1, b_1, c_1) a (a_1, b_1, c_2) , platí aj požadovaná funkčná závislosť.

Keby sme mali v jednej tabuľke veľa FZ, ich vynucovanie pomocou UNIQUE môže navzájom kolidovať. To tiež naznačuje vhodnosť rozdelenia veľkej tabuľky na malé (dekompozícia relácie).

Dekompozícia relácií

V princípe každú databázu možno vnímať ako jedinú reláciu (predstavte si join všetkých tabuľiek). Preto dekompozícia relácie na menšie je pri návrhu kľúčová, aj keď ju explicitne nevnímame.

Pri dekompozícii **nechceme polámať funkčné závislosti**.

Príklad: relácia $order_items(Order, Product, Amount)$ s FZ

$$\{Order, Product\} \rightarrow \{Amount\}.$$

Ak ju akokoľvek dekomponujeme, stratíme informáciu o tom, že $\{Order, Product\}$ je kľúč. Pri kontrole integrity budeme musieť pri každej zmene niektornej z čiastkových relácií spočítať join a overiť, či nedošlo k porušeniu FZ.

Dekompozícia relácií — bezstratovosť

Z dekomponovanej relácie musíme vedieť získať pôvodné záznamy. Inak povedané, ak spravíme join, dostaneme práve pôvodné záznamy a **nič navyše** — nestratili sme žiadnu informáciu, čiže ide o **bezstratovú dekompozíciu (lossless decomposition)**.

Otestovať, či je dekompozícia bezstratová, možno polynomiálnym algoritmom (link: chase algorithm).

Jednoduchšie riešenie pre 2 relácie: požadujeme, aby množina spoločných atribútov bola v aspoň jednej z nich nadklúčom.

Dekompozícia relácií — bezstratovost'

$r(Zamestnanec, Pracovisko, AdresaPracoviska)$

dekomponujeme na

$r_1(Zamestnanec, Pracovisko),$

$r_2(Pracovisko, AdresaPracoviska).$

Je táto dekompozícia bezstratová?

Dekompozícia relácií — bezstratovosť

$r(Zamestnanec, Pracovisko, AdresaPracoviska)$

dekomponujeme na

$r_1(Zamestnanec, Pracovisko),$

$r_2(Pracovisko, AdresaPracoviska).$

Je táto dekompozícia bezstratová?

Označme $Z = Zamestnanec$, $P = Pracovisko$,

$A = AdresaPracoviska$.

$\forall r$ platí $Z \rightarrow P$, $P \rightarrow A$. Preto $P (= r_1 \cap r_2)$ je kľúč v r_2 a do (natural) joinu $r_1 \bowtie r_2$ sa nemôže dostať záznam, ktorý v r neboli, pretože tento join ku každému riadku r_1 len doplní hodnotu A z r_2 .

Dekompozícia relácií — bezstratovost'

Veta: Dekompozícia XYZ do XY a XZ je bezstratová práve vtedy, keď X je nadkl'účom v XY alebo v XZ .

Dekompozícia relácií — bezstratovosť

Veta: Dekompozícia XYZ do XY a XZ je bezstratová práve vtedy, keď X je nadklúčom v XY alebo v XZ .

Dôkaz: Ak X je nadklúčom v XY (čiže $X \rightarrow Y$), tak join $XY \bowtie XZ$ len pripíše hodnotu Y k záznamom v XZ , dekompozícia preto je bezstratová.

Predpokladajme, že neplatí $X \rightarrow Y$ ani $X \rightarrow Z$. Ak XYZ obsahuje xy_1z_1 a xy_2z_2 , join $XY \bowtie XZ$ bude obsahovať aj xy_1z_2 .

Dekompozícia relácií — bezstratovosť

Bezstratovosť je iná vlastnosť ako zachovanie funkčných závislostí. Uvažujme reláciu ABC s jedinou FZ $A \rightarrow B$, ktorú dekomponujeme do AB a BC .

