**16. Databáze - charakteristika, druhy databází, jejich princip a použití, SŘBD, DBS**

**Databáze** **je systém souborů s pevnou strukturou záznamů.** Tyto soubory jsou mezi sebou navzájem propojeny pomocí klíčů. **V širším smyslu jsou součástí databáze i** [**softwarové**](https://cs.wikipedia.org/wiki/Software) **prostředky,** které umožňují manipulaci s uloženými daty a přístup k nim. **Tento software se nazývá** [systém řízení báze dat](https://cs.wikipedia.org/wiki/Syst%C3%A9m_%C5%99%C3%ADzen%C3%AD_b%C3%A1ze_dat) (SŘBD). Běžně se označením *databáze* – v závislosti na kontextu – myslí jak uložená data, tak i software (SŘBD).

**Databázové modely** nám souhrnně říkají, jakým způsobem jsou data organizována a jakým způsobem s nimi můžeme pracovat, resp. jak jsou uložena a reprezentována.

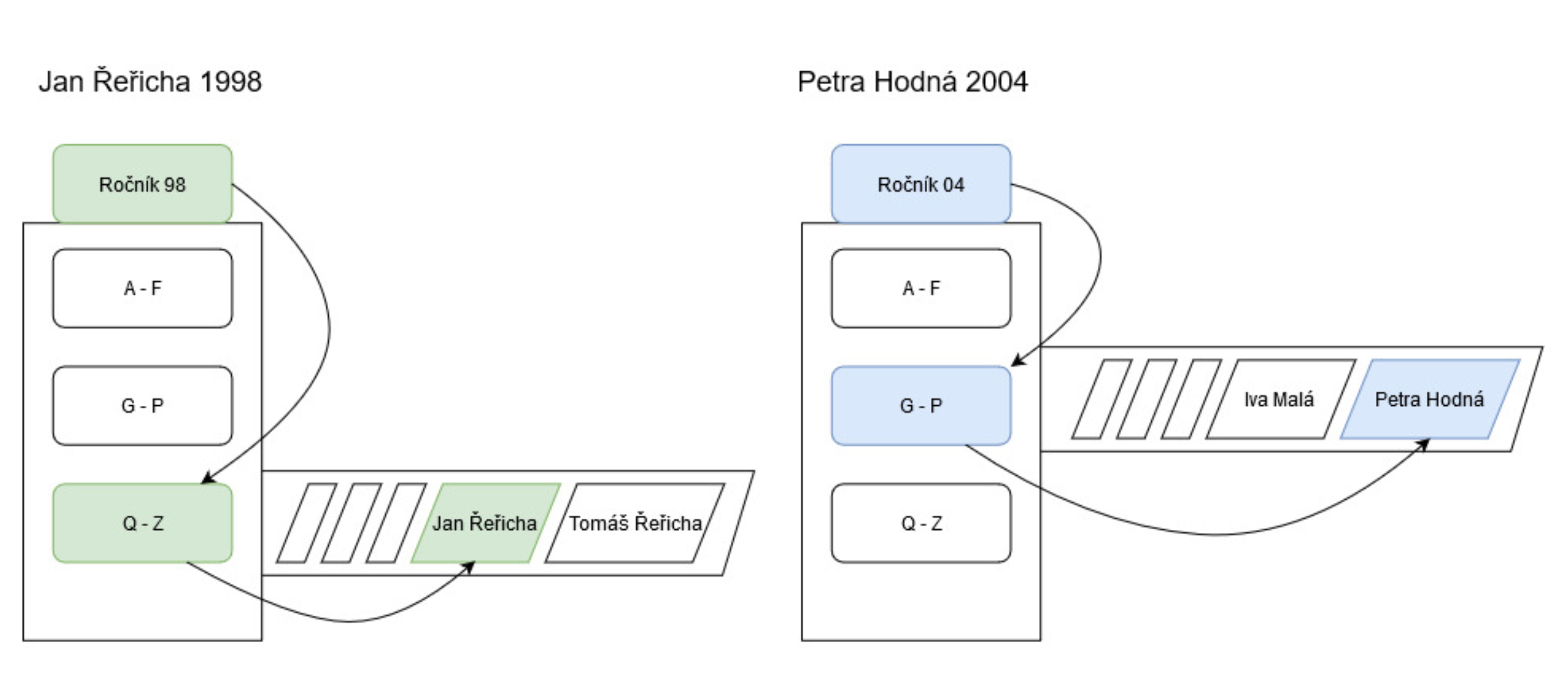
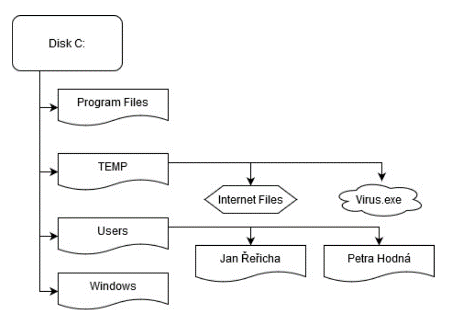
Druhy databází **Hierarchická databáze** (=stromová datová struktura) je datový model, ve kterém jsou data uspořádána ve stromové struktuře. **Je to první z datových modelů, který byl v minulosti hojně**

**využíván v praxi.** Nejznámějším hierarchickým systémem řízení báze dat byl systém od společnosti [IBM](https://cs.wikipedia.org/wiki/IBM). Základní konstrukty: **Věta** – obdoba relace v relačním datovém modelu. Každá věta je definována svými **atributy**. Oproti relačnímu modelu neexistuje omezení při tvorbě atributu – atributy nemusí být atomické (není zde zaveden pojem [normalizace](https://cs.wikipedia.org/wiki/Normalizace_datab%C3%A1ze) datové základny), atributy mohou být i vícerozměrnými strukturami, jako jsou například [pole](https://cs.wikipedia.org/wiki/Pole_(datov%C3%A1_struktura)). **Vlastnicko-členský vztah** (také vztah rodič–dítě) – vztah kardinality 1 : N mezi dvěma větami; věta na straně jedna se nazývá *vlastník*, věta na straně N se nazývá *člen*.

* hierarchické schéma má jeden **kořen**, který není člen v žádném vztahu
* každá věta, kromě kořene, **je členem** **právě v jednom vztahu**
* každá věta může být **vlastníkem 1 až *n* počtu vět**
* věta, která **není vlastníkem v žádném vztahu, se nazývá list**

Hierarchický model je schopen bezproblémově modelovat pouze vztahy kardinality 1 : N. **Vztahy kardinality M : N je možné modelovat za využití dvou vztahů 1 : N za pomocí tzv. *směrníkových vět***, jedná se o speciální případ „virtuální věty“, která obsahuje klíče z obou vztahů 1 : N a binární ukazatele na umístění vztahů v hierarchickém schématu.

Data jsou uložena postupně v hierarchii, např. kartotéka u lékaře. Data hledáme v postupu: Ročník > Písmeno příjmení > Jméno a příjmení > Konkrétní datum narození v případě shody jména:



**Síťová databáze** =je databázový model podobný hierarchickému databázovému modelu, který byl po dlouhou dobu takřka výhradně užívaným databázovým modelem. Na rozdíl od hierarchického databázového modelu **poskytuje navíc vztahy více ku více**, tedy jedna entita může mít více otců. Tato datová koncepce však byla **překonána**[**relačním konceptem** databáze](https://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze). Také navíc umožňuje [**rekurzi**](https://cs.wikipedia.org/wiki/Rekurze), tedy entita může být otcem svému otci. Nevýhodou využívání síťové databáze je nepružnost a z toho vyplývající obtížná změna její struktury.

* zdokonalení hierarchické koncepce, přímá návaznost na hierarchickou koncepci
* síťová struktura se modeluje pomocí tzv. Bachmanových diagramů (Bachmann tvůrce koncepce)

Základní konstruktory: **Věta** je souhrn vzájemně souvisejících datových položek.

**Set** je vztah mezi dvěma větami kardinality 1:N.

* je definován jménem, vlastnickou a členskou větou.
* ta na straně 1 se jmenuje věta vlastnická a na straně N věta členská.
* set má své výskyty, výskyt setu je dán výskytem konkrétního vlastníka a výskytem jeho členů

**Typy členství v setu**[

1. **Automatické** – výskyt členské věty nově vstupující do databáze je automaticky připojen k odpovídajícímu výskytu v setu.

povinné=výskyt členské věty **nemůže** existovat v databází, aniž je přiřazen k výskytu určité vlastnické věty

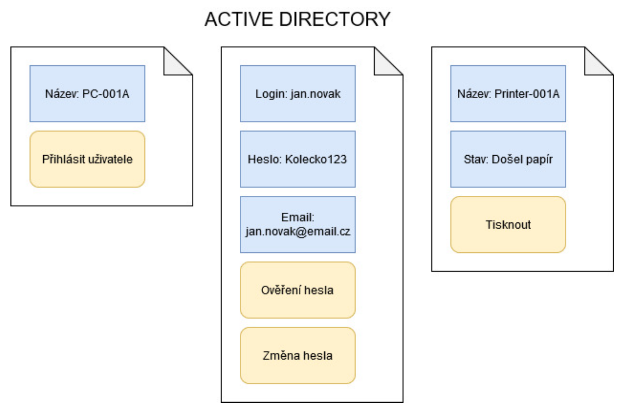
volitelné=výskyt členské věty **může** existovat v databází, aniž je přiřazen k výskytu určité vlastnické věty

1. **Manuální** - manuálně můžeme volně přesouvat výskyt věty do příslušného výskytu setu.
   1. povinné (viz výše) 2. volitelné (viz výše)

* singulární set – takový set, ve kterém je virtuální vlastník systém a tento set má pouze jeden výskyt členských vět (kompromisní prostředek, který umožňuje vložit do databáze plochou strukturu).
* vícečlenský set – set, který může mít více vět (nikoliv výskytu vět) za své členy
* rekurzivní set – pro řešení tzv. konceptuální smyčky, tzn. že daná **věta je současně vlastníkem i členem** síťové modelování
* 1:n vyjádřeny stavebním prvkem setu, m:n se dá vyjádřit pomocí vazební věty
* rekurzivní-nelze modelovat přímo, řeší se pomocí vazební (prázdné) věty

Příklad: Stejná struktura jako jsou složky v PC. Data mohou být rovněž organizována heirarchicky, ale zároveň složky Program Files a Users nemají nadřazenou složku a jsou si postavením rovny, proto jde o strukturu síťovou. Důležitým aspektem je rovněž fakt, že jeden rodič může mít více potomků: viz obrázek výše.

