SQL pro SPŠ a VOŠ Kladno

Jaroslav Holeček
holecek@spskladno.cz

Poslední úprava: 20. dubna 2020

Obsah

1 Databáze

Databáze je místo, kde jsou uloženy data - informace.
Kromě hloupých databazí - jako např. napsat si informace na papír, nebo do obyčejného textového souboru, nebo mnoho informací do Excelové tabulky - exitují i „chytřejší“ databáze - např. MySQL, MS-SQL, Oracle SQL.
Kromě samotných informací je v těchto databázích ještě tak zvaný SŘBD (Systém řízení báze dat) - anglicky RDBMS (Relation database managment system), který umí s uloženými informacemi pracovat - např. ukládat, vyhledávat, provádět výpočty.

2 SQL

2.1 Odkazy

Vyzkoušejte si různé dotazy v příkladech na:
`https://www.w3schools.com/sql/default.asp`

2.2 SQL

Jazyk SQL je **dotazovací** jazyk, pomocí kterého si můžeme „povídat“ s chytřejšími databázemi. V dotazech napsaných v SQL **neříkáme**, **jak** se má požadovaný výsledek získat, ale pouze **zapíšeme co chceme za výsledek**.

2.3 CREATE TABLE - vytvoření tabulky

Tabulku v databázi vytvoříme pomocí klíčových slov CREATE TABLE.

```
1  CREATE TABLE nazev_tabulky(sloupec1 DATOVY_TYP_1, sloupec2 DATOVY_TYP_2, ...);
2
3  CREATE TABLE Auto(
4  id INTEGER,
5  spz TEXT,
6  pocet_sedadel INTEGER,
7  max_rychlost INTEGER,
8  nosnost INTEGER,
9  nutna_kvalifikace TEXT,
10 datum_vyroby DATE
11 );
```

Zdrojový kód 1: CREATE TABLE

řádek 3: Dotaz si můžeme pro lepší čitelnost rozepsat na více řádků

Při vytváření tabulky **musíme zadat**:

- Název tabulky
- Názvy jednotlivých sloupečků (atributy)
- Datový typ jednotlivých sloupečků

Dále **můžeme ke sloupečkům přidat**:

- Vlastní klíč (Primary key - PK)
- Cizí klíč (Foreign key - FK)
- NOT NULL - hodnota nesmí být prázdná (nevyplněná)
- UNIQUE - hodnota se v tabulce (v tomto sloupečku) nesmí opakovat
- AUTOINCREMENT - SŘBD („databáze“) sama přiřadí hodnotu - o jedna vyšší, než je doposud nejvyšší
- DEFAULT - pokud není hodnota zadaná, uloží se tato nastavená

```
1  CREATE TABLE Auto(
2  id INTEGER PRIMARY KEY AUTOINCREMENT,
3  spz TEXT NOT NULL UNIQUE,
4  pocet_sedadel INTEGER NOT NULL,
5  max_rychlost INTEGER,
6  nosnost INTEGER,
7  nutna_kvalifikace TEXT NOT NULL DEFAULT 'B',
8  datum_vyroby DATE
9  );
```

Zdrojový kód 2: CREATE TABLE

řádek 2: Vlastní klíč - může být jen jeden v tabulce
AUTOINCREMENT - automaticky přiřazuje hodnotu (vždy nejvyšší+1)

řádek 3: UNIQUE - v tabulce se nemůže opakovat spz (nemohou být dvě stejné)
NOT NULL - spz musí být vyplněná

řádek 4: NOT NULL - počet sedadel musí být vyplněný

řádek 7: NOT NULL - nutná kvalifikace musí být vyplněná
DEFAULT - pokud nezadáme hodnotu doplní se „B“
pozn.: Protože je sloupec nastavený na NOT NULL, nedovolí nám databáze vložit NULL manuálně

2.4 ALTER TABLE

ALTER TABLE slouží k úpravě tabulky poté, co jsme ji již vytvořili - a nechceme ji mazat a vytvářet znovu (to bychom přišli o všechna uložená data...).
Pomocí ATLER TABLE můžeme s tabulkou provádět libovolné úpravy - přidávat a mazat sloupečky, měnit datové typy, přidávat cizí klíče, ...

