**1. Navržení a zhotovení relační SQL databáze – Cheet Sheet**

**Základní pojmy a principy:**

* **Relační databáze**: Databáze založená na relačním modelu, kde jsou data uložena v tabulkách (relations), které mezi sebou mohou mít vztahy.
* **Tabulka** (Table): Základní struktura, která obsahuje data. Každá tabulka je složena z **sloupců (columns)** a **řádků (rows)**.
  + Každý sloupec má definovaný datový typ (např. VARCHAR, INT, DATE).
  + Každý řádek obsahuje hodnoty odpovídající těmto sloupcům.
* **Primární klíč (Primary Key)**: Unikátní identifikátor pro každý řádek v tabulce. Může být složený (více sloupců).
  + Např. v tabulce Zákazníci by mohl být sloupec id primární klíč.
* **Cizí klíč (Foreign Key)**: Sloupec v tabulce, který odkazuje na primární klíč jiné tabulky. Tento klíč vytváří **relaci** mezi tabulkami.
  + Např. tabulka Objednávky může mít sloupec zakaznik\_id, který bude cizí klíč odkazující na id tabulky Zákazníci.
* **Normalizace**: Proces, který eliminuje redundantní data a zajišťuje konzistenci.
  + **1NF**: Tabulka musí mít jedinečné řádky a každý sloupec obsahuje atomické hodnoty (bez více hodnot v jednom sloupci).
  + **2NF**: Tabulka je ve 1NF a každý neklíčový sloupec je plně závislý na primárním klíči.
  + **3NF**: Tabulka je ve 2NF a žádný neklíčový sloupec není závislý na jiném neklíčovém sloupci.

**Datové typy:**

* **INT**: Celé číslo (např. 5, -100).
* **VARCHAR(n)**: Text s proměnnou délkou (např. VARCHAR(255) pro jména, adresy).
* **CHAR(n)**: Text pevné délky.
* **DATE**: Datum ve formátu YYYY-MM-DD.
* **DECIMAL(p, s)**: Desetinné číslo s určeným počtem číslic před a za desetinnou čárkou.
* **BOOLEAN**: Hodnota pravda/nepravda (1/0).

**Příklady SQL příkazů pro vytváření tabulek:**

**Vytvoření tabulky Zákazníci:**

CREATE TABLE zakaznici (

id INT AUTO\_INCREMENT PRIMARY KEY,

jmeno VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

telefon VARCHAR(15),

adresa TEXT,

datum\_registrace DATE DEFAULT (CURRENT\_DATE)

);

* **AUTO\_INCREMENT**: Automaticky generuje jedinečná čísla pro id.
* **NOT NULL**: Zajistí, že sloupec nemůže obsahovat hodnotu NULL.
* **UNIQUE**: Zajistí, že hodnota v tomto sloupci je unikátní.
* **DEFAULT**: Nastaví výchozí hodnotu pro sloupec, pokud není zadaná.

**Vytvoření tabulky Objednávky s cizím klíčem:**

CREATE TABLE objednavky (

id INT AUTO\_INCREMENT PRIMARY KEY,

zakaznik\_id INT,

datum\_objednavky DATE NOT NULL,

celkova\_cena DECIMAL(10, 2),

FOREIGN KEY (zakaznik\_id) REFERENCES zakaznici(id)

ON DELETE CASCADE

);

* **FOREIGN KEY**: Definuje cizí klíč, který se vztahuje na id v tabulce zakaznici.
* **ON DELETE CASCADE**: Pokud je záznam v tabulce zakaznici smazán, všechny související objednávky se automaticky smažou.

**Příklad SELECT dotazu:**

* **SELECT**: Používá se pro získání dat z tabulky.

SELECT jmeno, email, telefon

FROM zakaznici

WHERE email LIKE '%@gmail.com%';

* Tento dotaz vybere jména, e-maily a telefony zákazníků, kteří mají e-mailovou adresu končící na @gmail.com.

**Příklad JOIN dotazu:**

* **INNER JOIN**: Spojí dvě tabulky podle podmínky (např. cizí klíč).

SELECT z.jmeno, o.datum\_objednavky, o.celkova\_cena

FROM zakaznici z

INNER JOIN objednavky o ON z.id = o.zakaznik\_id

WHERE o.celkova\_cena > 1000;

* Tento dotaz vybere jména zákazníků a jejich objednávky, pokud byla cena objednávky vyšší než 1000.

