# Databázové systémy

Integritné obmedzenia

2026 Lukáš Miňo

# Motivácia

- Dáta bez integrity strácajú hodnotu.
- **Integritné obmedzenia** zabezpečujú presnosť, úplnosť a korektnosť dát počas celého ich životného cyklu.
- **Rola integritných obmedzení:** Pravidlá definované nad databázovou schémou, ktoré systém riadenia bázy dát (DBMS) vynucuje automaticky. Zabraňujú náhodnému alebo neoprávnenému poškodeniu dát.

# Definícia

- **Integrita** (celistvosť, úplnosť a neporušenosť) **dát** je platnosť (validity) dát. Integrita dát sa zabezpečuje pomocou **obmedzení**.
- **Integritné obmedzenia:** Súbor pravidiel a podmienok, ktoré garantujú, že dáta v databáze sú **korektné** v súlade s definovanými obmedzeniami **bez nedostatku** a **logicky konzistentné bez protirečení**. Databáza by mala obsahovať iba **platné hodnoty** (prípustné, správne – valid, accurate, correct).

# Prečo sú integritné obmedzenia dôležité?

- **Dôsledky chýbajúcich obmedzení:** Duplicitné kontá, objednávky naviazané na neexistujúcich zákazníkov, záporný vek či cena.
- **Výhody implementácie:** Presun biznis logiky z aplikácie priamo do databázy, vyššia bezpečnosť a jednoduchšia údržba zdrojového kódu aplikácie.

# Typy integritných obmedzení

- **Doménová integrita** (Domain Integrity)
- **Integrita entít** (Entity Integrity)
- **Referenčná integrita** (Referential Integrity)
- **Používateľom definovaná integrita** (User-defined Integrity)

# Doménová integrita (Domain Integrity)

- Definuje množinu povolených hodnôt pre konkrétny stĺpec.

- **Dátový typ** – text, číslo, dátum, napr. VARCHAR(20), INTEGER, DEC(5,2), DATE, ...
- **NOT NULL** – zákaz prázdnych hodnôt
- **CHECK** – overenie podmienky, napr. vek >= 18
- **DEFAULT** – predvolená hodnota

# Integrita entít (Entity Integrity)

- Každý riadok v tabuľke musí byť jednoznačne identifikovateľný.

- **PRIMARY KEY** (primárny kľúč, PK) – musí byť unikátny a **nesmie** byť NULL
  - každá tabuľka má **maximálne jeden** PK
  - PK sa skladá z jednej alebo viacerých stĺpcov
- **UNIQUE** – zabezpečí unikátnosť (napr. e-mail), ale **povoľuje** hodnotu NULL
  - na rozdiel od PK v tabuľke môže byť viac UNIQUE obmedzení

# Referenčná integrita (Referential Integrity)

- Udržiava správne vzťahy medzi tabuľkami.

- **FOREIGN KEY** (cudzí kľúč) je stĺpec alebo kombinácia stĺpcov v jednej databázovej tabuľke, ktorá **odkazuje na primárny kľúč** (prípadne unikátny (UNIQUE) kľúč) v inej tabuľke (alebo v tej istej tabuľke pri rekurzívnom vzťahu).

# FOREIGN KEY (cudzí kľúč)

- **Vytvára vzťahy:** Prepája "dcérsku" (podriadenú) tabuľku s "materskou" (nadriadenou) tabuľkou.
- **Kontroluje hodnoty:** DBMS zabezpečí, že v stĺpci cudzieho kľúča sa môžu nachádzať **iba také hodnoty, ktoré už existujú** v primárnom kľúči odkazovanej tabuľky (alebo hodnota NULL, ak je to povolené).
- **Zabraňuje sirotským záznamom:** Nedovolí vytvoriť záznam, ktorý by odkazoval na neexistujúci objekt (napr. objednávku pre neexistujúceho zákazníka).