**Relační databáze** - viz otázka 17

**Objektová**[**databáze**](https://cs.wikipedia.org/wiki/Datab%C3%A1ze) je databázový řízený systém, ve kterém je informace reprezentována ve formě objektu a používá se v objektově orientovaném programování. Objektové databáze se odlišují od [relačních databází](https://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze), které jsou tabulkové. Také existují objektově relační databáze, které jsou hybridem dvou předchozích přístupů. Většina objektových databází navíc nabízí různé typy dotazovacích jazyků, které dovolují objektům používat způsob deklarativního programování. To je oblast objektových dotazovacích jazyků, integrací dotazů a navigačních rozhraní. Přístup k datům může být rychlejší, protože objekty můžou být vyhledány přímo bez hledání přes ukazatel. Jinou možností rozlišování mezi produkty je způsob zadání diagramu, podle kterého je databáze je definovaná. Základní podmínkou je, že programovací jazyk a databázový diagram mají stejnou definici. Mnoho objektových databází podporuje **systém správy verzí.** Nějaký objekt může být představen jako sada všech verzí. Kromě toho se různé verze objektů mohou chovat jako samy objekty. Efektivita takové databáze je dobrá, kde je potřeba obsahovat velkou částku informaci v jednom elementu. Například banka může poskytnou klientovi informaci o transakcích, záznamy uživatele, … Porovnání s relační databází: Objektové databáze uchovávají komplexní data a vztahy mezi daty, bez mapování vztahů řádků a sloupců, což je činí velmi pohodlnými pro aplikaci, pracující s komplexními daty. **Objekty mají relaci N:N (vztah many-to-many) a jsou přístupné přes ukazatele**. Ukazatele jsou připojeny k objektům pro provozní vztahů. Dalším významným rozdílem oproti relačním databázím je, že **můžou být programované s malými změnami bez vlivu na celý systém**. Nejužívanější objektovou databází je např. **ActiveDirectory ve Windows.** Kde objekty mohou představovat *Uživatelské účty* nebo *Počítače aj*. Každý z objektů má své vlastnosti a metody:

* **Uživatelské účty**

Vlastnosti: název, login, heslo Metody: ověření hesla, změna hesla

* **Počítače**

Vlastnosti: název, ip adres Metody: přihlášení uživatele

* **Tiskárny**

Vlastnosti: název, typ, stav Metody: tisk

**XML/JSON** Tento databázový model je textovým blobem, který má strukturovaný zápis a využívá se především pro import/export dat, nebo pro sdílení dat skrze aplikační rozhranní (API).

**HIT (Hierarchický klasifikační model)** Datový model HIT je [konceptuální model](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6323) dat, který kombinuje hierarchickou a klasifikační strukturu. Byl vyvinut na Ústavu teoretické a aplikované informatiky Akademie věd České republiky v Praze (rok 1986 Marie Duží, Zlatuška). Byl navržen pro klasifikaci a organizaci biologických druhů v taxonomických systémech. Jeho vývoj probíhal v rámci úsilí o efektivní a strukturované zachycení vztahů mezi biologickými druhy. Datový model HIT se snaží reflektovat reálné vztahy mezi objekty. **Principy modelu HIT:**

1. **Hierarchie:** Model HIT vytváří hierarchickou strukturu, kde objekty (např. organismy) jsou uspořádány do různých úrovní. Každá úroveň představuje určitý druh klasifikace.
2. **Klasifikace:** Objekty jsou klasifikovány do různých kategorií podle jejich vlastností, charakteristik a vztahů. Klasifikace může být založena na různých kritériích, jako jsou morfologické rysy, genetická podobnost, ekologické faktory atd.
3. **Unikátní identifikace:** Každý objekt má unikátní identifikátor, který umožňuje jeho jednoznačnou identifikaci v rámci celého systému.
4. **Vzájemné vztahy:** Model umožňuje definovat vztahy mezi objekty, např. nadřazený-podřazený vztah mezi různými úrovněmi klasifikace.

**SŘBD Systém řízení báze dat** ([anglicky](https://cs.wikipedia.org/wiki/Angli%C4%8Dtina) *Database Management System*, **DBMS**) je [softwarové](https://cs.wikipedia.org/wiki/Software) vybavení, které zajišťuje práci s [databází](https://cs.wikipedia.org/wiki/Datab%C3%A1ze), tzn. tvoří rozhraní mezi [aplikačními programy](https://cs.wikipedia.org/wiki/Aplika%C4%8Dn%C3%AD_program) a uloženými daty. Občas se pojem zaměňuje s pojmem databázový systém. DBS = **Databázový systém** **= SŘBD + DB (báze dat)**

Aby mohl být nějaký programový systém označený za DBMS, musí být jednak schopen efektivně pracovat s velkým množstvím dat, ale také musí být schopný řídit (vkládat, modifikovat, mazat) a definovat strukturu těchto perzistentních dat (čímž se liší od prostého [souborového systému](https://cs.wikipedia.org/wiki/Souborov%C3%BD_syst%C3%A9m)). Další vlastnosti:

* **podporu pro definici** [datových **modelů**](https://cs.wikipedia.org/w/index.php?title=Datov%C3%BD_model&action=edit&redlink=1) (například [relační](https://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze), [logický](https://cs.wikipedia.org/wiki/Logika), [objektový](https://cs.wikipedia.org/wiki/Objektov%C3%A1_datab%C3%A1ze))
* **správa**[**klíčů**](https://cs.wikipedia.org/wiki/Index_(datab%C3%A1ze)): vlastní (interně implementované) indexování, dodržování unikátnosti hodnot ve sloupcích, nad kterými je definován primární klíč implementace [fulltextového](https://cs.wikipedia.org/wiki/Fulltext) vyhledávání pro fulltextové klíče; implementace [cizích klíčů](https://cs.wikipedia.org/wiki/Ciz%C3%AD_kl%C3%AD%C4%8D), **využití některého jazyka vyšší úrovně pro manipulaci a definici dat (např.**[**SQL**](https://cs.wikipedia.org/wiki/SQL)**)** a vyřešení komunikačního kanálu mezi uživatelem či skriptem a DBMS v tomto jazyku,
* [autentizaci](https://cs.wikipedia.org/wiki/Autentizace) uživatelů a jejich [autorizaci](https://cs.wikipedia.org/wiki/Autorizace) k operacím nad daty (u každého uživatele může být definováno, jaký typ příkazů je oprávněn spouštět), [uložené procedury](https://cs.wikipedia.org/wiki/Ulo%C5%BEen%C3%A9_procedury), [triggery](https://cs.wikipedia.org/wiki/Trigger_(datab%C3%A1ze))
* správu [transakcí](https://cs.wikipedia.org/wiki/Datab%C3%A1zov%C3%A1_transakce), [atomicitu](https://cs.wikipedia.org/wiki/Atomicita" \o "Atomicita) jednotlivých příkazů, robustnost a zotavitelnost po chybách bez ztráty dat
* [integritu dat](https://cs.wikipedia.org/wiki/Integrita_dat); například nepovolením vložení duplicitního řádku s unikátním klíčem nebo řádku s hodnotami [NULL](https://cs.wikipedia.org/wiki/NULL) u sloupců, které NULL být nesmějí
* kanály pro hlášení zpráv po úspěšně vykonaných dotazech, chybových hlášek, varování
* pokročilé funkce, zpožděné zápisy, [profilování](https://cs.wikipedia.org/wiki/Profilov%C3%A1n%C3%AD), statistické informace o běhu dotazů, procesů

**DBS =** je výraz pro systém, který je tvořen spojením dvou částí: BD a SŘBD. Dohromady tvoří jeden neoddělitelný celek, který je pak provozován např. na serveru.

**BD - Báze dat**(Angl. DB - Database) je souhrn veškerých dat, které máme k dispozici. Lze říci, že jde o datový základ, se kterým následně pracujeme. Typickou bází dat v kartotéce u lékaře je ona samotná kartotéka, šuplíky a šanony se složkami pacientů. Jsou to fyzicky uložená data na nějakém místě, fyzicky nebo elektronicky na paměťovém médiu.

**SŘBD - Systém Řízení Báze Dat** (Angl. DBMS - Database Management System) je systém, který pracuje s bází dat, vykonává nad bází dat operace jako **vyhledávání, úpravu, zápis nebo smazání** (CRUD - Create Read Update Delete). Uživatel komunikuje se SŘBD pomocí zvolené metody, nejčastěji jazykem SQL, pomocí něhož předává SŘBD pokyny, co má provést a SŘBD vykoná tento příkaz s bází dat. Např. vyhledat konkrétního pacienta. SŘBD na základě předaných parametrů vykoná příkaz a předá hledaná data z báze dat. V případě kartotéky u lékaře, je SŘBD zdravotní sestra, která vyhledává, vkládá, aktualizuje a nebo odstraňuje záznamy ze složky pacienta na pokyn lékaře.

**Dělení SŘBD**

**⚫Relační SŘBD** vychází z konceptu propojení entit vazbami-relacemi, nejčastěji Entitně-Relační model. Z pravidla v těchto SŘBD komunikujeme pomocí dotazovacího **jazyka SQL** (Structured Query Language - Strukturovaný dotazovací jazyk). **Zástupci:** MySQL, Microsoft SQL server, MariaDB, Oracle, SAP (HANA), Microsoft Access, PostgreSQLSQLite

**⚫NoSQL SŘBD** Zkratka NoSQL je v dnešní podobě vnímána spíše jako Not-Exactly SQL. Historicky je tento výraz směřován k Non-SQL, tedy Ne-Relační databáze, což vývojem databázových systémů úplně neplatí. I v NoSQL SŘBD, lze využít dotazování pomocí SQL, nicméně nejedná se o typické entitně-relační modely.Jestliže u NoSQL SŘBD zpravidla nevyužíváme relací mezi daty, můžeme pak tyto systémy dělit na další druhy jako: Key-Value - Redis, Memcached, Document Store - MongoDB, Couchbase, Time-Based - InfluxDB, Kdb+, Graph Based - Neo4j, Dgraph, Search Engine (Multi-model) - Elasticsearch, Splunk

**SŘBD (například SQL) zahrňuje:**

**DDL**= Data Definition Language (jazyk pro definici dat): prostředky pro popis dat, sloužící k vytvoření všech definic uživatelských dat potřebných v aplikaci, včetně určení omezujících podmínek. Pomocí příkazů tohoto jazyka lze vytvářet strukturu databáze. Zároveň do této skupiny patří příkazy, které umožňují realizovat případné  změny nebo odstranění existujících objektů. Například: Create Table, Drop Table.

**DML**= Data Manipulation Language (jazyk pro manipulaci s daty): prostředky pro popis algoritmu, které se používají k aktualizaci dat (přidávání, změny a rušení dat) a k výběru dat z databáze na základě kladených požadavků. Například: [Select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329" \o "SELECT), Insert, Delete, Update.

**Other Command** (ostatní příkazy):

DCL=Data Control Language-správa uživatelských rolí a práv (Grant, Revoke)

TCL=Transaction Control Language-správa databázových transakcí (Begin, Commit)

17. Relační databáze - relace, diagramy, druhy klíčů, ACID transakce

**Relační databáze** je [databáze](https://cs.wikipedia.org/wiki/Datab%C3%A1ze) založená na [**relačním modelu**](https://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_model). Často se tímto pojmem označuje nejen databáze samotná, ale i její konkrétní [softwarové](https://cs.wikipedia.org/wiki/Software) řešení. Relační databáze je založena na tabulkách, jejichž řádky obvykle chápeme jako záznamy a eventuálně některé sloupce v nich (tzv. [cizí klíče](https://cs.wikipedia.org/wiki/Ciz%C3%AD_kl%C3%AD%C4%8D)) chápeme tak, že uchovávají informace o [relacích](https://cs.wikipedia.org/wiki/Relace_(matematika)) mezi jednotlivými záznamy v matematickém slova smyslu. Základním konstruktorem relačních databází jsou **entity (=**[**databázové tabulky**](https://cs.wikipedia.org/wiki/Tabulka_(datab%C3%A1ze))**),** což jsou dvourozměrné struktury tvořené záhlavím a tělem. Jejich sloupce se nazývají atributy, řádky tabulky jsou pak [**záznamy**](https://cs.wikipedia.org/wiki/Z%C3%A1znam_(informatika))**.** Atributy mají určen svůj konkrétní [datový typ](https://cs.wikipedia.org/wiki/Datov%C3%BD_typ) a doménu, což je množina přípustných hodnot daného atributu. Řádek je řezem přes sloupce tabulky a slouží k vlastnímu uložení dat. Pojem „relační databáze“ souvisí s teorií množin. Každá tabulka totiž realizuje [podmnožinu](https://cs.wikipedia.org/wiki/Podmno%C5%BEina) [kartézského součinu](https://cs.wikipedia.org/wiki/Kart%C3%A9zsk%C3%BD_sou%C4%8Din) množin přípustných hodnot všech sloupců – [**relaci**](https://cs.wikipedia.org/wiki/Relace_(matematika))**.**

**Entita** je jsoucno, tedy cokoliv co je - existuje. Takové jsoucno jsme schopni popsat pomocí vlastností =atributy. Atributy mají své konkrétní podoby, tj. záznamy v případě databází. Entitou může být např. Nábytek. Vlastností nábytku bude např. název. Konkrétní podobou vlastnosti název je: židle. Nábytek má vlastnost název a tím je židle. Nábytek má vlastnost materiál a tím je dřevo.