FZ ostala zachovaná, ale AB a BC sa nespájajú bezstratovo, lebo spoločný atribút B nie je klúčom v AB ani v BC .

(Vhodnejšia dekompozícia je AB a AC . Overte, že je bezstratová.)

Dekompozícia relácií

Požiadavky na dekompozíciu:

- ▶ Je bezstratová. (Nutné.)
- ▶ Neláme funkčné závislosti. (Z tohto sa dá trocha zľaviť, ale je oveľa lepšie FZ zachovať, umožňuje to kontrolu integrity štandardnými prostriedkami.)
- ▶ Odstraňuje čo najviac redundancie a anomalií.

Redundancia sa odstraňuje **normalizáciou** — cieleným dekomponovaním. Mieru normalizácie popisujú **normálne formy**.

Normálne formy

- 1NF** Nemáme duplikáty ani kompozitné atribúty (hodnotou atribútu nie je relácia).
 - 3NF** Odstraňuje väčšinu redundancie a problémov s anomáliami. Vždy existuje a ľahko sa hľadá.
 - BCNF** Čosi ako 3.5NF, ideál. Relácia v BCNF neobsahuje žiadnu redundanciu vzhľadom na FZ, ani nedochádza k spomínaným anomáliám.
- Normálne formy vytvárajú hierarchiu (ak je niečo vo vyššej, je aj v nižšej). Existujú aj vyššie normálne formy (napr. 4NF), tie sa venujú odstraňovaniu tzv. multivalued dependencies. V praxi sú málo relevantné.

Relácia r je v BCNF, ak ľavá strana každej netriviálnej FZ je nadklúč r .

Problémy:

- ▶ Pre niektoré relácie neexistuje (napr. $r(ABC)$, kde FZ sú $AB \rightarrow C$, $C \rightarrow A$).
- ▶ Pre danú reláciu je NP-úplné rozhodnúť, či je v BCNF.

Ak relácia R nie je v BCNF kvôli FZ $X \rightarrow Y$, dekomponujeme ju do XY a $R - Y$. (Zachováva bezstratovosť.)
(V literatúre možno nájsť iné, hoci ekvivalentné, definície jednotlivých normálnych foriem. BCNF nemá pekné číslo: zámerom bolo už v 3NF odstrániť všetku redundanciu, ale vyšlo to až na druhý pokus.)

Relácia r je v 3NF, ak **ľavá strana každej netriviálnej FZ je nadklúč r** alebo každý atribút na pravej strane je súčasťou nejakého klúča.

Vlastnosti:

- ▶ Pre danú reláciu je NP-úplné rozhodnúť, či je v 3NF.
- ▶ Dekompozícia do 3NF vždy existuje.
- ▶ Existuje polynomiálny algoritmus, ktorý aspoň jednu 3NF nájde (cez minimálne pokrytie, pozri kap. 8, 9, 10). Nájdená dekompozícia je bezstratová a zachováva FZ.

Schéma vytvorená na základe dobrého entitno-relačného modelu je zväčša už v 3NF.

Príklad porušenia 3NF

Relácia s **tranzitívou** FZ: $A \rightarrow C$, ak $A \rightarrow B$ a $B \rightarrow C$ sú FZ.

Predpokladáme, že iné netriviálne FZ nemáme. Potom A je jediný kľúč: nie je na pravej strane žiadnej FZ, takže sa nedá odvodiť, preto je súčasťou každého kľúča; nič ďalšie v kľúči nemôže byť, lebo samotné A je kľúč.

Lavá strana $B \rightarrow C$ nie je nadkľúč, pravá strana nie je súčasťou žiadneho kľúča, preto relácia ABC nie je v 3NF.

Aké anomálie v ABC hrozia?