# Referenčné akcie na cudzom kľúči (Referential Actions)

- Reakcia na vymazanie/zmenu primárneho kľúča
- **Čo definujeme:** Správanie prepojených (dcérskych) záznamov, aby v databáze nevznikli neplatné odkazy.

- ON DELETE / UPDATE RESTRICT / NO ACTION – blokuje operáciu (predvolené).
- ON DELETE / UPDATE CASCADE – vymaže/aktualizuje aj prepojené riadky.
- ON DELETE / UPDATE SET NULL – nastaví hodnotu cudzieho kľúča na NULL.
- ON DELETE / UPDATE SET DEFAULT – nastaví predvolenú hodnotu.

# Integrita dát

| Typ integrity | Čo definuje? | Príklad z E-shopu |
|---|---|---|
| **Integrita entít** | Každá tabuľka musí mať unikátny Primárny kľúč (PK), ktorý nesmie byť NULL. | Nemôžu existovať dve objednávky s rovnakým id_objednavky. |
| **Referenčná integrita** | Cudzí kľúč (FK) musí odkazovať na existujúci PK alebo byť prázdny. | Systém nedovolí priradiť objednávku neexistujúcemu zákazníkovi. |
| **Doménová integrita** | Hodnoty v stĺpci musia patriť do množiny povolených hodnôt. | V stĺpci mnozstvo nemôže byť záporné číslo alebo text. |

# Používateľom definovaná integrita (User-defined Integrity)

- Umožňuje definovať špecifické biznis pravidlá a bezpečnostné politiky, ktoré nie je možné pokryť štandardnými kľúčmi, dátovými typmi ani podmienkami CHECK.
- **Kedy sa používa:** Ak overenie správnosti dát vyžaduje komplexnú logiku, porovnávanie hodnôt naprieč viacerými tabuľkami alebo analýzu histórie zmien.
- **Príklad:** Kontrola, či zákazník neprekročil limit nezaplatených faktúr.

- **Spúšťače (triggers):** Procedurálny kód spúšťaný automaticky databázou na základe konkrétnej udalosti.
  - Udalosti: INSERT, UPDATE, DELETE, CREATE, ALTER, DROP, LOGON, LOGOFF, STARTUP, SHUTDOWN
- **Uložené procedúry (stored procedures) a funkcie (functions):** Zapuzdrená logika priamo na databázovom serveri.

- Viac sa dozviete v letnom semestri

# Spôsoby zápisu integritných obmedzení

# Biznis pravidlá

- Každý zákazník má unikátne ID a e-mail.
- E-mail, meno a priezvisko nesmú byť prázdne.
- Cena objednávky musí byť kladné číslo.
- Objednávka musí patriť existujúcemu zákazníkovi.
- Stav môže byť len '*Prijatá*', '*Zaplatená*', '*Odoslaná*'.

# 1. Zápis na úrovni stĺpca (Column-level)

```
CREATE TABLE zakaznik (
   id INT PRIMARY KEY,
   meno VARCHAR(100) NOT NULL,
   priezvisko VARCHAR(100) NOT NULL,
   email VARCHAR(150) NOT NULL UNIQUE
);
CREATE TABLE objednavka (
   id INT PRIMARY KEY,
   zakaznik_id INT NOT NULL REFERENCES zakaznik(id),
   cena DECIMAL(10, 2) NOT NULL CHECK (cena > 0),
   stav VARCHAR(20) NOT NULL CONSTRAINT chk_objednavka_stav CHECK (stav IN ('Prijatá', 'Zaplatená',
'Odoslaná'))
);
```

# Upozornenie

- **Zložené kľúče:** Ak primárny alebo cudzí kľúč pozostáva zo zloženého kľúča (viac ako jeden stĺpec), tento **priamy zápis** za stĺpcom **použiť nemôžete**. Vtedy musíte prejsť na zápis na úrovni tabuľky (na samostatný riadok na konci).
- **Pozor na MySQL:** Ak používate databázu **MySQL**, tento skrátený zápis za stĺpcom (REFERENCES zakaznik(id)) databáza síce neodmietne a kód prebehne bez chyby, ale **v skutočnosti cudzí kľúč nevytvorí** (odignoruje ho). V MySQL musíte foreign key pre správnu funkčnosť definovať buď na úrovni tabuľky (na samostatnom riadku) alebo cez ALTER TABLE.
- V systémoch ako **PostgreSQL**, **MS SQL** alebo **Oracle** zápis za stĺpcom pre cudzí kľúč funguje.