**Atribut** je vlastnost entity podstatná z hlediska zkoumané oblasti a její evidence Přiřazuje každé entitě z množiny entit hodnotu z nějaké neznámé množiny. Příklad: Atributem entity Zaměstnanec je jeho osobní číslo, jméno, příjmení, atd...

**Klíčový atribut (= klíč)** je jeden atribut či množina atributů, jejichž hodnoty *jednoznačně určují* (identifikují) každou entitu v množině entit. Příklad: Jako klíčový atribut lze brát například rodné číslo u člověka, osobní číslo u studenta či zaměstnance.

**Relace** je **vztah mezi entitami**. Tento vztah má Kardinalitu (Mocnost), Mandatornost (Povinnost) a Stupeň vztahu. Relace nám určuje, zda mezi entitami je nějaká vazba či nikoliv. Obvykle bývá vyjádřena slovesem, např. má, obsahuje, zapůjčil si, atd. Věta "učitel vyučuje studenta" je vyjádřením vztahu "vyučuje" mezi entitami "učitel" a "student". Příklad: Věta "zákazník zakoupil zboží" vyjadřuje vztah "zakoupil" mezi entitami zboží a zákazník. Potřebujeme-li vědět kdy, za kolik a v jakém množství, pak to jsou již atributy daného vztahu "zakoupil". Vztahy mezi tabulkami (angl. *relationships*) slouží ke svázání dat, která spolu souvisejí a jsou umístěny v různých databázových tabulkách. **Stupeň vztahu:**

***Unární*** *vztah* – relace je spojena sama se sebou. Typickým příkladem je vztah zaměstnance a jeho nadřízeného, kdy nadřízený je také jedním ze zaměstnanců, pročež také může mít nadřízeného. Vztah se realizuje vložením primárního klíče relace *zaměstnanec* ve formě cizího klíče opět do relace *zaměstnanec*.

***Binární*** *vztah* – vztah mezi dvěma relacemi.

***Ternární*** *vztah* – vztah mezi třemi relacemi najednou.

***N-ární*** *vztah* – vztah mezi n-relacemi zároveň.

**Kardinalita vztahu (mocnost)** je maximální počet vztahů instancí, kterých se může entita účastnit. Vyjadřuje, jak mocný vztah mezi entitami je.

***1:1*** používáme, pokud záznamu odpovídá právě jeden záznam v jiné databázové tabulce a naopak. Takovýto vztah je používán pouze ojediněle, protože většinou není pádný důvod, proč takovéto záznamy neumístit do jedné databázové tabulky. Jedno z mála využití je zpřehlednění rozsáhlých tabulek. Jako ilustraci je možné použít vztah řidiče a automobilu – v jednu chvíli (diskrétní časový okamžik) jeden řidič řídí právě jeden automobil a zároveň jeden automobil je řízen právě jedním řidičem.

***1:N*** přiřazuje jednomu záznamu více záznamů z jiné tabulky. Jedná se o nejpoužívanější typ relace, jelikož odpovídá mnoha situacím v reálném životě. Jako reálný příklad může posloužit vztah autobusu a jím cestujícího pasažéra – v jednu chvíli pasažér jede právě jedním autobusem a v jednom autobuse může zároveň cestovat více pasažérů.

***M:N*** umožňuje každému záznamu z jedné tabulky přiřadit libovolný počet záznamů z druhé tabulky, přičemž záznam v druhé tabulce přiřazením k záznamu k první tabulce svou možnost přiřazení „nespotřebuje“, takže jej lze přiřadit k libovolnému počtu záznamů první tabulky. V databázové praxi bývá tento vztah z praktických důvodů nejčastěji realizován kombinací dvou vztahů, a sice 1:N a 1:M, které ukazují do pomocné tabulky, složené z kombinace obou použitých klíčů (jde o třetí, resp. tzv. **vazební tabulku**). Příkladem z reálného života by mohl být vztah výrobku a jeho vlastnosti – výrobek může mít více vlastností a jednu vlastnost může mít více výrobků.

**Povinnost (mandatornost)** vyjadřuje, zda je účast relace ve vztahu povinná nebo volitelná. Jedná-li se o vztah povinný, musíme v dané entitě vyplnil vzájemnou vazbu mezi entitami (záleží na kardinalitě) a nelze bez doplnění tohoto vztahu záznam do entity vložit. Např.: Vztah mezi entitou žák a třída bude povinný, protože nelze zapsat do databáze žáka, aniž by nebyl přiřazen k jakékoliv třídě. Vztah je tedy mandatorní. V případě, že máme firmu, ve které zaměstnanci mohou pracovat pouze pro jedno konkrétní oddělení, pak můžeme tento vztah označit i jako NE-mandatorní, neboť můžeme do databáze zapsat i zaměstnance (ředitel), který nemusí být zařazen do žádného oddělení. V případě, že by tento vztah byl mandatorní, pak musíme vyplnit i název oddělení, jinak bychom zaměstnance nemohli do databáze zapsat

**Druhy klíčů**

**Kandidátní klíč** je atribut nebo skupina atributů, které jednoznačně identifikují záznam v relační tabulce. Kandidátní klíč se může stát primárním klíčem. Ty, které se primárním klíčem nestanou, jsou označovány jako **alternativní** klíče. Např. v relaci Zaměstnanec, která má atributy číslo\_zaměstnance, rodné\_číslo, jméno a příjmení, jsou kandidátními klíči atributy číslo\_zaměstnance a rodné\_číslo. Pokud primárním klíčem zvolíme číslo\_zaměstnance, alternativním klíčem bude rodné\_číslo a naopak.

**Primární klíč** je jednoznačný identifikátor záznamu, řádku tabulky. Primárním klíčem může být jediný sloupec či kombinace více sloupců tak, aby byla zaručena jeho jednoznačnost. Pole klíče musejí obsahovat hodnotu, tzn. nesmí se zde vyskytovat nedefinovaná prázdná hodnota NULL. V praxi se dnes často používají umělé klíče, což jsou číselné či písmenné identifikátory – každý nový záznam dostává identifikátor odlišný od identifikátorů všech předchozích záznamů (požadavek na unikátnost klíče), obvykle se jedná o celočíselné řady a každý novější záznam dostává číslo vždy o jednotku vyšší (zpravidla zcela automatizovaně) než je číslo u posledního vloženého záznamu (číselné označení záznamů s časem stoupá).

**Cizí klíč (= nevlastní)** Slouží pro vyjádření vztahů, relací, mezi databázovými tabulkami. Jedná se o pole či skupinu polí, která nám umožní identifikovat, které záznamy z různých tabulek spolu navzájem souvisí. Použiji-li kardinální klíč v jiné entitě, pak se jedná o klíč cizí, neboť není pro danou entitu vlastní, ale pouze odkazuje do primární entity.

[**Integrita databáze**](https://cs.wikipedia.org/wiki/Integrita_datab%C3%A1ze)znamená, že data v ní uložená jsou konzistentní vůči definovaným pravidlům. Lze zadávat pouze data, která vyhovují předem definovaným kritériím (např. musí respektovat datový typ nastavený pro daný sloupec tabulky, či další omezení hodnot přípustných pro daný sloupec). K zajištění integrity slouží [integritní omezení](https://cs.wikipedia.org/wiki/Integritn%C3%AD_omezen%C3%AD). Jedná se o nástroje, které zabrání vložení nesprávných dat či ztrátě nebo poškození stávajících záznamů v průběhu práce s databází. Typicky je možné zajistit mazání dat, která již ztratila svůj význam (kupř. smažeme-li uživatele z tabulky uživatelů, odstraní se i na něj navázané záznamy v ostatních databázových tabulkách). **Druhy integritních omezení :**

*Entitní integritní omezení* – někdy je mylně považováno za pouhé omezení primárního klíče tabulky (jež má zamezit uložení dat, která neobsahují všechna pole sdružená do klíče, nebo data, jež by v těchto polích byla stejná jako v nějakém jiném, již zapsaném, řádku tabulky). Ve skutečnosti jde o to, aby o jedné entitě nebylo možno do databáze vložit duplicitní záznamy; jedná se tedy o zajištění unikátnosti skutečných identifikátorů reálných objektů (což například umělé automaticky generované ID nezajistí).

*Doménová integritní omezení* – zajišťují dodržování datových typů/domén definovaných sloupcům databázové tabulky.

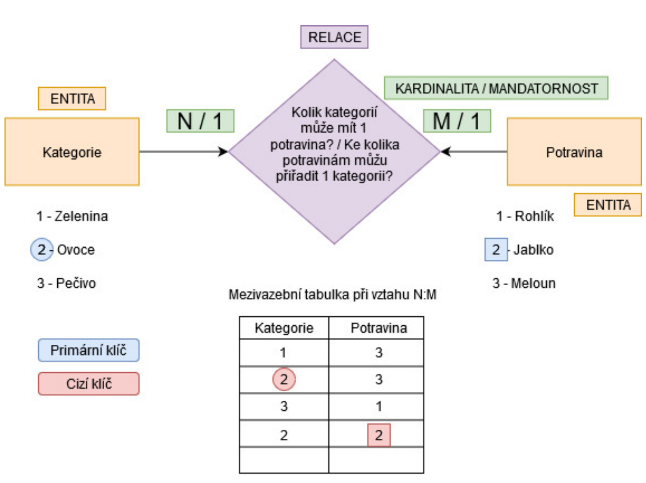
*Aktivní referenční integrita* – definují činnosti, které databázový systém provede, pokud jsou porušena některá pravidla.

*Referenční integritní omezení* – zabývají se vztahy dvou tabulek, kde jejich relace je určena vazbou primárního a cizího klíče.