řádek 10: Přidáme zapomenutý sloupeček.

```
1  CREATE TABLE Auto(
2  id INTEGER PRIMARY KEY AUTOINCREMENT,
3  spz TEXT NOT NULL UNIQUE,
4  pocet_sedadel INTEGER NOT NULL,
5  max_rychlost INTEGER,
6  nosnost INTEGER,
7  nutna_kvalifikace TEXT NOT NULL DEFAULT 'B'
8  );
9
10 ALTER TABLE Auto ADD datum_vyroby DATE;
```

Zdrojový kód 3: ALTER TABLE

2.5 Datové typy

V databázích existuje mnoho datových typů. Na naší úrovni se jimi nebudeme příliš zabývat.
Použití vhodného datového typu zefektivňuje chod databáze - získáme rychleji výsledek, data zaberou méně paměti.
Přehled datových typů např.: zde `https://www.w3schools.com/sql/sql_datatypes.asp`
Je dobré vědět že v databázích existují i datové typy **DATE**, **TIME**, **TIMESTAMP**, apd.

2.6 Další vlastnosti sloupečků

Následující vlastnosti nejsou „nutné“ (databáze bude fungovat i bez nich), ale je nanejvýš vhodné je používat. Vyhneme se tak spoustě chyb a nepříjemností, které bychom museli řešit jinak.

2.6.1 Vlastní klíč, Primary key

Vlastní klíč slouží k **jednoznačné identifikaci řádku** v tabulce. Ve sloupečku, který označíme jako „Primary key“, se nesmí žádná hodnota opakovat - to za nás pohlídá sama databáze (SŘBD). A pokud řekneme, že chceme řádek s konkrétní hodnotou v tomto sloupečku - dostaneme buď právě jeden řádek, nebo žádný (pokud takový řádek neexistuje). Nikdy při použití hodnoty Vlastního klíče nemůžeme dostat jako výsledek více než jeden řádek.
Jako „Vlastní klíč“ můžeme označit i více sloupečků dohromady. V tabulce se potom nebude opakovat řádek, který má hodnoty ve všech těchto sloupečcích stejné.

```
1 CREATE TABLE Clovek(
2 id INTEGER PRIMARY KEY AUTOINCREMENT,
3 jmeno TEXT NOT NULL,
4 prijmeni TEXT NOT NULL,
5 datum_narozeni DATE,
6 kvalifikace TEXT
7 );
```

řádek 2: id jako Vlastní klíč - téměř vždy budeme mít v tabulce sloupeček id a nastavíme ho jako Vlastní klíč

Zdrojový kód 4: Primary key - id

Ukázka, jak nastavit PK přes více sloupečků. Použijeme tak zvanou CONSTRAINT (omezení, podmínku). Toto omezení pojmenujeme a nastavíme, co má omezovat - zde PRIMARY KEY na sloupečky „jmeno, prijmeni“

```
1 CREATE TABLE Clovek(
2 jmeno TEXT NOT NULL,
3 prijmeni TEXT NOT NULL,
4 datum_narozeni DATE,
5 kvalifikace TEXT,
6 CONSTRAINT PK_Clovek PRIMARY KEY (jmeno,prijmeni)
7 );
```

řádek 6: Dva sloupečky „jmeno“ a „prijmeni“ jako Vlastní klíč - v tabulce tak nemůžou být dvě osoby se stejným jménem a příjmením. Uvažme, zda je to dobrý nápad... CONSTRAINT se jmenuje „PK_Clovek“

Zdrojový kód 5: Primary key - jméno a příjmení

2.6.2 Cizí klíč, Foreign key

Pomocí „Cizího klíče“ řekneme databázi, že **hodnoty** uložené **v tomto sloupečku** jsou **převzaté** z jiného sloupečku v jiné tabulce.
Databáze (SŘBD) se nám za to odmění tím, že bude hlídat, zda v něm nemáme hodnotu, která v druhé tabulce neexistuje. Také můžeme nastavit, co se má stát, když některou hodnotu chceme smazat a máme na ní navázanou hodnotu (Cizí klíč) v jiné tabulce - možností je několik (nepůjde smazat, smaže se řádek v obou tabulkách).

```
1  CREATE TABLE Kvalifikace(
2  id INTEGER PRIMARY KEY AUTOINCREMENT,
3  oznaceni TEXT NOT NULL,
4  popis TEXT
5  );
6
7
8  CREATE TABLE Clovek(
9  id INTEGER PRIMARY KEY AUTOINCREMENT,
10 jmeno TEXT NOT NULL,
11 prijmeni TEXT NOT NULL,
12 datum_narozeni DATE,
13 kvalifikace INTEGER FOREIGN KEY REFERENCES Kvalifikace(id)
14 );
```

řádek 13: sloupec „kvalifikace“ nastavíme jako „Cizí klíč“, ve kterém jsou hodnoty ze sloupce „id“ z tabulky Kvalifikace. Vlastní klíč „id“ v tabulce Kvalifikace volíme záměrně - zapsat do sloupečku „kvalifikace“ můžeme jen jednu hodnotu a proto musíme zajistit, aby byla jen jedna možná hodnota v „id“.