# Pomenované vs. nepomenované constraints

- Pri akomkoľvek zápise sa môžete rozhodnúť, či obmedzenie pomenujete sami, alebo to necháte na databázu.
- **Nepomenované (Implicitné):** Databáza vygeneruje náhodný názov (napr. SYS_C007143). Výhodou je rýchlejší zápis, nevýhodou je, že ak budete chcieť constraint neskôr vymazať, musíte si jeho názov dohľadávať v systémových tabuľkách.
  - CHECK (cena > 0)
- **Pomenované (Explicitné):** Pomocou kľúčového slova CONSTRAINT definujete vlastný názov.
  - CONSTRAINT chk_objednavka_cena CHECK (cena > 0)
  - **Odporúčané**

# Zápis na úrovni tabuľky (Table-level) s udelením vlastných názvov

```
CREATE TABLE zakaznik (
   id INT,
   meno VARCHAR(100) NOT NULL,
   priezvisko VARCHAR(100) NOT NULL,
   email VARCHAR(150) NOT NULL,

   CONSTRAINT pk_zakaznik PRIMARY KEY (id),
   CONSTRAINT uc_zakaznik_email UNIQUE (email)
);
```

# Zápis na úrovni tabuľky (Table-level) s udelením vlastných názvov

```
CREATE TABLE objednavka (
   id INT,
   zakaznik_id INT NOT NULL,
   cena DECIMAL(10, 2) NOT NULL,
   stav VARCHAR(20) NOT NULL,

   CONSTRAINT pk_objednavka PRIMARY KEY (id),
   CONSTRAINT fk_objednavka_zakaznik FOREIGN KEY (zakaznik_id) REFERENCES zakaznik(id),
   CONSTRAINT chk_objednavka_cena CHECK (cena > 0),
   CONSTRAINT chk_objednavka_stav CHECK (stav IN ('Prijatá', 'Zaplatená', 'Odoslaná'))
);
```

# 3. Zápis cez ALTER TABLE - dodatočné priradenie

```
-- 1. Vytvorenie čistých tabuliek
CREATE TABLE zakaznik (
    id INT,
    meno VARCHAR(100) NOT NULL,
    priezvisko VARCHAR(100) NOT NULL,
    email VARCHAR(150) NOT NULL
);
CREATE TABLE objednavka (
    id INT,
    zakaznik_id INT NOT NULL,
    cena DECIMAL(10, 2) NOT NULL,
    stav VARCHAR(20) NOT NULL
);
```

# 3. Zápis cez ALTER TABLE - dodatočné priradenie

```
-- 2. Dodatočné pridanie kľúčov a podmienok

ALTER TABLE zakaznik ADD CONSTRAINT pk_zakaznik PRIMARY KEY (id);

ALTER TABLE zakaznik ADD CONSTRAINT uc_zakaznik_email UNIQUE (email);


ALTER TABLE objednavka ADD CONSTRAINT pk_objednavka PRIMARY KEY (id);

ALTER TABLE objednavka ADD CONSTRAINT fk_objednavka_zakaznik FOREIGN KEY (zakaznik_id)
REFERENCES zakaznik(id);

ALTER TABLE objednavka ADD CONSTRAINT chk_objednavka_cena CHECK (cena > 0);

ALTER TABLE objednavka ADD CONSTRAINT chk_objednavka_stav CHECK (stav IN ('Prijatá', 'Zaplatená',
'Odoslaná'));
```