**Dodržování integritních omezení**

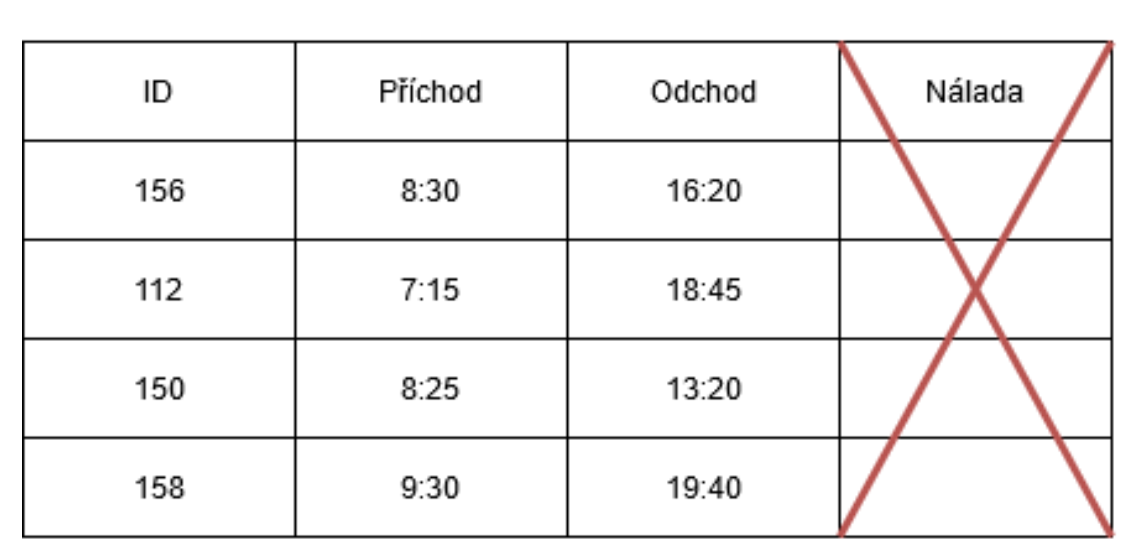
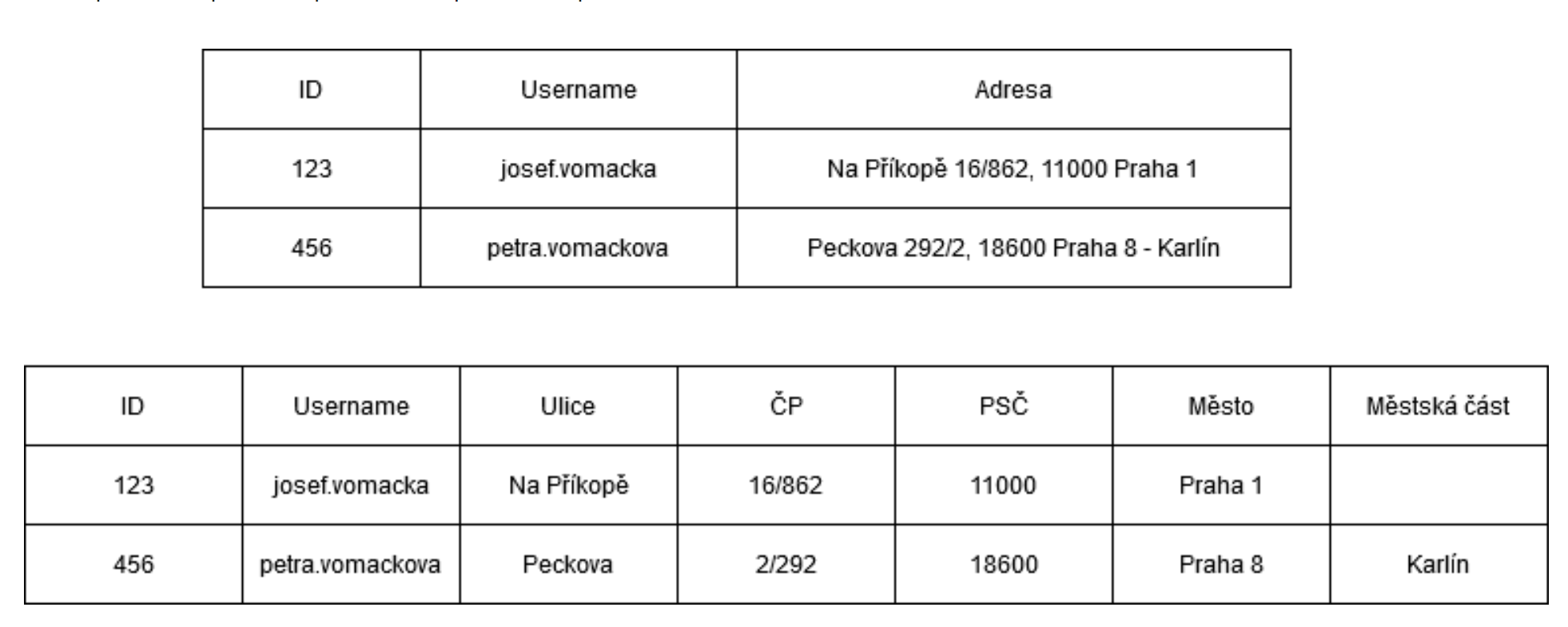
1. umístění jednoduchých mechanismů pro dodržování integritních omezení na straně databázového serveru. Jedná se o nejlepší způsob z hlediska ochrany dat. Uživateli však obvykle přináší delší odezvu systému a nelze vždy zajistit jejich přenositelnost na jiný databázový systém.
2. umístění ochranných mechanismů na straně klienta. Pro komfort a nezávislost na databázovém systému je nejlepší volbou. Nutnost kontrolních mechanismů pro každou operaci může způsobit chyby u aplikací a v případě většího počtu aplikací je potřeba je opravit na více místech.
3. samostatné programové moduly na straně serveru. V moderních databázových systémech jsou pro tento účel implementovány tzv. [triggery](https://cs.wikipedia.org/wiki/Trigger" \o "Trigger). Jedná se o samostatné procedury, které lze spouštět automatizovaně před a po operacích manipulujících s daty. Tento způsob umožňuje implementaci i složitých integritních omezení. Nevýhody opět přináší provádění na serveru, i velmi omezená možnost přenesení na jiný databázový systém.

Možná je i kombinace předchozích variant v závislosti na konkrétních podmínkách. Kontroly integritních omezení se zpravidla provádějí po každé provedené operaci, což snižuje nároky na server.

**Normální formy** jsou soupis několika pravidel, které se dodržují proto, aby při návrhu, resp. [modelování databáze](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6322) nedošlo k chybám, které následně půjde těžko odstranit. Jsou to pravidla, která nám umožňují modelovat databáze dle normy, která je v praxi běžná. Celkem máme 7. normálních forem s tím, že existuje i 0tá forma, která je brána jako samozřejmost. V praxi se modeluje minimálně až do 3.NF

**0. NF - Žádný atribut nesmí být nulový** (viz obrázek vlevo)

Atributem v Entitně-Relačních databází je ve fyzické podobě sloupec v tabulce. Jestliže bude v celé databázi celý sloupec (atribut) nulový, pak nemá význam takový sloupec v databázi navrhovat. Např. máte-li tabulku s docházkou zaměstnanců a navrhnete-li atribut "nálada", který nikdo při příchodu ani odchodu vyplňovat nebude, pak nemá smysl takový atribut při modelování vkládat do databáze.

**

**1. NF - Každý atribut musí obsahovat pouze atomární hodnoty** (viz dva obrázky vpravo)

Žádný sloupec (atribut) tabulky nelze dále dělit na části nesoucí nějakou informaci, jinými slovy prvky musí být atomické (žádný sloupec neobsahuje složené hodnoty). Při zápisu dat do jednotlivých atributů, musíme dbát na to, aby byla data v co atomární podobě. Máme-li např. atribut adresa, mohou zapsaná data v buňce vypadat takto: U*lice 23/454, 14569 Praha 14 - Křižíkov*. Na první pohled je jasné, že pokud bychom chtěli vyhledávat např. všechny uživatele dle města, nelze s takto zapsanými daty dotaz provést, neboť textový řetězec obsahuje více, než pouhé město uživatele. proto musíme data rozdělit na atomární hodnoty a tj. *Ulice | ČP | PSČ | Město | Městská část |*  *Ulice | 23/454 | 14569 | Praha 14 | Křižíkov |*

Pro splnění 1.NF je zapotřebí zajistit i splnění nulté normální formy (0NF).

**Příklad 1**

Klasickým příkladem tabulky porušující první normální formu bývá nejčastěji problém s telefonními čísly, kdy naším cílem je umožnit evidovat pro každou osobu dvě různá telefonní čísla, jak lze vidět v tabulce níže: Podtržené názvy atributů představují [primární klíč](https://cs.wikipedia.org/wiki/Prim%C3%A1rn%C3%AD_kl%C3%AD%C4%8D).

|  |  |  |
| --- | --- | --- |
| **Tabulka Osoba** | | |
| **ID Osoby** | **Jméno** | **Telefonní číslo** |
| 1 | Petr Novák | +420 111 222 333 |
| 2 | Jarmil Hnízdo | +420 123 123 123, +420 123 123 124 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tabulka Osoba** | | | |
| **ID Osoby** | **Jméno** | **Telefonní číslo 1** | **Telefonní číslo 2** |
| 1 | Petr Novák | +420 111 222 333 |  |
| 2 | Jarmil Hnízdo | +420 123 123 123 | +420 123 123 124 |

|  |  |  |
| --- | --- | --- |
| **Tabulka Kontakt** | | |
| **ID Kontaktu** | **ID Osoby** | **Telefonní číslo** |
| 1 | 1 | +420 111 222 333 |
| 2 | 2 | +420 123 123 123 |
| 3 | 2 | +420 123 123 124 |

Tato tabulka ovšem porušuje první normální formu, jelikož sloupec (atribut) telefonní číslo není atomický. Zjevným řešením této situace by mohlo být přidání druhého sloupce pro telefonní číslo, abychom zajistili splnění pravidla atomických atributů: Přestože je tato tabulka formálně správná a již neporušuje pravidlo první normální formy, její návrh je stále problematický. Problém nastane zejména v případě, že bude zapotřebí evidovat čísel více. Rozšiřování tabulky o další sloupce telefonních čísel není často realizovatelné a také se označuje za špatný návrh. Správným řešením je vytvoření nové tabulky a odstranění sloupce telefonních čísel z tabulky osob, jak lze vidět na příkladu níže:

|  |  |
| --- | --- |
| **Tabulka Osoba** | |
| **ID Osoby** | **Jméno** |
| 1 | Petr Novák |
| 2 | Jarmil Hnízdo |

V tomto případě je tedy první normální forma splněna a rozdělením tabulek jsme umožnili bezproblémové přidávání libovolného počtu telefonních čísel pro každou osobu. Všimněme si, že tabulka s kontakty obsahuje navíc kromě primárního klíče také [cizí klíč](https://cs.wikipedia.org/wiki/Ciz%C3%AD_kl%C3%AD%C4%8D), který nám zajišťuje svázání telefonního čísla se správnou osobou.

**2. NF - Všechny neklíčové atributy jsou závislé na kandidátním klíči**

Abychom mohli identifikovat neklíčové atributy jako např. adresa, je třeba, aby tyto atributy měli vzájemnou vazbu na kandidátní klíče, ať už primární nebo cizí klíč. Např. ve výše uvedeném příkladu s adresou. Jestliže budeme chtít znát adresu Josefa Vomáčky, vyhledáme si všechny atributy adresy pomocí primárního klíče s názvem ID 123. Pro splnění 2.NF je zapotřebí zajistit i splnění první normální formy (1NF).

