

Databázové systémy

Návrh databáz v praxi: Od teórie k praxi

Návrh databáz v praxi: Od teórie k praxi

Tri piliere pri návrhu databázovej štruktúry:

1. Návrh entít a tabuliek
2. Návrh atribútov
3. Návrh vzťahov

1. Návrh entít a tabuliek

- **Princíp jednej zodpovednosti:** Každá tabuľka musí reprezentovať **iba jeden konkrétny objekt** reálneho sveta (napr. Zakaznik, Objednavka) alebo jeden koncept. Nemiešajte autorov a knihy do jednej tabuľky.
- **Jednotné pomenovanie:** Používajte výhradne **podstatné mená** a zvolte si jednotný formát (napr. **všetko v jednotnom čísle**: Pouzivatel, nie Pouzivatelia).
- **Vyhnite sa diakritike a medzerám:** Pre názvy tabuliek používajte **snake_case** (napr. detaily_objednavky) alebo CamelCase bez dĺžňov a mäkčeňov.
- **Rozlišujte entity a stavy:** Ak potrebujete sledovať históriu zmien, vytvorte samostatnú tabuľku pre históriu/logy, namiesto preťažovania hlavnej entity stavovými stĺpcami.

Číselník (Lookup Table) vs ENUM

- Novým vývojárom často napadne: „*Prečo na stavy objednávky (**Nová, Zaplatená, Odoslaná**) vytvárať celú tabuľku? Nedám tam len stĺpec typu ENUM alebo obyčajný text?*“
- V praxi platia tieto pravidlá:
- **Kedy použiť reálnu tabuľku (Číselník): V 90 % prípadov.** Ak sa hodnoty môžu v budúcnosti meniť bez programátora (napr. admin chce pridať novú kategóriu produktov), alebo ak k hodnote potrebujete viazať ďalšie dáta (napr. k menu pridať symbol € a kurz), **musí** to byť samostatná tabuľka.
- **Kedy stačí ENUM / Text v kóde:** Iba vtedy, ak sú hodnoty pevne späté s biznis logikou aplikácie a ich zmena by aj tak vyžadovala prepísanie kódu (napr. dni v týždni, pohlavie, alebo stav AKTIVNY/NEAKTIVNY).

Číselník (Lookup Table) namiesto ENUM

- **Neduplikujte** textové kategórie v hlavných tabuľkách. **Použite číselník.**
- **Konzistencia:** Namiesto písania textu „Slovensko“ 10 000-krát do tabuľky adries, odkazujte na ID v číselníku krajín (eliminácia preklepov).
- **Kedy použiť:** Na zoznamy hodnôt, ktoré sú relatívne stále (stavy objednávok, meny, typy dopravy, kategórie).
- **Základná anatómia:** Každý číselník obsahuje id (alebo kód) + zobrazovany_nazov + stĺpec poradie (pre správne zoradenie hodnôt v rozbaľovacom zozname v UI).

Pokročilé číselníky: Riadenie histórie v čase

- **Hodnoty** z číselníkov sa **nemažú**. Použite **časové okno**.
- **Riziko**: Ak prepíšete sadzbu DPH z 20 % na 23 % alebo zmeníte názov stavu, spätne pokazíte historické faktúry a reporty.
- **Riešenie z praxe**: Pridajte stĺpce **plati_od** a **plati_do** (typu DATETIME).
- **Presný audit**: Aplikácia pri zobrazení starého záznamu vytiahne z číselníka hodnotu platnú k danému dátumu v minulosti.
- **Plánovanie**: Umožňuje nahráť legislatívne zmeny alebo nové cenníky do databázy vopred – aktivujú sa samy, keď nastane správny čas.

2. Návrh atribútov

- **Jednoznačný primárny kľúč (PK):** Každá tabuľka musí mať definovaný primárny kľúč. Najlepšou praxou je používať **bezvýznamový (surrogátny) kľúč**, ako je automaticky inkrementované ID (id alebo pouzivatel_id), namiesto rodného čísla či e-mailu, ktoré sa môžu v budúcnosti zmeniť.
- **Atómicita hodnôt (1. normálna forma):** Jeden atribút musí obsahovať iba **jednu nedeliteľnú hodnotu**. Namiesto atribútu **adresa** vytvorte **ulica, mesto, psc**. Namiesto ukladania **viacerých telefónnych čísiel** do jedného poľa **vytvorte väzobnú tabuľku**.
- **Správna voľba dátových typov: Šetrite miesto.** Pre texty s premenlivou dĺžkou použite VARCHAR s **rozumným limitom**. Pre finančné sumy nikdy **nepoužívajte FLOAT**, ale presný typ **DECIMAL**.
- **Rozlišujte NULL a NOT NULL:** Dôsledne definujte, ktoré polia sú povinné. Atribúty, ktoré tvoria jadro entity (napr. cena pri produkte), by mali mať striktne nastavené NOT NULL.