# Spôsoby zápisu integritných obmedzení

| Vlastnosť | Na úrovni stĺpca (Column-level) | Na úrovni tabuľky (Table-level) | Cez ALTER TABLE |
|---|---|---|---|
| **Kedy použiť** | Pri vytváraní tabuľky pre jednoduché obmedzenia. | Pri vytváraní tabuľky pre komplexné/zložené obmedzenia. | Pre existujúce tabuľky v bežiacej databáze. |
| **Zložené kľúče (viac stĺpcov)** | Nie je možné. | Podporované. | Podporované. |
| **Čitateľnosť kódu** | Výborná pre rýchly prehľad o stĺpci. | Prehľadná pri veľkom množstve logických väzieb. | Čistá (nezaťažuje štruktúru CREATE príkazu). |

# 4. Zmiešaný zápis (Mixed-level)

```
CREATE TABLE objednavka (
   id INT,
   zakaznik_id INT NOT NULL,
   cena DECIMAL(10, 2) NOT NULL CHECK (cena > 0),
   stav VARCHAR(20) NOT NULL CHECK (stav IN ('Prijatá', 'Zaplatená', 'Odoslaná')),

   CONSTRAINT pk_objednavka PRIMARY KEY (id),
   CONSTRAINT fk_objednavka_zakaznik FOREIGN KEY (zakaznik_id) REFERENCES zakaznik(id)
);
```

# Výhody zmiešaného zápisu

- **Čitateľnosť:** Pri pohľade na stĺpce okamžite vidíte základné pravidlá (či je povinný, jedinečný alebo aký má rozsah/CHECK).
- **Kompatibilný s MySQL:** Keďže je FOREIGN KEY zapísaný na úrovni tabuľky, spoľahlivo funguje vo všetkých SQL databázach vrátane MySQL.
- **Čistota kódu:** Príkaz CREATE TABLE nie je prehnane dlhý ako pri čistom Table-level zápise, no zachováva si jeho hlavné výhody.

# Referenčné akcie na cudzom kľúči (Referential Actions)

```
ALTER TABLE objednavka
ADD CONSTRAINT fk_objednavka_zakaznik
FOREIGN KEY (zakaznik_id) REFERENCES zakaznik(id)
ON DELETE RESTRICT
ON UPDATE CASCADE;
```

# Referenčné akcie na cudzom kľúči (Referential Actions)

```
ALTER TABLE objednavka
ADD CONSTRAINT fk_objednavka_zakaznik
FOREIGN KEY (zakaznik_id) REFERENCES zakaznik(id)
ON DELETE SET NULL
ON UPDATE CASCADE;
```

# Čo sa stane pri porušení integrity?

# Čo sa stane pri porušení integrity?

- **Zachytenie chyby:** Keď SQL príkaz poruší pravidlo, DBMS transakciu okamžite preruší (nastane rollback) a vráti chybový kód s popisom.
- **Úloha vývojára:** Tieto výnimky musíme v aplikačnom kóde (Java, Python, C#) zachytiť (try-catch).
- **Zobrazenie používateľovi:** Surová chyba z DB nesmie preniknúť do frontend aplikácie – používateľovi musíme zobraziť ľudsky zrozumiteľnú správu (napr. "Zadaná cena nie je platná").

# Čo sa stane pri porušení integrity?

- INSERT INTO objednavka (id, zakaznik_id, cena, stav)
  VALUES (101, 1, -50.00, 'Prijatá‘);
- ERROR 3819 (HY000): Check constraint 'chk_objednavka_cena' is violated.