**Příklad vkládání dat (INSERT):**

INSERT INTO zakaznici (jmeno, email, telefon, adresa)

VALUES ('Jan Novák', 'jan.novak@email.cz', '123456789', 'Ulice 123');

* Tento příkaz vloží nový záznam do tabulky zakaznici.

**Příklad aktualizace dat (UPDATE):**

UPDATE zakaznici

SET telefon = '987654321'

WHERE id = 1;

* Tento příkaz aktualizuje telefonní číslo zákazníka s id = 1.

**Příklad smazání dat (DELETE):**

DELETE FROM zakaznici

WHERE id = 2;

* Tento příkaz smaže zákazníka s id = 2.

**Tipy pro návrh databáze:**

1. **Zaměř se na normalizaci**, abys minimalizovala duplicity a zajistila konzistenci dat.
2. **Vždy definuj primární klíč** pro každou tabulku.
3. **Používej cizí klíče** pro propojení tabulek a zajištění integrity dat.
4. **Optimalizuj dotazy**: Např. použití indexů na často dotazované sloupce (např. email, id).
5. **Testuj na malých vzorcích dat**: Ujisti se, že tvoje dotazy a struktura databáze fungují správně.

**Vztahy mezi tabulkami (Relace)**

**1. Relační typy vztahů mezi tabulkami:**

* **1:1 (Jedna k jedné)**: Každý řádek v jedné tabulce odpovídá právě jednomu řádku v druhé tabulce. Tato relace není příliš běžná a je vhodná, pokud potřebuješ oddělit data, ale stále je budeš potřebovat propojovat. Např. každý uživatel má jedinečnou adresu.
* **1:M (Jedna k mnoha)**: Jeden řádek v první tabulce může být propojen s více řádky v druhé tabulce. Tento vztah je nejběžnější a používá se např. v případě zákazníka a jeho objednávek, kdy každý zákazník může mít více objednávek, ale každá objednávka patří jen jednomu zákazníkovi.
* **M:M (Mnoho k mnoha)**: Tento vztah nastává, když více řádků v jedné tabulce může být propojeno s více řádky v druhé tabulce. Aby se tento vztah vyřešil, často používáme **propojovací tabulku**, která obsahuje cizí klíče z obou tabulek. Např. studenti a kurzy – student může navštěvovat více kurzů a každý kurz může mít více studentů.

**2. Příklady cizích klíčů a jejich využití:**

* **1:1 vztah (Zákazník a profil):**

Tabulka zakaznici:

CREATE TABLE zakaznici (

id INT AUTO\_INCREMENT PRIMARY KEY,

jmeno VARCHAR(100),

email VARCHAR(100)

);

Tabulka profily (1:1 vztah se zákazníkem):

CREATE TABLE profily (

id INT PRIMARY KEY,

zakaznik\_id INT,

profilova\_fotka VARCHAR(255),

biografie TEXT,

FOREIGN KEY (zakaznik\_id) REFERENCES zakaznici(id)

);

* **1:M vztah (Zákazník a objednávky):**

Tabulka zakaznici:

CREATE TABLE zakaznici (

id INT AUTO\_INCREMENT PRIMARY KEY,

jmeno VARCHAR(100),

email VARCHAR(100)

);

Tabulka objednavky (1:M vztah):

CREATE TABLE objednavky (

id INT AUTO\_INCREMENT PRIMARY KEY,

zakaznik\_id INT,

datum\_objednavky DATE,

celkova\_cena DECIMAL(10, 2),

FOREIGN KEY (zakaznik\_id) REFERENCES zakaznici(id)

);

* **M:M vztah (Student a kurz):**

Tabulka studenti:

CREATE TABLE studenti (

id INT AUTO\_INCREMENT PRIMARY KEY,

jmeno VARCHAR(100)

);

Tabulka kurzy:

CREATE TABLE kurzy (

id INT AUTO\_INCREMENT PRIMARY KEY,

nazev VARCHAR(100)

);

Propojovací tabulka studenti\_kurzy pro vztah M:M:

CREATE TABLE studenti\_kurzy (

student\_id INT,

kurz\_id INT,

PRIMARY KEY (student\_id, kurz\_id),

FOREIGN KEY (student\_id) REFERENCES studenti(id),

FOREIGN KEY (kurz\_id) REFERENCES kurzy(id)

);

**Příklad JOIN dotazů**

**JOIN** se používá pro spojování dvou nebo více tabulek na základě **společného sloupce**, což umožňuje získat data z více tabulek v jednom dotazu.