**Příklad 2**, kdy máme tabulku evidující informace o kurzech. Všimněme si toho, že tabulka má složený primární klíč {ID Kurzu, ID Semestru}:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tabulka Záznam** | | | |
| **ID Kurzu** | **ID Semestru** | **Počet míst** | **Jméno kurzu** |
| IT101 | ls 2017 | 100 | Programování |
| IT101 | zs 2017 | 100 | Programování |
| IT102 | ls 2017 | 200 | Databáze |
| IT102 | zs 2017 | 150 | Databáze |
| IT103 | zs 2017 | 120 | Web design |

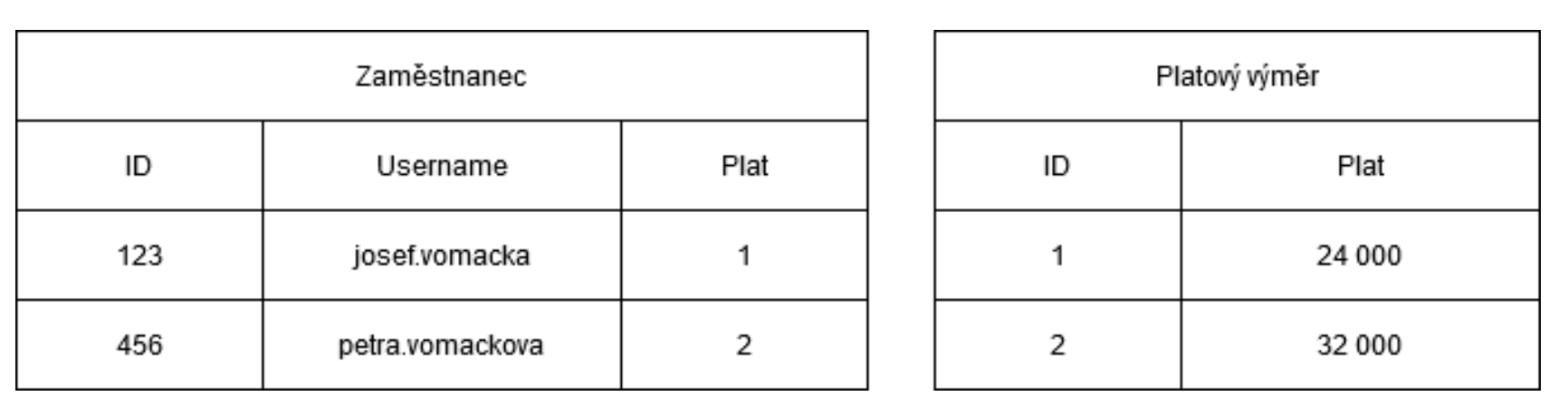
|  |  |  |
| --- | --- | --- |
| **Tabulka Záznam** | | |
| **ID Kurzu** | **ID Semestru** | **Počet míst** |
| IT101 | ls 2017 | 100 |
| IT101 | zs 2017 | 100 |
| IT102 | ls 2017 | 200 |
| IT102 | zs 2017 | 150 |
| IT103 | zs 2017 | 120 |

Tato tabulka porušuje druhou normální formu, jelikož sloupec *Jméno kurzu* není plně závislý na celém primárním klíči. *Jméno kurzu* je závislé na sloupci *ID Kurzu*, ovšem není již závislé na sloupci *ID Semestru*. V takovéto tabulce dochází k redundanci dat, jak lze vidět na opakujících se jménech každého kurzu. Při vypisování již existujících kurzů v budoucnu (v nových semestrech) by docházelo k opakování těchto záznamů. Způsob, jakým lze tento problém vyřešit je dekompozice tabulky a zajištění, že všechny neklíčové atributy dané tabulky budou závislé na celém klíči. Možným řešením může být například následující rozdělení tabulky: Uvedená dekompozice tabulky zajišťuje splnění druhé normální formy. Jediný neklíčový atribut v původní tabulce – *Počet míst* je již závislý na celém složeném primárním klíči, obdobně jako jediný neklíčový atribut *Jméno kurzu* v tabulce Kurz, který je již závislý na jediném primárním klíči nově vytvořené tabulky.

|  |  |
| --- | --- |
| **Tabulka Kurz** | |
| **ID Kurzu** | **Jméno kurzu** |
| IT101 | Programování |
| IT102 | Databáze |
| IT103 | Web design |

**3. NF - Neklíčové atributy jsou na sobě vzájemně nezávislé**

Mezi neklíčovými atributy (sloupci) tabulky neexistují žádné závislosti (vztahy). Typickým příkladem, kdy je tato normální forma porušena je v případě, že chceme např. identifikovat plat Josefa Vomáčky. Dotazujeme se na atribut **jméno** + **příjmení** a hledáme atribut **plat**. Tedy Josef Vomáčka 24.000 Kč. Ale atribut jméno i příjmení jsou neklíčové atributy, proto nelze pomocí těchto atributů vyhledávat další neklíčový atribut tj. plat. Správné řešení:



Pro splnění 3.NF je zapotřebí zajistit i splnění druhé normální formy (2.NF). 3NF neobsahuje tranzitní závislosti. Tranzitivní závislost chápeme jako vztah mezi třemi atributy (např. X, Y, Z), kdy atribut Y je funkčně závislý na atributu X, atribut Z je funkčně závislý na atributu Y, a proto lze implikovat, že atribut Z je také funkčně závislý na atributu X.

**Příklad 3** Mějme následující tabulku evidující jednotlivé filmy, jejich žánry a délku jejich stopáže:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tabulka Film** | | | |
| **ID Filmu** | **ID Žánru** | **Název žánru** | **Délka filmu** |
| 1 | 3 | Dokumentární | 01:20:00 |
| 2 | 2 | Akční | 01:05:00 |
| 3 | 1 | Komedie | 01:50:00 |
| 4 | 2 | Akční | 01:10:00 |
| 5 | 3 | Dokumentární | 01:11:00 |

Na první pohled není s tabulkou nic v nepořádku. Tabulka splňuje druhou normální formu, všechny neklíčové atributy jsou závislé na celém primárním klíči (tedy v tomto případě na atributu *ID Filmu*). Přesto tato tabulka nesplňuje pravidla určená třetí normální formou. V tabulce vlevo vidíme, že primární klíč identifikující každý film rozhoduje o tom, jaký žánr bude zvolen, tedy jinými slovy atribut *ID Žánru* je funkčně závislý na atributu *ID Filmu* (ID Filmu → ID Žánru). Další závislost, kterou můžeme identifikovat je vztah identifikačního čísla žánru a názvu daného žánru, jinými slovy atribut *Název žánru* je funkčně závislý na atributu *ID žánru* (ID Žánru → Název žánru). Jak již z definice tranzitivní závislosti víme, z těchto dvou vztahů plyne i vztah třetí, určující funkční závislost atributu *Název žánru* na primárním klíči *ID Filmu*.

|  |  |  |
| --- | --- | --- |
| **Tabulka Film** | | |
| **ID Filmu** | **ID Žánru** | **Délka filmu** |
| 1 | 3 | 01:20:00 |
| 2 | 2 | 01:05:00 |
| 3 | 1 | 01:50:00 |
| 4 | 2 | 01:10:00 |
| 5 | 3 | 01:11:00 |

|  |  |  |
| --- | --- | --- |
| **Tabulka Žánr** | |  |
| **ID Žánru** | **Název žánru** |  |
| 1 | Komedie |  |
| 2 | Akční |  |
| 3 | Dokumentární |  |

Způsob, jakým lze třetí normální formu uspokojit je dekompozice tabulky, díky čemuž dosáhneme odstranění nalezené tranzitivní závislosti. Pro příklad výše by dekompozice vypadala následovně: Rozkladem původní tabulky nyní dostáváme dvě tabulky, které již neobsahují žádné tranzitivní závislosti. Navíc jsme díky naplnění třetí normální formy napomohli ke snížení redundance dat.

4.NF - Každý sloupec (atribut) tabulky popisuje pouze jeden fakt nebo jednu souvislost.

5.NF - Tabulka je ve stavu, že přidáním libovolného nového sloupce (atributu) by se rozpadla na více tabulek.

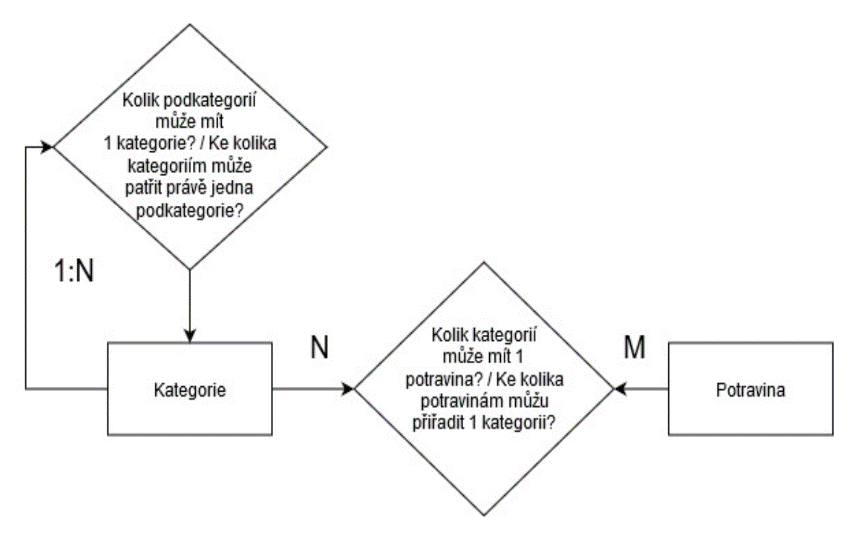
***Normalizace***= proces zjednodušování a optimalizace navržených struktur databázových tabulek. Hlavním cílem je navrhnout databázové tabulky tak, aby vykazovaly minimum [redundance](https://cs.wikipedia.org/wiki/Redundance) (opakování stejné informace na více místech). Správnost navržení struktur v tomto smyslu lze ohodnotit číslem normální formy.

**Příklady** (relačních) databází:[Microsoft Access](https://cs.wikipedia.org/wiki/Microsoft_Access), [MySQL](https://cs.wikipedia.org/wiki/MySQL), [PostgreSQL](https://cs.wikipedia.org/wiki/PostgreSQL), [Oracle](https://cs.wikipedia.org/wiki/Oracle_Database), [Microsoft SQL Server](https://cs.wikipedia.org/wiki/Microsoft_SQL_Server), [SQLite](https://cs.wikipedia.org/wiki/SQLite), [Firebird](https://cs.wikipedia.org/wiki/Firebird)

**Databázová transakce** je skupina příkazů, které převedou [databázi](https://cs.wikipedia.org/wiki/Datab%C3%A1ze) z jednoho konzistentního stavu do druhého. Databázové transakce musí splňovat tzv. vlastnosti [***ACID***](https://cs.wikipedia.org/wiki/ACID)**:**

**Atomicita** (angl. *atomicity*, *A*)Databázová transakce je jako operace dále nedělitelná (atomická). Provede se buď jako celek, nebo se neprovede vůbec (a daný databázový systém to dá uživateli na vědomí, např. chybovým hlášením).

**Konzistence** (angl. *consistency*, *C*)Transakce převádí databázi z jednoho konzistentního stavu na druhý.[[2]](https://cs.wikipedia.org/wiki/Datab%C3%A1zov%C3%A1_transakce#cite_note-2) To znamená, že její účel je zajistit určité dynamické integritní omezení (tj. omezení, jež nespadá do kategorie tzv. [databázových integritních omezení](https://cs.wikipedia.org/wiki/Datab%C3%A1zov%C3%A1_integrita), jež jsou ovšem ze své podstaty statická). Například nelze ukončit převod peněz z účtu na účet bez toho, aby jak operace odečtení ze zdrojového účtu tak operace připočtení na cílový účet byly obě provedeny.

**Izolovanost** (angl. *isolation*, *I*) Operace uvnitř transakce jsou skryty před vnějšími operacemi. Vrácením transakce (operací [ROLLBACK](https://cs.wikipedia.org/wiki/ROLLBACK)) není zasažena jiná transakce, a když ano, i tato musí být vrácena. V důsledku tohoto chování může dojít k tzv. řetězovému vrácení.