- **Error Code 3819:** Špecifický kód MySQL pre porušenie podmienky CHECK. Podľa tohto čísla dokáže backend kód (napr. v Jave alebo C#) okamžite identifikovať, o aký typ chyby ide, bez zložitého čítania textu.
- **SQLState HY000:** Všeobecný stav chyby (chyba integrity dát).
- **Názov constraintu:** MySQL priamo v správe uvádza 'chk_objednavka_cena', takže presne viete, ktorý stĺpec a aké pravidlo zlyhalo.

# Odstraňovanie integritných obmedzení (DROP CONSTRAINT)

# Odstraňovanie obmedzení (DROP CONSTRAINT)

- **Prečo mažeme obmedzenia:** Zmena biznis logiky (napr. e-shop zavedie darčeky s nulovou cenou) alebo migrácia dát.
- **Kľúčové pravidlo:** Pre jednoduché zmazanie musíme poznať presný názov obmedzenia – preto je dôležité používať pomenované obmedzenia (CONSTRAINT).

- ALTER TABLE objednavka DROP CONSTRAINT chk_objednavka_cena;
- ALTER TABLE zakaznik DROP CONSTRAINT uc_zakaznik_email;
- ALTER TABLE objednavka DROP CONSTRAINT fk_objednavka_zakaznik;
- ALTER TABLE zakaznik DROP CONSTRAINT pk_zakaznik;

# Odstraňovanie obmedzení (DROP CONSTRAINT) v MySQL

- V databáze **MySQL** (engine InnoDB) má príkaz DROP špecifickú syntax, ktorá sa líši od ANSI SQL štandardu. V starších verziách (pred 8.0.16) navyše chýbala plná podpora niektorých obmedzení.

- ALTER TABLE objednavka DROP CHECK chk_objednavka_cena;
- ALTER TABLE zakaznik DROP INDEX uc_zakaznik_email;
- ALTER TABLE objednavka DROP FOREIGN KEY fk_objednavka_zakaznik;
- ALTER TABLE zakaznik DROP PRIMARY KEY;

# Závislosť cudzích kľúčov

- Ak sa pokúsite zmazať primárny kľúč v tabuľke **zakaznik**, pričom tabuľka **objednavka** naň stále odkazuje cez **zakaznik_id**, databáza príkaz **odmietne** a vyhodí **chybu integrity**.
- **Riešenie:** Manuálne zmazanie v správnom poradí.

- ALTER TABLE objednavka DROP CONSTRAINT fk_objednavka_zakaznik;
- ALTER TABLE zakaznik DROP CONSTRAINT pk_zakaznik;

# Odstranenie NOT NULL a DEFAULT

- Tieto dve obmedzenia sa v SQL neodstraňujú cez tradičný DROP CONSTRAINT, ale úpravou samotného stĺpca (zmena dátového typu alebo stavu).

- **PostgreSQL**
- ALTER TABLE zakaznik ALTER COLUMN meno DROP NOT NULL;
- ALTER TABLE objednavka ALTER COLUMN stav DROP DEFAULT;
- **MS SQL**
- ALTER TABLE zakaznik ALTER COLUMN meno VARCHAR(100) NULL;
- **MySQL**
- ALTER TABLE zakaznik MODIFY meno VARCHAR(100) NULL;
- ALTER TABLE objednavka ALTER COLUMN stav DROP DEFAULT;

# Úprava dátového typu stĺpca

- Ak potrebujete zmeniť dátový typ stĺpca (napr. zväčšiť dĺžku textu alebo zmeniť celé číslo na desatinné), syntax sa medzi databázovými systémami výrazne líši.