3. Návrh vzťahov

- **Kardinalita a voliteľnosť:** Vždy presne určte typ vzťahu (1:1, 1:N, M:N) a či je účasť na vzťahu povinná alebo voliteľná. Pomáha to pri správnom zápise JOIN dopytov.
- **Dekompozícia vzťahu M:N (Mnoho ku mnohým):** Relačné databázy nedokážu priamo implementovať vzťah M:N. Každý takýto vzťah musíte v ERD rozbiť pomocou **medziľahlej (spojovacej) tabuľky**, ktorá bude obsahovať cudzie kľúče (FK) z oboch hlavných tabuliek (napr. vzťah medzi Student a Kurz vyrieši tabuľka ZapisanyKurz).
- **Konzistentné cudzie kľúče (FK):** Pre zachovanie prehľadnosti pomenujte cudzí kľúč rovnako ako primárny kľúč, na ktorý odkazuje, ideálne s prefixom tabuľky (napr. stĺpec zakaznik_id v tabuľke Objednavka odkazuje na id v tabuľke Zakaznik).

Vzťah 1:1 Spojiť do jednej tabuľky alebo rozdeliť?

- V učebniciach sa vzťah 1:1 často znázorňuje oddelene (napr. Pouzivatel a ProfilPouzivatela). V reálnej praxi **vo viac ako 80 % prípadov zlúčite tieto dve tabuľky do jednej**, pretože to zjednodušuje dopyty a zvyšuje výkon (**netreba robiť JOIN**).
- **Kedy** je ale v praxi správne **nechať ich rozdelené**?
 - **Optimalizácia výkonu (Vertikálny partitioning)**: Ak má tabuľka 50 stĺpcov, ale aplikácia pri 95 % dopytoch potrebuje len 3 stĺpce (napr. id, email, heslo), oplatí sa zvyšných 47 stĺpcov (biografia, nastavenia, adresa) presunúť do druhej tabuľky. Databáza tak číta z disku oveľa menej dát.
 - **Bezpečnosť a oprávnenia**: Citlivé údaje (napr. Pouzivatel vs BankoveUdaje_1:1) môžete v databáze oddeliť, aby ste k nim mohli reštriktívne obmedziť prístup na úrovni databázových práv.
 - **Voliteľnosť (Nullable columns)**: Ak by 90 % používateľov malo v daných stĺpcoch NULL (napr. len málokto vyplní firemné údaje), oddelením do 1:1 tabuľky udržíte hlavnú tabuľku čistú a bez zbytočných prázdnych polí.
- **Verdikt pre 1:1**: Pokiaľ nemáte obrovské množstvo dát alebo striktné bezpečnostné požiadavky, **dajte to do jednej (hlavnej) tabuľky**. Ušetríte si písanie zbytočných JOIN-ov.

Vzťah M:N – Použiť vlastné ID v spojovacej tabuľke?

- Otázka, či má mať spojovacia tabuľka (napr. **student_kurz**) vlastné **id (surrogátny kľúč)**, alebo má použiť zložený primárny kľúč (**student_id, kurz_id**), rozdeľuje programátorov na dva tábory. **V modernom vývoji sa však čoraz viac presadzuje prístup mať vlastné ID.**
- **Kedy dať vlastné ID určite?** Vždy, keď spojovacia tabuľka nesie **dodatočné informácie (atribúty vzťahu).**
- **Príklad:** Tabuľka **objednavka_produk** (obsahuje **objednavka_id** a **produkt_id**). V praxi tam ale takmer vždy potrebujete uložiť aj **mnoztvo, aktualna_cena_v_case_nakupu** alebo **zlava**. V tom momente sa zo spojovacej tabuľky stáva plnohodnotná entita (často nazývaná **polozka_objednavky**) a **vlastné id je tu prakticky nutnosťou.**
- **Verdikt pre M:N:** Ak používate akýkoľvek moderný backendový framework (Laravel, Spring, Django, .NET), **používajte vlastné autoinkrementované id (alebo UUID) ako primárny kľúč pre spojovaci tabuľku.** Ušetrí vám to bolesť hlavy pri programovaní. Nezabudnite však pridať **UNIQUE** obmedzenie na **dvojicu cudzích kľúčov**, aby **nevznikali duplicitné väzby.**

Zhrnutie

- **Nepreháňajte** to s predčasnou **normalizáciou** – **myslite na JOIN výkon.**
- V **spojovacích tabuľkách** používajte vlastné id + **UNIQUE** nad cudzími kľúčmi (ale iba tam, kde to dáva zmysel).
- Pre **primárne kľúče** voľte **bezvýznamové ID** (nespoliehajte sa na e-maily či rodné čísla).
- **Dôsledne definujte NOT NULL** tam, kde dáta jednoducho chýbať nesmú.