**1. INNER JOIN (Spojení pouze řádků, které mají odpovídající záznamy v obou tabulkách):**

* **Příklad**: Chceme získat seznam objednávek a zákazníků, kteří je udělali, ale pouze pro objednávky, které existují v obou tabulkách.

SELECT z.jmeno, o.datum\_objednavky, o.celkova\_cena

FROM zakaznici z

INNER JOIN objednavky o ON z.id = o.zakaznik\_id;

**Význam**: Tento dotaz vybere jména zákazníků, datum jejich objednávky a celkovou cenu, ale pouze pro ty objednávky, které mají odpovídajícího zákazníka v tabulce zakaznici.

**2. LEFT JOIN (Zobrazí všechny řádky z levé tabulky a odpovídající řádky z pravé tabulky. Pokud pravá tabulka neexistuje, zobrazí NULL):**

* **Příklad**: Chceme získat seznam všech zákazníků a jejich objednávek, ale i těch zákazníků, kteří nemají žádnou objednávku.

SELECT z.jmeno, o.datum\_objednavky, o.celkova\_cena

FROM zakaznici z

LEFT JOIN objednavky o ON z.id = o.zakaznik\_id;

**Význam**: Tento dotaz vrátí seznam všech zákazníků a jejich objednávky, pokud nějaké mají. Pokud zákazník nemá žádnou objednávku, místo dat objednávky se zobrazí NULL.

**3. RIGHT JOIN (Protipól LEFT JOIN – zobrazí všechny řádky z pravé tabulky a odpovídající řádky z levé tabulky):**

* **Příklad**: Chceme získat seznam všech objednávek a jejich zákazníků, včetně těch objednávek, které nemají přiřazeného zákazníka.

SELECT z.jmeno, o.datum\_objednavky, o.celkova\_cena

FROM zakaznici z

RIGHT JOIN objednavky o ON z.id = o.zakaznik\_id;

**4. FULL OUTER JOIN (Zobrazí všechny řádky z obou tabulek. Pokud v jedné tabulce není odpovídající záznam, zobrazí NULL):**

* **Příklad**: Chceme získat všechny zákazníky a všechny objednávky, i když některé objednávky nemají zákazníka a někteří zákazníci nemají objednávky.

SELECT z.jmeno, o.datum\_objednavky, o.celkova\_cena

FROM zakaznici z

FULL OUTER JOIN objednavky o ON z.id = o.zakaznik\_id;

**Pokročilé příklady JOIN:**

**Spojení více tabulek:**

Pokud máme více než dvě tabulky, můžeme je spojovat postupně.

SELECT z.jmeno, k.nazev AS kurz, o.datum\_objednavky, o.celkova\_cena

FROM zakaznici z

INNER JOIN objednavky o ON z.id = o.zakaznik\_id

INNER JOIN studenti\_kurzy sk ON z.id = sk.student\_id

INNER JOIN kurzy k ON sk.kurz\_id = k.id;

**Význam**: Tento dotaz spojí zákazníky, jejich objednávky a kurzy, které navštěvují, vše v jednom výsledku.

**Self JOIN (Spojení tabulky s ní samou):**

Pokud má tabulka relaci sama k sobě (např. zaměstnanci a jejich nadřízení), použijeme **SELF JOIN**:

SELECT e1.jmeno AS zamestnanec, e2.jmeno AS nadrizeny

FROM zamestnanci e1

LEFT JOIN zamestnanci e2 ON e1.nadrizeny\_id = e2.id;

**Význam**: Tento dotaz vrátí seznam zaměstnanců a jejich nadřízených. Pokud zaměstnanec nemá nadřízeného, zobrazí se NULL.

**Shrnutí:**

* **INNER JOIN**: Spojuje pouze odpovídající řádky v obou tabulkách.
* **LEFT JOIN**: Zobrazí všechny řádky z levé tabulky, včetně těch, které nemají odpovídající řádek v pravé tabulce.
* **RIGHT JOIN**: Zobrazí všechny řádky z pravé tabulky.
* **FULL OUTER JOIN**: Zobrazí všechny řádky z obou tabulek, přičemž u chybějících odpovídajících záznamů zobrazí NULL.
* **Self JOIN**: Používá se pro propojení tabulky s ní samou.

Toto by mělo pokrýt všechny klíčové pojmy a příklady pro **vztahy mezi tabulkami** a **JOIN**. Jak to vypadá, chceš pokračovat k dalšímu tématu nebo máš nějaké dotazy?