Normalizácia

Normalizujme reláciu R :

<i>Zamestnanec Z</i>	<i>Pracovisko P</i>	<i>Adresa pracoviska A</i>
Adam	KAGDM	Mlynská dolina
Cyril	KAGDM	Mlynská dolina

Normalizácia

Normalizujme reláciu R :

Zamestnanec Z	Pracovisko P	Adresa pracoviska A
Adam	KAGDM	Mlynská dolina
Cyril	KAGDM	Mlynská dolina

FZ: $Z \rightarrow P$, $P \rightarrow A$. Jediný kľúč je Z .

FZ $P \rightarrow A$ porušuje BCNF, lebo ľavá strana nie je nadkľúč.
Aj 3NF, lebo atribút A na pravej strane nie je súčasťou
žiadneho kľúča.

Preto dekomponujeme: do PA a $R - A = ZP$.

Tie sú binárne, preto v BCNF. FZ ostali zachované.

Normalizácia

Uvažujme reláciu R(Student, Course, Teacher). Vieme, že:

- ▶ Študent môže študovať viacero predmetov.
- ▶ V rámci jedného predmetu má študent jedného učiteľa.
- ▶ Predmet môže mať viac učiteľov,
avšak každý učiteľ učí len jeden predmet.

1. Nájdite všetky platné netriviálne FZ.
2. Nájdite všetky kľúče tejto relácie.
3. Zistite najvyššiu normálnu formu, v ktorej je táto relácia.
4. Bezstratovo ju dekomponujte do BCNF, ak treba.
Zachovajú sa FZ? Ak nie, ako kontrolovať tie porušené?

Normalizácia

V relácii s atribútmi idFaktúry, idObjednávky, idZákazníka, idProduktu, Cena, Množstvo platia FZ

$$F \rightarrow O$$

$$O \rightarrow Z$$

$$OP \rightarrow M$$

$$P \rightarrow C$$

1. Nájdite všetky kľúče tejto relácie.
2. Dekomponujte ju do 3NF, prípadne BCNF.

Normalizácia

V relácii s atribútmi idFaktúry, idObjednávky, idZákazníka, idProduktu, Cena, Množstvo platia FZ

$$F \rightarrow O$$

$$O \rightarrow Z$$

$$OP \rightarrow M$$

$$P \rightarrow C$$

1. Najdite všetky kľúče tejto relácie.
2. Dekomponujte ju do 3NF, prípadne BCNF.

Všimnite si, ako dekompozícia smeruje k jednej tabuľke pre každú FZ (ktorá sa ľahko vynúti v SQL pomocou UNIQUE).

Overenie NF

„Byť v normálnej forme“ nie je „existenčná“ vlastnosť: nestačí čosi malé nájsť a ukázať, že to má požadované vlastnosti; naopak treba zdôvodniť, že neexistuje závislosť, ktorá by NF pokazila.

Typická relácia (napr. zodpovedajúca entite v ERM) má primárny kľúč a od neho funkčne závislé atribúty. Z toho však nijako nevyplýva, že je v NF; treba skúmať všetky potenciálne platné závislosti medzi ostatnými atribútmi.

Postup pre 3NF a BCNF (jediné dve v praxi zaujímavé NF): podľa ich definícií stačí skúmať závislosti, ktoré majú na pravej strane práve jeden atribút.

Overenie NF

Zoberieme všetky podmnožiny atribútov relácie R ako potenciálne ľavé strany platnej FZ; pre každú podmnožinu L treba zistiť všetky možné pravé strany.

- ▶ Ak máme komplettnú množinu FZ, vypočítame uzáver L . Ak sa v uzávere vyskytne atribút, ktorý nie je na ľavej strane, porovnáme túto FZ s definíciou príslušnej NF (pri testovaní 3NF je vhodné poznať všetky kľúče R).
- ▶ Alebo postupujeme podľa definície FZ: pre $L \rightarrow A$ sa pýtame, či môžu pre danú hodnotu L existovať dve rôzne hodnoty A (odpovedať musí niekto, kto rozumie modelovanému svetu).