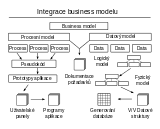
**Trvalost** (angl. *durability*, *D*) Změny, které se provedou jako výsledek úspěšných transakcí, jsou skutečně uloženy v databázi, a již nemohou být ztraceny.

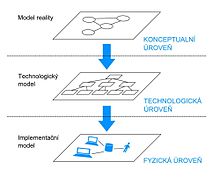
**Entitně-Relační (ER)** Model vyplývá ze vzájemných vztahů (relací) mezi entitami. Na rozdíl od síťového modelu lze rodiče rekurzivně přiřadit k totožnému rodiči.

**18. Konceptuální, logický a fyzický model databáze - jejich popis a účel**

**Datové modelování** je jednou z disciplín [softwarového inženýrství](https://cs.wikipedia.org/wiki/Softwarov%C3%A9_in%C5%BEen%C3%BDrstv%C3%AD). Je to proces, při němž se definují a analyzují požadavky na strukturu dat, s nimiž pracuje [informační systém](https://cs.wikipedia.org/wiki/Informa%C4%8Dn%C3%AD_syst%C3%A9m). Výsledkem tohoto procesu je [datový model](https://cs.wikipedia.org/w/index.php?title=Datov%C3%BD_model&action=edit&redlink=1). Datové modely popisují (definují) formát a strukturu dat v těchto systémech a určují vzájemné vztahy jednotlivých datových prvků navzájem, čímž v nich reprezentují vymezenou část reality. Datové modelování je postaveno na přístupu nazvaném Princip tří architektur.

Příklad interakce business modelu a datového modelu:

[](https://commons.wikimedia.org/wiki/File:Process_and_data_modeling_cz.svg)**Datové modelování** je proces návrhu struktury a uspořádání dat, jehož cílem je převést reálné objekty na objekty datové ([data](https://cs.wikipedia.org/wiki/Data)). Datové modelování by mělo být zahrnuto do všech projektů vyžadujících analýzu dat. Umožňuje vytvořit standardizovaný a konzistentní návrh datové struktury. Využívá se v mnoha oblastech, například pro: návrh databází a datových úložišť, integraci informačních systémů a správu dat. Datové modelování mimo jiné usnadňuje komunikaci mezi jednotlivými osobami podílejícími se na návrhu [informačního systému](https://cs.wikipedia.org/wiki/Informa%C4%8Dn%C3%AD_syst%C3%A9m). Datový model totiž obsahuje veškeré informace, které jednotliví členové realizačního týmu (programátoři, analytici i osoby, které navrhují databázové struktury) potřebují. Výsledný datový model doplňuje model podnikových procesů – popisuje strukturu dat, s kterými procesy pracují.

**Princip tří architektur** je přístup, jenž rozděluje proces tvorby [informačního](https://cs.wikipedia.org/wiki/Informa%C4%8Dn%C3%AD_syst%C3%A9m) ([databázového](https://cs.wikipedia.org/wiki/Datab%C3%A1zov%C3%BD_syst%C3%A9m)) systému do tří mentálně zvládnutelných fází, které představují různé úrovně [abstrakce](https://cs.wikipedia.org/wiki/Abstrakce). Každá z těchto tří vrstev abstrahuje od specifických charakteristik ostatních úrovní a zaměřuje se na jeden z hlavních aspektů vyvíjeného systému: obsah, technologii a implementační/realizační specifika. Tyto vrstvy navzájem tvoří přirozenou posloupnost, neboť ze specifikace obsahu vyplývají možnosti technologického řešení a konkrétní použitá technologie pak určuje možnosti [implementace](https://cs.wikipedia.org/wiki/Implementace). **Výsledkem každé fáze je jeden konkrétní**[**model**](https://cs.wikipedia.org/wiki/Model_(abstrakce))**, který je vstupem do fáze navazující**, ve které je přetransformován do konkrétnější podoby, vyplývající z úrovně obecnosti příslušné vrstvy. V případě změny se nejprve vždy určí, které z vrstev této architektury se daná změna týká, a podle toho se provedou potřebná opatření. Například aktualizace [databázového systému](https://cs.wikipedia.org/wiki/Datab%C3%A1zov%C3%BD_syst%C3%A9m) dopadá na implementační úroveň, přechod ze stromové reprezentace dat na [relační databázi](https://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze) se týká úrovně technologické a potřeba přidání funkcionality spadá do úrovně konceptuální. Princip tří architektur tedy představuje postup shora dolů, kdy se od nejobecnějšího pohledu postupně dostáváme k tomu nejdetailnějšímu.[](https://commons.wikimedia.org/wiki/File:P3A.jpg)

**Konceptuální model**

V první fázi datového modelování je nutné vymezit veškeré informace, které budeme v našem systému uchovávat. Ze všech dostupných informací je třeba určit ty, které jsou pro nás důležité a mají tedy být o dané části reality dostupné. Všechny tyto informace včetně vztahů mezi nimi je pak třeba věrně (tedy bez jakéhokoliv přizpůsobování se potřebám momentálního [implementačního](https://cs.wikipedia.org/wiki/Implementace) záměru) popsat. V této části vzniká [model](https://cs.wikipedia.org/wiki/Model_(abstrakce)) reality, popisující čistě její obsahovou stránku. K tomu, aby informace o popisovaných objektech byly relevantní, je třeba použít určitou míru [abstrakce](https://cs.wikipedia.org/wiki/Abstrakce). To znamená, že jsou zde záměrně ignorovány některé další informace, jako jsou například technologická či implementační specifika. Toto zjednodušení nám poskytne zpřehlednění příliš obsáhlé a složité reality. Jelikož z této vrstvy vycházejí zbylé dvě, je nutné, aby v konceptuálním modelu bylo popsané vše, co je nebo bude podstatné pro kohokoli z členů realizačního týmu. Nikdo z nich nemá mít potřebu domýšlet si něco, co by mu v konceptuálním modelu chybělo. **V tomto návrhu tedy určujeme pouze CO je obsahem systému, a ne jaké řešení pro zprostředkování tohoto obsahu v našem systému následně použijeme.**

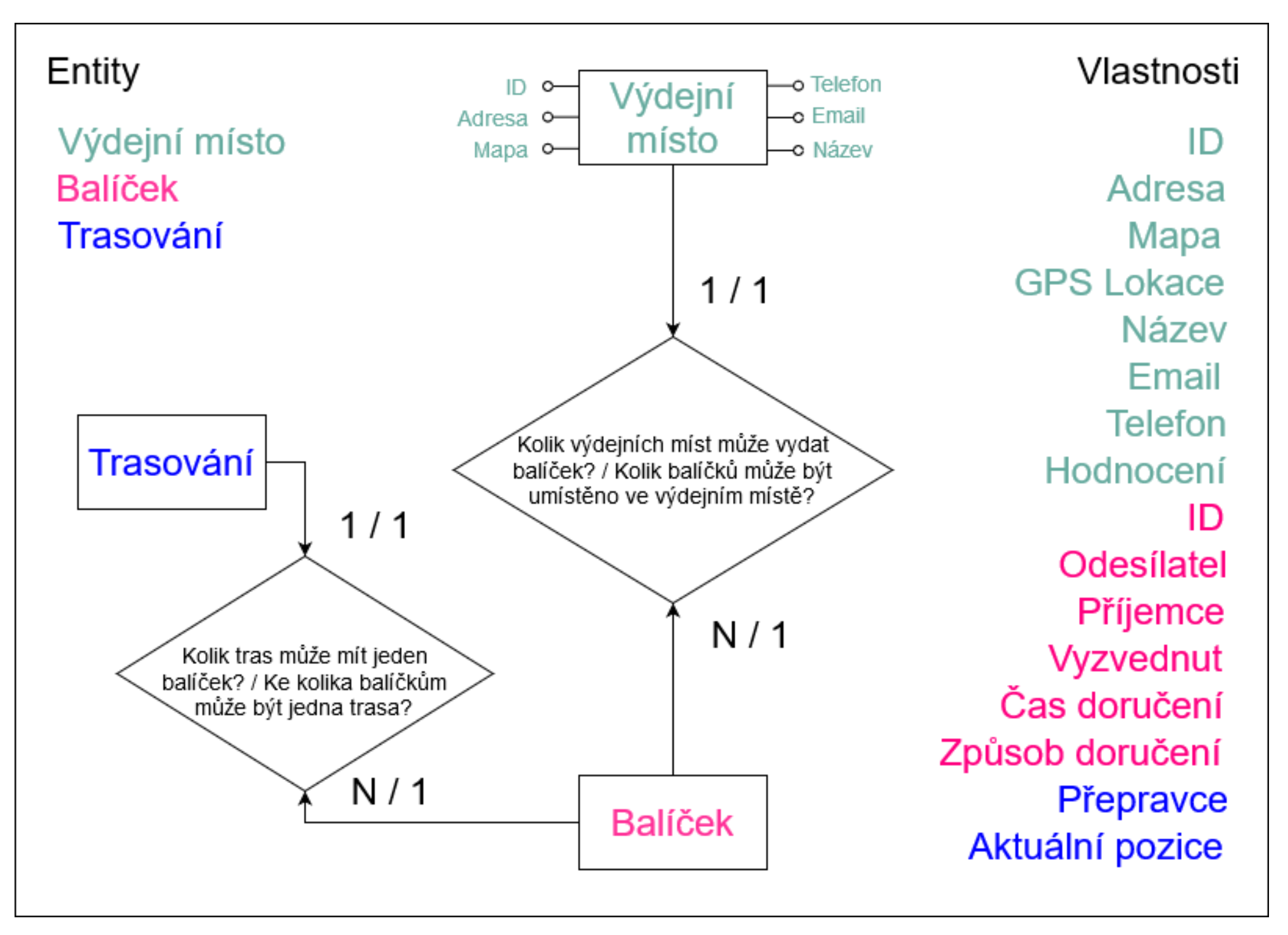
K tvorbě datového modelu na konceptuální úrovni se nejčastěji používá některá z modifikací [ER diagramů](https://cs.wikipedia.org/wiki/Entity-relationship_model).

**Neexistuje žádný specificky definovaný postup, jak tvořit konceptuální model**, proto následující postup je doporučení vycházející z praxe.