Zdrojový kód 6: Foreign key

2.6.3 NOT NULL - neprázdný

Označuje sloupeček, ve kterém nemůže být uložena hodnota NULL - tedy „prázdno“. Viz. 2.

2.6.4 AUTOINCREMENT - automatická hodnota

V tomto sloupečku se automaticky přiřadí hodnota. Databáze zjistí nejvyšší hodnotu v tomto sloupečku, přičte jedna a výsledek vloží na nové místo. Viz. 2.

2.6.5 UNIQUE

V takto označeném sloupečku se nesmí žádná hodnota opakovat - tedy musí být v tablce nejvýše jednou. Viz. 2.
Lze nastavit i na více sloupečků. Potom nemůže být v tabulce stejná tato dvojice, ale hodnota v každém sloupečku zvlášť se opakovat může. Takovou vlastnost zapíšeme pomocí CONSTRAINT - Viz. 5.

2.6.6 DEFAULT

Pokud nastavíme sloupečku DEFAULT hodnotu a poté při vkládání záznamu nezapíšeme, jaká hodnota se má do tohoto sloupečku vložit; vloží se do sloupečku tato nastavená hodnota. Viz. 2.

2.7 CONSTRAINT - omezení, podmínky

CONSTRAINT využejeme ve chvíli:

1. Do tabulky chceme vložit nějaké omezení (např. NOT NULL, UNIQUE) až poté, co jsme ji již vytvořili (např. jsme na něj zapomněli) a nechceme celou tabulku mazat a vytvářet znovu (protože v ní už máme spoustu dat).
2. Chceme nastavit omezení přes více sloupečků (např. UNIQUE - unikátní kombinace dvou sloupečků (třeba jméno a příjmení)).
3. Chceme nějaké takové omezení pojmenovat.

Můžeme ho zapsat přímo při vytváření tabulky:

```
1  CREATE TABLE Clovek(
2  jmeno TEXT NOT NULL,
3  prijmeni TEXT NOT NULL,
4  datum_narozeni DATE,
5  kvalifikace TEXT,
6  CONSTRAINT PK_Clovek PRIMARY KEY (jmeno,prijmeni),
7  CONSTRAINT nechceme_dvojcata UNIQUE (datum_narozeni)
8  );
```

Zdrojový kód 7: CONSTRAINT - přímo

řádek 6: Dva sloupečky „jmeno“ a „prijmeni“ jako Vlastní klíč - v tabulce tak nemůžou být dvě osoby se stejným jménem a příjmením. Uvažme, zda je to dobrý nápad... CONSTRAINT se jmenuje „PK_Clovek“

řádek 7 datum narození bude v tabulce unikátní - těžko říct, k čemu by to bylo ve skutečnosti dobré...

Nebo až později:

```
1  CREATE TABLE Clovek(
2  jmeno TEXT NOT NULL,
3  prijmeni TEXT NOT NULL,
4  datum_narozeni DATE,
5  kvalifikace TEXT
6  );
7
8  ALTER TABLE Clovek ADD CONSTRAINT PK_Clovek PRIMARY KEY (jmeno,prijmeni);
9  ALTER TABLE Clovek ADD CONSTRAINT nechceme_dvojcata UNIQUE (datum_narozeni);
```

Zdrojový kód 8: CONSTRAINT - ALTER

řádek 8: Dva sloupečky „jmeno“ a „prijmeni“ jako Vlastní klíč - v tabulce tak nemůžou být dvě osoby se stejným jménem a příjmením. Uvažme, zda je to dobrý nápad... CONSTRAINT se jmenuje „PK_Clovek“

řádek 9: datum narození bude v tabulce unikátní - těžko říct, k čemu by to bylo ve skutečnosti dobré...