Overenie NF

Bežné chyby:

- ▶ Miesto uzáveru množiny všetkých FZ pracujeme len s FZ v konkrétnnej reprezentácii tohto uzáveru (napr. niektoré minimálne pokrytie). Napr. máme zapísané $A \rightarrow X$, $X \rightarrow B$, a pre reláciu obsahujúcu A aj B si nevšimneme, že v nej platí $A \rightarrow B$.
- ▶ Pri návrhu databázy zriedka pracujeme s reláciou, ktorá obsahuje všetky atribúty (hoci virtuálne taká vždy existuje, stačí si predstaviť join všetkých relácií v dekompozícii). Miesto toho máme čiastočnú dekompozíciu, z ktorej nemusí byť vidno vzťahy medzi jednotlivými jej časťami. Ak robíte dekompozíciu bez kompletnej množiny FZ, je vhodné začať do samostatných relácií oddelovať atribúty, ktoré s inými zjavne nesúvisia.

Normalizácia

Po normalizácii väčšina relácií zodpovedá jednotlivým funkčným závislostiam (napr. tabuľka s atribútmi O, P, M pre $OP \rightarrow M$), resp. skupinám FZ so spoločnou ľavou stranou, ktorá je kľúčom (tabuľka A, B, C pre $A \rightarrow B, A \rightarrow C$).

Preto na ich vynútenie postačí v SQL UNIQUE (hoci tento constraint ignoruje pravú stranu FZ).

Prehnávaná normalizácia vede k binárnym reláciám (tie sú vždy v BCNF), napr. binárne relácie (Id, Meno), (Id, Priezvisko), (Id, RodneCislo) miesto (Id, Meno, Priezvisko, RodneCislo).

Normalizácia núti používateľov písanie joins, ktoré naspäť spojajú tabuľky. Tie najbežnejšie či netriviálne možno napísanie vopred v podobe VIEW.

Príklad: databáza kníh

Atribúty:

- ▶ ISBN
- ▶ Názov
- ▶ Autor
- ▶ NárodnosťAutora
- ▶ Formát
- ▶ Cena (odporúčaná)
- ▶ Témy (zoznam tém)
- ▶ PočetStrán
- ▶ Vydavateľstvo
- ▶ KontaktVydavateľstvo
- ▶ Žáner
- ▶ DefiníciaŽánru

Predpoklady:

- ▶ Každú knihu napísal jedený autor a má jediný žáner.
- ▶ Knihu identifikuje dvojica (Názov, Autor), oplatí sa však pridať umelé idDiela, aby sme mali jednoduchý kľúč.
- ▶ Knihu vydáva len jediné vydavateľstvo, má len jediné vydanie a jeden jazyk. Môže však byť vo viacerých formátoch; ISBN sa prideluje osobitne pre jednotlivé formáty (papier, ePub, PDF...).

Identifikujte FZ a bezstratovo dekomponujte do BCNF.

Primary keys

V relácii (Meno, RodnéČíslo, ČísloPasu) sú atribúty RodnéČíslo a ČísloPasu kľúčmi. Pre externé nástroje je výhodné, keď možno záznamy identifikovať úplne jednoznačne, preto si zo všetkých kľúčov chceme vybrať primárny.

- ▶ PostgreSQL: **PRIMARY KEY** = NOT NULL UNIQUE.
- ▶ SQLite: PRIMARY KEY je unique, ale *nie* NOT NULL.

<https://www.sqlitetutorial.net/sqlite-primary-key/>

Primárny kľúč môže zahŕňať viac atribútov:

```
CREATE TABLE ta (
    a1 integer, a2 integer,
    PRIMARY KEY (a1, a2)
);
```

Surrogate keys

Pre niektoré tabuľky nemáme žiadnych prirodzených kandidátov na primárny kľúč (napr. ak tabuľka len zachytáva vzťah M:N). Vtedy môžeme pridať ako primárny kľúč **surrogate key** — umelý identifikátor, ktorý jednoznačne identifikuje záznam.