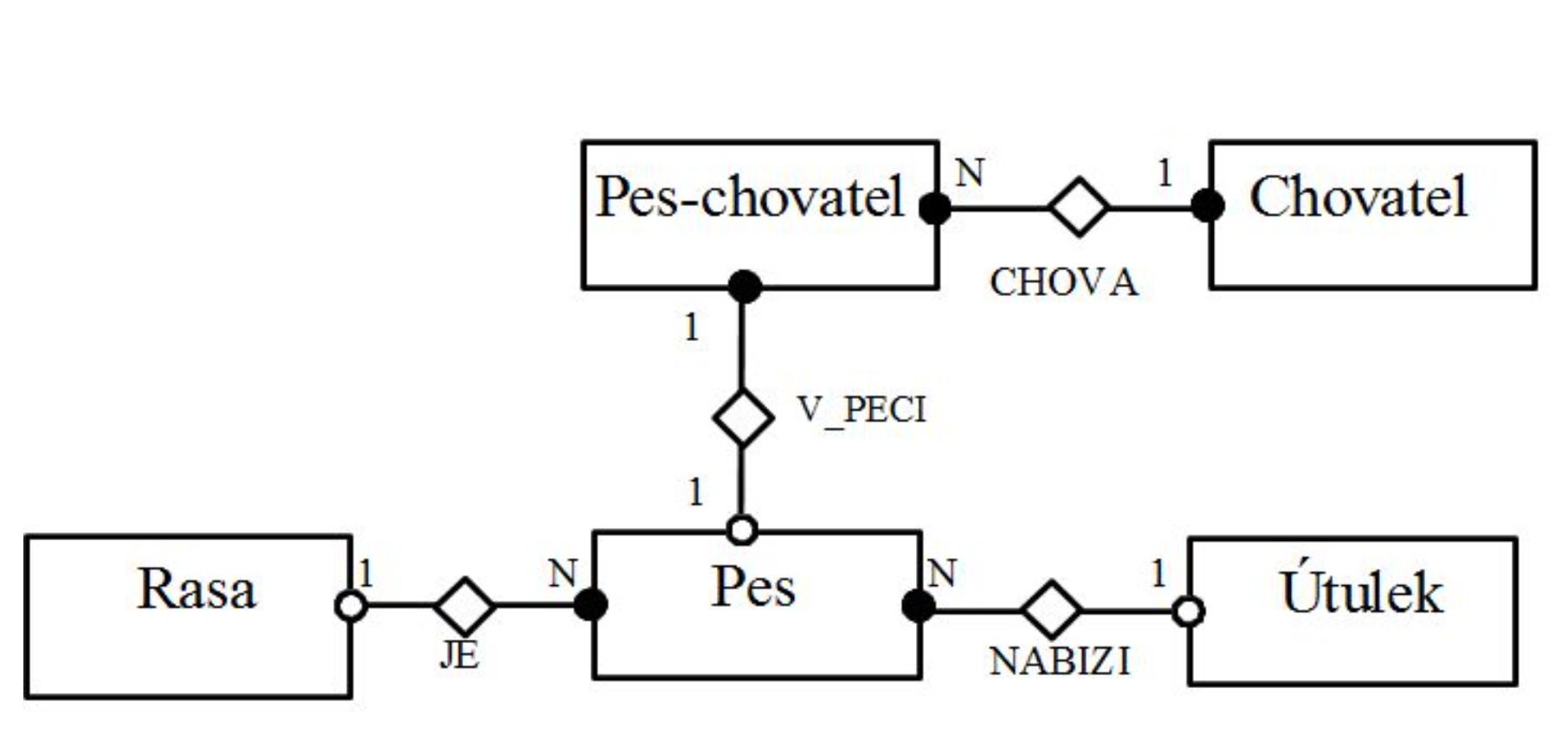
**Konceptuální model - 1. fáze** V první fázi konceptuálního modelování doporučuji vypsat si entitu po entitě, které vyplývají ze zadání. A zároveň ke každé entitě vymýšlet i její vlastnosti. Např.: Entity si pište na plátno vlevo a vlastnosti entit na plátno vpravo. Při sepisování vlastností a enti nám vyplývá, že mnohé entity budou mít společné vlastnosti, např. ID nebo Název. Proto je vhodné si rozepsat vlastnosti entitu po entitě a v závěru projít celý seznam vlastností, zda jsme u nějaké entity nezapomněli na vlastnost, která nás nemusela zprvu napadnout. Dále je ze seznamu patrné, že některé vlastnosti by bylo vhodné pojmout spíše jako entity, protože se u nich mohou záznamy duplikovat, např. Odesílatel, Příjemce, způsob doručení, aj.

**Konceptuální model - 2. fáze** V druhé fázi konceptuálního modelu navrhujeme vazbu mezi entitami a popisujeme vztahy pomocí vlastností (Kardinalita + Mandatornost) a u entit popisujeme jejich vlastnosti.

* **Entity označujeme pomocí obdélníku.**
* **Vlastnosti u entit označujeme jako tykadlo vystupující z entity**.
* **Relaci, tedy vztah popisujeme spojením čar ke kosodélníku, ve kterém vypisujeme otázku, která vyjadřuje kardinalitu vztahu.**



**ER Diagram** je konceptuální model, který se používá k návrhu na vysoké úrovni a k porozumění základním entitám a vztahům mezi nimi. Může mít několik podob v závislosti na používaném prostředí a detailnosti s jakou jej potřebujeme vypracovat. Atributy můžou být v grafu znázorněny ovály spojenými s objekty (obdélníky), vazba 1:N může být znázorněna "hráběmi" místo N, či celý diagram se může podobat třídnímu diagramu s atributy vepsanými do objektu.



**Entitně-relační model (ER model) se primárně používá jako konceptuální model**. Jeho hlavním cílem je graficky a srozumitelně zobrazit strukturu dat, včetně entit, jejich atributů a vztahů mezi nimi, na vysoké úrovni. Tento model je nezávislý na konkrétním databázovém systému a slouží k lepšímu pochopení dat a jejich vzájemných vztahů. L**ogický model** je následným krokem po konceptuálním modelu a **ER model může být také použit k vytvoření logického modelu databáze**. V této fázi se ER model překládá do konkrétnější podoby, která zahrnuje detaily potřebné pro implementaci v konkrétním databázovém systému, jako jsou tabulky, sloupce, datové typy, primární a cizí klíče.

Nástroje pro tvorbu ERdiagramů:

1. **Microsoft Visio:** Placený nástroj od Microsoftu pro tvorbu různých diagramů, včetně ER diagramů.
2. **Draw.io:** Bezplatný online nástroj pro tvorbu různých druhů diagramů, včetně ER diagramů.
3. **ERDPlus:** Další online nástroj, který umožňuje tvorbu ER diagramů zdarma.
4. **SmartDraw:** Placený nástroj pro tvorbu diagramů, včetně ER diagramů.
5. **MySQL Workbench:** je primárně určen pro vývoj a správu databází, ale umožňuje tvorbu ER diagramů.

**Logický model** Na konceptuální úroveň přirozeně **navazuje technologická (někdy též zvaná logická) úroveň, která představuje střední míru**[**abstrakce**](https://cs.wikipedia.org/wiki/Abstrakce)**.** V této fázi, obsahově vyplývající z konceptuálního schématu, je zohledňováno technologické řešení problému. Zde se určuje, jak **budou uchovávaná**[**data**](https://cs.wikipedia.org/wiki/Data)**strukturována (například zda budou uchovávána v objektové**[**databázi**](https://cs.wikipedia.org/wiki/Datab%C3%A1ze)**, relační**[**databázi**](https://cs.wikipedia.org/wiki/Datab%C3%A1ze)**apod.**) a jak bude toto pojetí v dané technologii realizováno. Jsou zde brány v úvahu dostupné technologické prostředky a možnosti dané architektury, nicméně tato úroveň je stále odlehčena od konkrétních implementačních specifik řešení. Ta jsou zohledňována až v poslední fázi. Technologický **model, který je dosud**[**platformě**](https://cs.wikipedia.org/wiki/Platforma_(informatika))**nezávislý, popisuje, JAK je obsah systému vymezený v předchozí konceptuální úrovni realizován**. Z toho vyplývá, že jeden konceptuální model může být interpretován mnoha různými technologickými modely, neboť jeden obsah může být vyjádřen mnoha způsoby.

**Lineární zápis entit a vztahů** je logický model, který popisuje, jak budou entity a vztahy implementovány v databázovém systému. Lineární zápisem popisujeme objekty, jejich vlastnosti a vztahy z pohledu implementačního. Lineárním zápisem entit jsou v podstatě definovány tabulky a jejich atributy včetně primárních a *cizích klíčů*.

Příklad lineárního zápisu entity: Pes (IDPes, jmeno, pohlavi, vek, *CRasa*, *IDUtulek*)  
Příklad lineárního zápisu vztahů: NABIZI (Útulek, Pes) 1:N

**Fyzický model (implementační)** Implementační úroveň datového modelování představuje nejnižší míru [abstrakce](https://cs.wikipedia.org/wiki/Abstrakce). **Není zde již prostor pro žádné zjednodušení. V této fázi probíhá realizace (**[**implementace**](https://cs.wikipedia.org/wiki/Implementace)**) datové struktury**, popsané v předchozí logické úrovni, **v konkrétním zvoleném**[**programovacím jazyce**](https://cs.wikipedia.org/wiki/Programovac%C3%AD_jazyk). Doposud nezávislý technologický model je na této úrovni transformován do [modelu](https://cs.wikipedia.org/wiki/Model_(abstrakce)), který vyplývá z konkrétních specifik použité databázové [platformy](https://cs.wikipedia.org/wiki/Platforma_(informatika)). Implementační fáze tedy musí zohledňovat všechny její dostupné prostředky a možnosti. Z toho vyplývá, že implementační úroveň popisuje, **ČÍM** je datový obsah systému, popsaný konceptuálním modelem a strukturovaný modelem technologickým, realizován.

Ze stejného důvodu, jako tomu bylo u konceptuálního modelu, může být model technologický rovnocenně vyjádřen několika implementačními modely. Konkrétní technologické řešení popsané v druhé fázi datového modelování lze totiž realizovat několika odlišnými způsoby. Vzniká zde fyzický model, který si lze představit jako model technologický (logický), rozšířený o některé specifické informace **pro konkrétní** [**databázovou**](https://cs.wikipedia.org/wiki/Datab%C3%A1ze)[**platformu**](https://cs.wikipedia.org/wiki/Platforma_(informatika))**(například datové typy).**

Datové modelování tedy slouží k vymezení standardů definujících formát a uspořádání dat, používaných v konkrétním [informačním systému](https://cs.wikipedia.org/wiki/Informa%C4%8Dn%C3%AD_syst%C3%A9m). Díky tomu jsou tato [data](https://cs.wikipedia.org/wiki/Data) napříč systémem konzistentní a je možné je plnohodnotně využívat a sdílet různými částmi systému. Mají-li být [data](https://cs.wikipedia.org/wiki/Data) sdílena i mezi různými [informačními systémy](https://cs.wikipedia.org/wiki/Informa%C4%8Dn%C3%AD_syst%C3%A9m) navzájem, je potřeba, aby mezi nimi bylo společné [rozhraní](https://cs.wikipedia.org/wiki/Rozhran%C3%AD_(informatika)) sloužící k jejich vzájemné komunikaci, neboť různé [informační systémy](https://cs.wikipedia.org/wiki/Informa%C4%8Dn%C3%AD_syst%C3%A9m) mají často odlišné datové modely.

**Každý sloupec v databázové tabulce musí mít svůj název a datový typ.** Máme tři druhy základních datových typů a to čísla, řetězce a časové (datum a čas).