- **PostgreSQL (ALTER COLUMN ... TYPE)**
- ALTER TABLE zakaznik ALTER COLUMN meno TYPE VARCHAR(250);
- **MS SQL (ALTER COLUMN)**
- ALTER TABLE zakaznik ALTER COLUMN meno VARCHAR(250) NOT NULL;
  -- ak NOT NULL neuvedieme, automaticky sa zmení na NULL
- **MySQL (MODIFY / MODIFY COLUMN)**
- ALTER TABLE zakaznik MODIFY COLUMN meno VARCHAR(250) NOT NULL;
- **Oracle (MODIFY)**
- ALTER TABLE zakaznik MODIFY meno VARCHAR2(250);

# Úprava dátového typu stĺpca

- **Závislosti (Constraints):** Databáza vám nedovolí zmeniť dátový typ stĺpca, ak je naň naviazaný PRIMARY KEY, FOREIGN KEY alebo sa využíva v CHECK podmienke. Tieto obmedzenia musíte najprv odstrániť (DROP), zmeniť typ stĺpca a následne ich znova pridať (ADD).
- **Strata dát (Truncation):** Pri zmenšovaní rozsahu (napr. z VARCHAR(100) na VARCHAR(20)) databáza príkaz odmietne, ak už v tabuľke existujú záznamy s dlhším textom ako 20 znakov.
- **Konverzia typov:** Zmena z textu (VARCHAR) na číslo (INT) prejde len vtedy, ak sú všetky existujúce texty v tabuľke validnými číslami. V PostgreSQL je niekedy nutné explicitne použiť konverznú klauzulu USING meno::integer.

# Generovanie ID (Surrogate Keys)

- Každý DBMS generuje automatické sekvencie pre primárne kľúče vlastnou syntaxou.

- **Cez samostatnú sekvenciu (PostgreSQL / Oracle)**
- CREATE SEQUENCE seq_objednavky START WITH 100 INCREMENT BY 1;
- objednavka_id INT DEFAULT NEXTVAL('seq_objednavky') PRIMARY KEY -- (PostgreSQL)
- objednavka_id INT DEFAULT seq_objednavky.NEXTVAL PRIMARY KEY -- (Oracle)
- **PostgreSQL: Používa SERIAL alebo IDENTITY**
- objednavka_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY
- objednavka_id SERIAL PRIMARY KEY
- **MS SQL: Používa IDENTITY(štart, krok)**
- objednavka_id INT IDENTITY(1,1) PRIMARY KEY
- **MySQL: Používa kľúčové slovo AUTO_INCREMENT**
- objednavka_id INT AUTO_INCREMENT PRIMARY KEY

# Zhrnutie a rady

- **Vždy definujte Primárny kľúč**
  - Každá relácia (tabuľka) musí explicitne definovať primárny kľúč na zabezpečenie jedinečnosti entít a elimináciu anomálií pri modifikácií dát.
- **Presúvajte validáciu čo najbližšie k dátam**
  - Hoci je validácia na aplikačnej vrstve (UI/API) nevyhnutná pre UX, databázový subsystém reprezentuje finálnu autoritu. Integritné obmedzenia na úrovni DBMS garantujú konzistenciu dát nezávisle od stavu klientskych aplikácií.
- **Vždy pomenúvajte obmedzenia (CONSTRAINT)**
  - Vyhnite sa implicitnému generovaniu názvov zo strany DBMS (napr. SYS_C00123). Explicitne pomenované obmedzenia (napr. chk_objednavka_cena) sú kritické pre programovú interpretáciu runtime výnimiek a bezchybný database refactoring (schema migrations).
- **Dbajte na analýzu rizík tranzitívnej deštrukcie dát pri CASCADE**
  - Deklarácia referenčnej akcie ON DELETE CASCADE maximalizuje vývojársku efektivitu, no pri komplexných reláciách typu M:N (Many-to-Many) nesie riziko lavínového zmazania historických transakčných dát. Pre kritické dáta (faktúry, audity) preferujte RESTRICT alebo SET NULL.
- **Optimalizácia domén na úrovni fyzického dátového modelu**
  - Voľba dátového typu je primárnym sémantickým obmedzením. Nahradenie generických typov (VARCHAR) exaktnými doménami (DATE, DECIMAL, ENUM) znižuje pamäťovú náročnosť, optimalizuje exekučný plán indexov a zabraňuje neplatným stavom priamo na úrovni diskového systému.