- ▶ PostgreSQL: atribút typu SERIAL.
- ▶ SQLite: atribút definovaný ako INTEGER PRIMARY KEY je automaticky aliasom pre ROWID (stĺpec s automaticky inkrementovanými hodnotami).

Surrogate keys

Umelé id možno využiť, aj keď máme iné kľúče, ale ich použitie by bolo nevýhodné:

- ▶ prirodzený kľúč nechceme zverejniť (napr. rodné číslo, číslo bankovej karty)
- ▶ pridlhé hodnoty kľúčových atribútov (index nad celými číslami zaberá menej miesta a je rýchlejší než nad dlhými reťazcami)
- ▶ chceme sa vyhnúť kompozitným kľúčom (ťažšia práca s externými nástrojmi, napr. ORM)
- ▶ pribúda počet atribútov v cudzích kľúčoch (pri ich reťazení)

Surrogate keys

Pridanie umelého id / nemení splňanie normálnej formy:

- ▶ Ak nejaká FZ pred pridaním / porušovala NF, porušuje ju naďalej.
- ▶ Ak nejaká FZ pred pridaním / neporušovala NF, nebude ju porušovať (pre 3NF a BCNF to vidno z toho, že každý pôvodný kľúč ostáva kľúčom, keďže z neho „vyplýva“ /).
- ▶ NF nemôže porušovať závislosť, ktorá má / na ľavej strane (lebo samotné / je nadkľúč).
- ▶ NF nemôže porušovať závislosť, ktorá má / na pravej strane, lebo každá množina atribútov, z ktorej „vyplýva“ /, je nadkľúčom (keďže z / „vyplýva“ všetko).

Funkčné závislosti v SQL

Aké funkčné závislosti musia platiť pre nasledujúcu reláciu?
(Hľadáme závislosti vynútené SQL kódom.)

```
CREATE TABLE customers (
    id          integer PRIMARY KEY,          -- I
    address     text NOT NULL,                 -- A
    name        text NOT NULL,                 -- N
    email       text NOT NULL UNIQUE,         -- E
    phone       text NOT NULL UNIQUE,         -- P
    UNIQUE(name, address)
);
```

Funkčné závislosti v SQL

Aké funkčné závislosti musia platiť pre nasledujúcu reláciu?
(Hľadáme závislosti vynútené SQL kódom.)

```
CREATE TABLE customers (
    id          integer PRIMARY KEY,          -- I
    address     text NOT NULL,                 -- A
    name        text NOT NULL,                 -- N
    email       text NOT NULL UNIQUE,         -- E
    phone       text NOT NULL UNIQUE,         -- P
    UNIQUE(name, address)
);
```

- ▶ PRIMARY KEY: $I \rightarrow ANEP$
- ▶ UNIQUE: $E \rightarrow I$, $P \rightarrow I$, $AN \rightarrow I$
- ▶ Z významu stĺpcov to vyzerá tak, že iné FZ (okrem tých, čo vyplývajú z uvedených) už medzi týmito atribútmi nie sú (a nechceme ich v SQL vynucovať).

Užitočná redundancia

Ak máme veľa dát, možno nechceme opakovane rátať agregačné funkcie, povedzme súčet. Jeho hodnotu H možno vypočítať dopredu, napr. tak, že záznamy dostávajú timestamp, podľa ktorého je spravený BTREE index, a v rámci dotazu počítame len H k nejakému timestampu plus záznamy, ktoré pribudli.

V takom prípade treba strážiť konzistenciu dát: pri zmene hodnôt originálnych záznamov (napr. zmazanie najstarších) aktualizovať H a naopak nedovoliť zmenu H bez úpravy originálnych záznamov.