2.8 SELECT - získání dat z databáze

`https://www.w3schools.com/sql/sql_select.asp`

Klíčové slovo SELECT je hlavním slovem pro získávání dat z databáze.

```
1  SELECT sloupec1, sloupec2, ... FROM nazev_tabulky;
2  SELECT * FROM nazev_tabulky;
3
4  SELECT spz, pocet_sedadel, znacka FROM Auto;
5  SELECT * FROM Auto;
```

Zdrojový kód 9: SELECT

řádek 1,4: Z tabulky zobrazíme jen některé sloupce

řádek 2,5: Pomocí * Zobrazíme všechny sloupce v tabulce

2.9 SELECT DISTINCT - unikátní řádky

Pomocí SELECT DISTINCT zobrazíme pouze unikátní řádky - tedy se nám ve výsledky nebude opakovat žádný řádek.

```
1  SELECT DISTINCT sloupec1, sloupec2, ... FROM nazev_tabulky;
2
3  SELECT DISTINCT znacka FROM Auto;
```

Zdrojový kód 10: SELECT DISTINCT

řádek 3: Zobrazí všechny značky aut v tabulce (každou značku jen jednou - i když je v tabulce více aut se stejnou značkou)

2.10 WHERE - podmínka na řádky

Pomocí SELECT jsme z tabulky vybírali některé sloupce pomocí WHERE pak můžeme zobrazit jen některé řádky - za slovo WHERE napíšeme podmínku, kterou musí řádek splňovat, aby se zobrazil ve výsledku.

```
1 SELECT * FROM nazev_tabulky WHERE podmínka;
2
3 SELECT * FROM Auto WHERE maxRychlost > 200;
```

Zdrojový kód 11: WHERE

řádek 3: Zobrazí jen auta s maximální rychlostí větší než 200

2.10.1 AND, OR, NOT

`https://www.w3schools.com/sql/sql_and_or.asp`
Za slovem WHERE můžeme podmínky, podle kterých řádek zobrazíme nebo ne, spojovat známými logickými funkcemi AND, OR, NOT

2.11 ORDER BY - seřazení výsledku

Často chceme, aby se nám výsledek zobrazil seřazený (např. podle abecedy, nebo velikosti čísel). Toho dosáneme pomocí ORDER BY a zadánim, podle kterého sloupce v tabulce se má výsledek seřadit.

```
1 SELECT * FROM nazev_tabulky ORDER BY sloupec;
2
3 SELECT * FROM Auto ORDER BY maxRychlost;
4 SELECT * FROM Auto ORDER BY maxRychlost ASC;
5 SELECT * FROM Auto ORDER BY maxRychlost DESC;
```

Zdrojový kód 12: ORDER BY

řádek 3,4: Zobrazí všechna auta seřazená podle maximální rychlosti **vzestupně** (od nejpomalejšího)
řádek 5: Zobrazí všechna auta seřazená podle maximální rychlosti **sestupně** (od nejrychlejšího)

2.12 SUM, MIN, MAX, AVG - agregační funkce

Databáze umí s daty provádět i jednoduché (resp. často používané) početní operace. Takovými operacemi jsou např. sečtení všech hodnot, minimum, maximum, průměr.
Tyto oprerace jsou vždy prováděny na jednom sloupci a pouze z výsledku, který by se zobrazil, pokud bychom tyto funkce nepoužili - tedy pokud omezíme počet řádků ve výsledku pomocí WHERE, bude se např. součet počítat pouze z těchto vybraných řádku.

```
1 SELECT SUM(sloupec1), MIN(sloupec2), MAX(sloupec1), AVG(sloupec2) FROM nazev_tabulky;
2
3 SELECT SUM(povinne_pojisteni), MIN(spotreba), MAX(rychlost), AVG(nosnost) FROM Auta;
4 SELECT MIN(spotreba) FROM Auta WHERE nosnost > 5000;
```

Zdrojový kód 13: SUM, MIN, MAX, AVG

řádek 3: Zobrazí 4 čísla - součet povinných ručení všech aut,
spotřebu auta s nejmenší spotřebou
rychlost auta s nejvyšší rychlostí a
průměrnou nosnost všech aut
řádek 4: Zobrazí spotřebu auta s nejmenší spotřebou, ale jen mezi těmi, která mají nosnost větší než 5000.

2.13 Vnoření dotazů

Často potřebujeme využít v jednom dotazu výsledek nějakého jiného dotazu. V SQL můžeme dotazy vnořovat. To znamená, že místo některé části dotazu (např. čísla, se kterým porovnáváme) můžeme zapsat celý nový dotaz.

```
1 SELECT sps, spotreba FROM Auta
2 WHERE nosnost > (SELECT AVG(nosnost) FROM Auta);
```

Zdrojový kód 14: Vnořené dotazy

řádek 2: Porovnáváme nosnost jednotlivých aut s průměrnou (AVG) nosností, která se spočítá ze všech aut.
„Vnitřní“ SELECT, který počítá průměrnou nosnost, můžeme spustit i samostatně - je to zcela plnohodnotný dotaz - jen použitý uvnitř jiného dotazu.