Mechanizmus: **trigger** — funkcia uložená v databáze, ktorá sa spustí zakaždým, keď nastane definovaná udalosť, napr. pred pridaním či po pridaní riadka.

Užitočná redundancia

Zaužívané pravidlá je pri normalizácii občas (hoci zriedka) výhodné porušiť, treba však mať jasno, že prečo, a uviesť to v dokumentácii. Kompromis medzi znížením redundancie (vyššia NF) a zachovaním FZ nemá jediné správne objektívne riešenie, treba sa riadiť skúsenosťou.

Vo všeobecnosti čím lepšia normalizácia (pri zachovaní FZ), tým jednoduchšie sa robí vkladanie a úpravy dát, ale výpočet niektorých dotazov môže byť pomalší. Naopak viac redundancie zrýchli výpočet dotazov, ale skomplikuje udržiavanie konzistencie dát.

Ak v schéme existujúcej databázy vidíte neštandardné prvky, navrhujte ich zmeny až po dôsledkom preverení ich pôvodu a dôvodov ich vzniku. (Pozri tiež Chestertonov plot.)

Nedostatky v návrhu databázovej schémy

- ▶ zlý, neaktuálny či príliš komplikovaný dátový model (bežné pri digitalizácii verejnej správy)
- ▶ nejasnosti v tom, aké dáta sú potrebné a či je možné ich získať
- ▶ schéma, ktorú je ťažké upraviť pri zmene dátového modelu (napr. kvôli nadmernému využitiu custom riešení miesto štandardizovaných; vol'te jednoduchosť)
- ▶ chýbajúca dokumentácia, najmä zdôvodnenia netriviálnych rozhodnutí
- ▶ nadmerné použitie indexov (netreba ich „pre každý prípad“ pre každý atribút)
- ▶ zlé pomenovania (nejasné, málo špecifické, nekonzistentné, opakovanie názvu pre rôzne veci)

Foreign keys

Atribút obsiahnutý v niekoľkých reláciách:

- ▶ V matematickom relačnom modeli vždy „ten istý“.
- ▶ V SQL rovnako pomenované stĺpce v rôznych tabuľkách nemajú medzi sebou žiadny vzťah.

Foreign keys (cudzie kľúče) — mechanizmus na zabezpečenie súladu hodnôt toho istého atribútu v rôznych tabuľkách (referenčná integrita).

Foreign keys

```
CREATE TABLE ta (
    a1 integer REFERENCES tb,
    a2 integer,
    a3 integer,
    FOREIGN KEY (a2, a3) REFERENCES tc (c1, c2)
);
```

Cudzie kľúče možno pomenovať (ľahšie sa tak identifikuje dôvod zamietnutia operácie):

```
CONSTRAINT fk_name1 FOREIGN KEY a1 REFERENCES tb (b1)
CONSTRAINT fk_name2 FOREIGN KEY (a2, a3) REFERENCES tc (c1, c2)
```

Foreign keys

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    price numeric
);
CREATE TABLE orders (
    order_id integer PRIMARY KEY
);
CREATE TABLE order_items (
    order_id integer REFERENCES orders ON DELETE CASCADE,
    product_no integer REFERENCES products
        ON DELETE RESTRICT ON UPDATE CASCADE,
    quantity integer,
    PRIMARY KEY (product_no, order_id)
);
```

Foreign keys

Možnosti pre ON DELETE

- ▶ **CASCADE** – zmaže riadok, ktorý sa odkazoval na mazaný riadok
- ▶ **RESTRICT** – nezmaže nič, operácia DELETE skončí s chybou
- ▶ **SET NULL** – nastaví odkaz na NULL (zlyhá pre NOT NULL stĺpce)
- ▶ **SET DEFAULT** – nastaví hodnotu odkazu na defaultnú (zlyhá, ak takto vznikne nekorektný odkaz)
- ▶ **NO ACTION** – toto je default, DELETE skončí s chybou pri COMMIT (ak DBMS podporuje odloženú kontrolu, inak ihned')

Foreign keys

Možnosti pre **ON UPDATE**

- ▶ **CASCADE** – zmení odkazujúcu hodnotu na novú hodnotu odkazovanej (zachová väzbu medzi riadkami)
- ▶ **RESTRICT** – zabráni zmene v odkazovanej tabuľke, UPDATE ihneď skončí s chybou
- ▶ **SET NULL** – nastaví odkaz na NULL (väzba zanikne)
- ▶ **SET DEFAULT** – nastaví hodnotu odkazu na defaultnú (zlyhá, ak takto vznikne nekorektný odkaz)
- ▶ **NO ACTION** – toto je default, UPDATE skončí s chybou pri COMMIT (ak DBMS podporuje odloženú kontrolu, inak ihned')

Foreign keys

Kedy sa vyhodnocuje platnosť odkazu?

- ▶ **DEFERRED** – až pri commite transakcie
- ▶ **IMMEDIATE** – okamžite po vykonaní operácie

Foreign keys

Kedy sa vyhodnocuje platnosť odkazu?

- ▶ **DEFERRED** – až pri commite transakcie
- ▶ **IMMEDIATE** – okamžite po vykonaní operácie

Možnosti pri definícii cudzieho kľúča:

- ▶ **NOT DEFERRABLE** – vždy okamžité vyhodnocovanie
- ▶ **DEFERRABLE INITIALLY IMMEDIATE** – v rámci tx možno zvoliť, kedy vyhodnocovať, default je ihned'
- ▶ **DEFERRABLE INITIALLY DEFERRED** – v rámci tx možno zvoliť, kedy vyhodnocovať, default je commit

Mazanie záznamov

Pri nastavení ON DELETE treba postupovať citlivo.

Použitie CASCADE vedie k implicitnému mazaniu, ktorého si autor dotazu nemusí byť vedomý. Napríklad pri mazaní objednávky je jasné, že bude zmazaný aj zoznam produktov z tejto objednávky, ale pri mazaní učiteľa nechceme prísť o dátu z hodnotenia študentov, ktoré sú naňho naviazané, ani o materiály, ktoré pre školu vytvoril.

Použitie RESTRICT spraví mazanie explicitným. Klientsky program si napr. môže vyžiadať zoznam mazaných objektov a nechať ho potvrdiť používateľom, alebo zmeniť naviazanie cez cudzie klúče na nezmazané záznamy.

Mazanie záznamov

Nevýhody ON DELETE CASCADE:

- ▶ Nezamýšľaná strata záznamov.
- ▶ Náročnejšie debugovanie — ľažšie logovanie či pátranie po príčine straty dát.
- ▶ Pokles výkonu pri masívnom mazaní — tabuľka, z ktorej sa maže veľa vecí, ostane pre ostatné transakcie de facto zablokovaná. Mazanie môže trvať dlhé minúty a autor DELETE to nemusí vedieť.
- ▶ Bezpečnostné problémy, napr. ak máme prístupové práva na úrovni jednotlivých záznamov.

Ďalšie možnosti:

- ▶ *Soft delete*: Záznamy len označíme (v extra stĺpci) ako zmazané, ale ponecháme v databáze. Cez trigger sa dá spraviť cascade. Nevýhoda: vo všetkých dotazoch potom musíme filtrovať cez pridaný flag.