2.14 JOIN - spojování tabulek při zobrazení

Obvykle jsou v databázi informace uloženy ve více tabulkách. Často potřebujeme získat informaci, která je rozložena do více tabulek (jedna část informace je v jedné tabulce a druhá část v druhé.)

V jedné tabulce máme jméno a kvalifikaci řidiče. Ve druhé tabulce máme spz auta a kvalifikaci nutnou pro jeho řízení. Chtěli bychom, seznam spz aut a jmen lidí, kteří je můžou řídit

Takový výsledek získáme tak, že dvě tabulky **spojíme** pomocí JOIN.
Při spojování tabulek musíme zadat názvy dvou tabulek a sloupec v jedné i druhé tabulce. Na těchto zadaných sloupcích, se tabulky „překryjí“. Řádky z jedné tabulky a druhé tabulky, ve kterých námi zadané sloupece splňují náš požadavek (obykle aby se hodnoty rovnaly), se spojí do jednoho dlouhého řádku se sloupečky z obou dvou tabulek.
Získáme tak jednu velkou tabulku, která zahrnuje sloupečky i z první i z druhé tabulky
Tato výsledná tabulka není uložena nikde v databázi - „existuje“ pouze v našem dotazu - ale můžeme s ní v dotazu zacházet tak, jako by to byla normální tabulka z naší databáze (a nevznikla spojením dvou skutečných tabulek). Můžeme z ní tedy vybrat jen sloupce, které nás zajímají 9, můžeme vybrat jen některé řádky 11.

```
1 SELECT jmeno, spz FROM
2 Auta INNER JOIN Lide ON Auta.potrebna_kvalifikace = Lide.kvalifikace;
```

Zdrojový kód 15: Jednoduchý JOIN

řádek 2: Tento řádek vytváří novou (velkou) tabulku (spojení dvou tabulek). Je na stejném místě v dotazu, jako obyčejná tabulka. Také se s touto speciální, spojenou, velkou tabulkou pracuje úplně stejně, jako bychom jí měli v databázi uloženou (a nevznikla spojením dvou skutečných tabulek).

Pokud máme dvě tabulky, máme více možností, jak zkombinovat jejich řádky. Sloupce budou

ve výsledku vždy všechny z obou tabulek. Lišit se může, které řádky budeme mít ve výsledku a které ne.
Tyto možnosti vychází z toho, že v každé tabulce mohou být řádky, které:

1. „pasují“ jen na **jeden** řádek druhé tabulky
2. „pasují“ na **více** řádků z druhé tabulky
3. „nepasují“ na **žádný** řádek z druhé tabulky

Pokud řádek z jedné tabulky „pasuje“ na více řádků z druhé tabulky, ve výsledku bude zobrazeno řádků více - pro každou tuto dvojici jeden. Ve výsledku se tedy bude opakovat první záznam vícekrát, ale bude k němu připojen vždy jiný záznam z druhé tabulky.

2.14.1 INNER JOIN

Ve výsledku se zobrazí pouze řádky, které k sobě „pasují“ - tedy mají odpovídající záznam v obou spojovaných tabulkách.

2.14.2 LEFT JOIN

Ve výsledku se zobrazí veškeré řádky z první (levé) tabulky. K těmto řádkům se připojí odpovídající řádky z druhé tabulky.
Pokud nějaký řádek z **levé** tabulky nemá žádného „parťáka“ v pravé tabulce - **bude ve výsledku** přesto zobrazen.
Pokud máme v **pravé** tabulce řádek, který nemá „parťáka“ v levé tabulce, **ve výsledku nebude**.
Samozřejmě, pokud má některý řádek více „partáků“, bude ve výsledku vícekrát - jednou s každým odpovídajícím řádkem z druhé tabulky.

2.14.3 RIGHT JOIN

Je stejný jako LEFT JOIN, jen pro druhou tabulku.

2.14.4 FULL JOIN, FULL OUTER JOIN

Je kombinací LEFT JOIN a RIGHT JOIN. Ve výsledku se tedy zobrazí veškeré řádky z levé tabulky i veškeré řádky z pravé tabulky. Ty, které k sobě „pasují“ se samozřejmě spojí. Zobrazí se ale i ty, které k sobě žádného „parťáka“ nemají - ať už z levé tabulky, nebo z pravé tabulky.