- ▶ Tabuľka historických hodnôt: zmazané záznamy presúvame (napr. trigger) do inej tabuľky a pripíšeme k nim obdobie platnosti.
- ▶ Temporal tables (SQL Server).

Vyššie normálne formy

Niekteré druhy redundancie sa nedajú zachytiť FZ.

Príklad: $R = \text{employee}(\text{emp_id}, \text{project}, \text{skill})$.

- ▶ Zamestnanec pracuje na viacerých projektoch.
- ▶ Zamestnanec má viacero zručností.
- ▶ Projekty a zručnosti zamestnanca **sú na sebe nezávislé**.

Problém (redundancia): Ak Anna pracuje na 7 projektoch a vie 'SQL' a 'Python', tabuľka **musí** obsahovať všetkých 14 riadkov karteziánskeho súčinu. Vznikajú anomálie: pridanie novej zručnosti pre Annu vyžaduje pridanie 7 nových záznamov.

R je v BCNF, lebo neplatia žiadne netriviálne FZ (klúčom je celá trojica emp_id , project , skill). BCNF však túto redundanciu neodstránila.

4NF

Multivalued dependencies (MVD) v relácii XYZ : $X \twoheadrightarrow Y$, ak pre danú hodnotu X existuje množina hodnôt Y , ktorá je rovnaká pre každú hodnotu Z .

Pre R : $emp_id \twoheadrightarrow project$, $emp_id \twoheadrightarrow skill$.

Relácia je v 4NF, ak v každej netriviálnej MVD $X \twoheadrightarrow Y$ je X je nadklúč.

MVD odstraňujeme tiež dekompozíciou, napr. R nahradíme $R_1 = (emp_id, project)$ a $R_2 = (emp_id, skill)$.

Literatúra

- ▶ entity-relationship diagrams
- ▶ normal forms
- ▶ normalization example 1
- ▶ normalization example 2
- ▶ <https://www.postgresql.org/docs/current/ddl-constraints.html>
- ▶ <https://www.postgresql.org/docs/current/sql-set-constraints.html>
- ▶ trigger v PostgreSQL
- ▶ trigger v SQLite
- ▶ <https://www.db-book.com/slides-dir/PDF-dir/ch6.pdf>
- ▶ <https://www.db-book.com/slides-dir/PDF-dir/ch7.pdf>
- ▶ <https://cs186berkeley.net/notes/note13/>

Online normalization tools (limited functionality)

- ▶ https://www.ict.griffith.edu.au/normalization_tools/normalization/index.html
- ▶ <https://arjo129.github.io/functionalDependencyCalculator/>
- ▶ <http://raymondcho.net/RelationalDatabaseTools/RelationalDatabaseTools.html>
- ▶ https://uisacad5.uis.edu/cgi-bin/mcrem2/database_design_tool.cgi

Na zamyslenie

- ▶ Nájdite príklad dekompozície, ktorá je bezstratová, ale nezachováva FZ.
- ▶ Nájdite príklad dekompozície, ktorá nie je bezstratová, ale zachováva všetky FZ (pričom platia aspoň dve).
- ▶ Ako možno vynútiť platnosť funkčnej závislosti $AB \rightarrow C$ v bezstratovej dekompozícii, ak žiadna relácia neobsahuje zároveň všetky tri atribúty A, B, C ?
- ▶ Čo znamená „zachovať FZ pri dekompozícii“? Je nutné zachovať každú FZ? Je to výhodné?
- ▶ Aj dobre napísaný trigger má nevýhody. Aké?

Na zamyslenie

- ▶ Akú časovú zložitosť má vyhodnotenie cudzieho kľúča pri vložení jedného záznamu? Majú na to vplyv indexy?
- ▶ Akú časovú zložitosť má CASCADE pri cudzích kľúčoch? Majú na to vplyv indexy?
- ▶ Uveďte príklad redundancie, ktorá sa nedá vyjadriť pomocou FZ.
- ▶ V prípade tabuľky, ktorá má veľa atribútov, ktorých hodnoty sú takmer všetky rovnaké (napr. NULL, FALSE či default), môže mať význam nahradíť ich všetky jediným stĺpcom typu json. V čom spočíva výhodnosť tohto postupu? Aká by bola normálna forma takto modifikovanej tabuľky?