Datové typy řetězcové: Většinu dat, kterou budete ukládat, bude ve formátu řetězce. Tento seznam popisuje běžné datové typy řetězců. Pole datového typu řetězce může obsahovat libovolné abecední a číselné znaky pro definování malého a velkého textu, jako je jméno, adresa, popis atd. Nebo i binární obsah, jako je obrázek nebo zvukový soubor.

| **Datový typ** | **Popis** |
| --- | --- |
| **CHAR(M)** | Řetězec s pevnou délkou mezi 1 a 255 znaky (například CHAR(5)), doplněný zprava mezerami na zadanou délku při uložení. Definování délky není povinné, ale výchozí hodnota je 1. |
| **VARCHAR(M)** | Řetězec s proměnnou délkou o délce 1 až 255 znaků. Například VARCHAR(25). Při vytváření pole VARCHAR musíte definovat délku.  **NVARCHAR (M)** je také datový typ určený pro ukládání proměnných délek znakových dat. Oproti VARCHAR používá NVARCHAR dvoubajtové (16-bitové) kódování znaků, což znamená, že každý znak je reprezentován dvěma bajty. NVARCHAR je často používán pro podporu více jazyků. |
| **BLOB**nebo TEXT | Pole s maximální délkou 65535 znaků. Objekty BLOB (Binary Large Object) jsou „binární velké objekty“ a používají se k ukládání velkého množství binárních dat, jako jsou obrázky nebo jiné typy souborů. Pole definovaná jako TEXT také obsahují velké množství dat. Rozdíl mezi nimi je v tom, že řazení a porovnání uložených dat rozlišují malá a velká písmena u objektů BLOB a nerozlišují malá a velká písmena v polích TEXT. Pomocí BLOB nebo TEXT nezadáte délku. |
| TINYBLOB nebo**TINYTEXT** | Sloupec BLOB nebo TEXT s maximální délkou 255 znaků. Pomocí TINYBLOB nebo TINYTEXT nezadáváte délku. |
| MEDIUMBLOB nebo MEDIUMTEXT | Sloupec BLOB nebo TEXT s maximální délkou 16777215 znaků. Pomocí MEDIUMBLOB nebo MEDIUMTEXT nezadáváte délku. |
| LONGBLOB nebo LONGTEXT | Sloupec BLOB nebo TEXT s maximální délkou 4294967295 znaků. Pomocí LONGBLOB nebo LONGTEXT nezadáváte délku. |
| ENUM | Výčet, což je módní výraz pro seznam. Při definování ENUM vytváříte seznam položek, ze kterých je třeba vybrat hodnotu (nebo může být NULL). Pokud byste například chtěli, aby vaše pole obsahovalo „A“ nebo „B“ nebo „C“, definovali byste svůj ENUM jako ENUM (‚A‘, ‚B‘, ‚C‘) a pouze tyto hodnoty (nebo NULL) mohl někdy zalidnit to pole. |

Datové typy číselné: Pole typu číselných dat ukládá celá čísla, jako je ID, bezpečnostní čísla, zlomkové čísla jako je cena, CGPA a další:

| **Datový typ** | **Popis** |
| --- | --- |
| **INT** | Celé číslo normální velikosti, které může být se znaménkem nebo bez znaménka. Pokud je podepsáno, je povolený rozsah od -2147483648 do 2147483647. Pokud není podepsán, je povolený rozsah od 0 do 4294967295. Můžete zadat šířku až 11 číslic. |
| **TINYINT** | Velmi malé celé číslo, které může být se znaménkem nebo bez znaménka. Pokud je podepsáno, je povolený rozsah od -128 do 127. Pokud není podepsán, je povolený rozsah od 0 do 255. Můžete zadat šířku až 4 číslic. |
| **SMALLINT** | Malé celé číslo, které může být se znaménkem nebo bez znaménka. Pokud je podepsáno, je povolený rozsah od -32768 do 32767. Pokud není podepsán, je povolený rozsah od 0 do 65535. Můžete zadat šířku až 5 číslic. |
| MEDIUMINT | Středně velké celé číslo, které může být se znaménkem nebo bez znaménka. Pokud je podepsáno, je povolený rozsah od -8388608 do 8388607. Pokud není podepsán, je povolený rozsah od 0 do 16777215. Můžete zadat šířku až 9 číslic. |
| **BIGINT** | Velké celé číslo, které může být se znaménkem nebo bez znaménka. Pokud je podepsáno, je povolený rozsah od -9223372036854775808 do 9223372036854775807. Pokud není podepsán, je povolený rozsah od 0 do 18446744073709551615. Můžete zadat šířku až 2 číslic. |
| **FLOAT(M,D)** | Číslo s plovoucí desetinnou čárkou, které nemůže být bez znaménka. Můžete definovat délku zobrazení (M) a počet desetinných míst (D). Toto není vyžadováno a výchozí hodnota bude 10,2, kde 2 je počet desetinných míst a 10 je celkový počet číslic (včetně desetinných míst). Desetinná přesnost může dosáhnout 24 míst pro FLOAT.  float(10,4) 012345,6789  float(20,10) 0123456789,0123456789 |
| **DOUBLE(M,D)** | Číslo s plovoucí desetinnou čárkou s dvojitou přesností, které nelze oddělit. Můžete definovat délku zobrazení (M) a počet desetinných míst (D). Toto není vyžadováno a výchozí hodnota bude 16,4, kde 4 je počet desetinných míst. Desetinná přesnost může dosahovat až 53 míst za DVOJNÁSOBEK. REAL je synonymem pro DOUBLE. |
| DECIMAL(M,D) | Rozbalené číslo s pohyblivou řádovou čárkou, které nelze zrušit. V rozbalených desítkách odpovídá každé desetinné místo jednomu bajtu. Je nutné definovat délku zobrazení (M) a počet desetinných míst (D). NUMERIC je synonymem pro DECIMAL. |

Datové typy časové (čas a datum): Tento datový typ se používá k ukládání hodnot souvisejících s datem a časem. Pole typu datum a čas může ukládat hodnoty data, času, data a času, roku a časového razítka. Pokud jsou pro hodnotu pole datum a čas zadána neplatná data, bude vložena hodnota NULL. Různé datové typy pro datum a čas jsou popsány níže.

| **Datový typ** | **Popis** |
| --- | --- |
| **DATE** | Datum ve formátu RRRR-MM-DD mezi 1000-01-01 a 9999-12-31. Například 30. prosinec 1973 bude uložen jako 1973-12-30. |
| **DATETIME** | Kombinace data a času ve formátu YYYY-MM-DD HH:MM:SS, mezi 1000-01-01 00:00:00 a 9999-12-31 23:59:59. Například 3:30 odpoledne 30. prosince 1973 bude uloženo jako 1973-12-30 15:30:00. |
| **TIMESTAMP** | časové razítko mezi půlnocí, 1. lednem 1970 a někdy v roce 2037. Vypadá to jako předchozí formát DATETIME, pouze bez pomlček mezi čísly; 3:30 odpoledne 30. prosince 1973 bude uloženo jako 19731230153000 (YYYYMMDDHHMMSS). |
| **TIME** | Ukládá čas ve formátu HH:MM:SS. |
| **YEAR(M)** | Ukládá rok ve 2místném nebo 4místném formátu. Pokud je délka zadána jako 2 (například YEAR(2)), YEAR může být mezi 1970 až 2069 (70 až 69). Pokud je délka zadána jako 4, pak YEAR může být 1901 až 2155. Výchozí délka je 4. |

Další datové typy jsou například:

1. **Booleovské hodnoty (Boolean)**: Ukládá pravdivostní hodnoty, například True nebo False.
2. **Referenční typy (Reference)**: Používají se k vytvoření vztahu mezi tabulkami. Například FOREIGN KEY sloupec v jedné tabulce odkazuje na primární klíč v jiné tabulce.

**Rozdíly mezi konceptuálním a logickým ER diagramem:**

* Konceptuální ER diagram je abstraktní a zaměřuje se na koncepty a vztahy, zatímco logický ER diagram se více zaměřuje na strukturu tabulek a implementační detaily.
* Konceptuální ER diagram je často používán pro komunikaci s klienty a uživateli, zatímco logický ER diagram je nástrojem pro vývojáře a administrátory databáze.
* Konceptuální ER diagram zahrnuje základní entity a vztahy, zatímco logický ER diagram zahrnuje tabulky, klíče, indexy a další technické detaily.

Atributy mají také své vlastnosti, které definují jejich charakteristiku, jako je datový typ, délka, jestli je povinný, unikátní atd. Pro vytvoření fyzického modelu musíte znát i vlastnosti atributů.V MySQL Workbench, už když navrhujete [logický model](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6324), můžete každému atributu (sloupci v tabulce) přiřadit různé vlastnosti.  Tyto vlastnosti můžete v MySQL Workbench nastavit v panelu vlastností atributu:

1. **Name (Název)**: Jednoznačný identifikátor atributu v rámci entity.
2. **Datatype (Datový typ)**: Určuje typ dat, která atribut může obsahovat. Běžné [datové typy](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6325) zahrnují VARCHAR (řetězec proměnné délky), INT (celé číslo), DATE (datum), BOOLEAN (logická hodnota) atd.
3. **PK (Primary Key, Primární klíč)**: Označuje, že atribut je primárním klíčem entity. Primární klíč jednoznačně identifikuje každý záznam v tabulce.
4. **NN (Not Null, Nesmí být NULL)**: Určuje, že hodnota atributu nesmí být NULL, tedy každý záznam musí mít pro tento atribut definovanou hodnotu.
5. **UQ (Unique, Unikátní)**: Označuje, že každý záznam v tabulce musí mít pro tento atribut unikátní hodnotu.
6. **BIN (Binary)**: Určuje, že porovnávání bude prováděno binárně. Tato vlastnost je relevantní pro řetězcové [datové typy](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6325).
7. **UN (Unsigned)**: Používá se u číselných datových typů k indikaci, že čísla budou kladná (bez znaménka).
8. **ZF (Zero Fill)**: Vyplní neobsazené pozice číselných hodnot nulami, aby dosáhly definované šířky.
9. **AI (Auto Increment)**: Hodnota atributu se automaticky zvyšuje o 1 s každým nově vloženým záznamem. Často se používá u primárních klíčů.
10. **Default (Výchozí hodnota)**: Přednastavená hodnota, která je automaticky přiřazena atributu, pokud při vkládání záznamu není specifikována jiná hodnota.
11. **Comments (Komentáře)**: Popis nebo poznámky k atributu, které mohou poskytnout další informace nebo vysvětlení.

**19. SQL - charakteristika, syntaxe, základní dotazy CRUD, příklady**

**SQL** (*es-kjů-el* ) je zkratka ([anglicky](https://cs.wikipedia.org/wiki/Angli%C4%8Dtina) *Structured Query Language*) pro standardizovaný strukturovaný [dotazovací jazyk](https://cs.wikipedia.org/wiki/Dotazovac%C3%AD_jazyk), který je používán pro práci s daty v [relačních databázích](https://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze)

**CRUD**: je zkratka pro 4 základní operace, které lze vždy s každým informačním systémem provádět:

Create - Vytváření (SQL: INSERT) **INSERT INTO** osoba (name) **VALUES** ('Novák'), ('Novotný'), ('Krásná');

Read - Čtení (SQL: [SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329)) [**SELECT**](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329)\* FROM osoba; [SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) idosoba, name FROM osoba;

Update - Aktualizace dat (SQL: UPDATE) **UPDATE** osoba **SET** name = 'Novotný' WHERE idosoba = 2;

Delete - Smazání dat (SQL: DELETE) **DELETE FROM** osoba WHERE idosoba = 2;

Zobrazení všech databází na serveru: **show schemas;**

Použití konkrétní databáze pro práci: **use** `osoba`**;**

Zobrazení všech tabulek/entit uvnitř databáze: **show tables;**

Zobrazení vlastností + atributů z konkrétní tabulky: **show \* from** osoba;

Jednořádkový komentář: -- Komentář

-- WHERE - podmínka pomocí které filtrujeme výsledek

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) <sloupce> FROM <nazev tabulky> **WHERE** <hodnota sloupce> = <hodnota>;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) jmeno, prijmeni FROM uzivatel WHERE prijmeni = 'Novák';

-- LIKE - Přidáváme ke klauzuli WHERE pokud používáme např. výraz pro vyhledávání s řetězci.

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) jmeno, prijmeni FROM uzivatel WHERE prijmeni **LIKE** 'Novak%';

-- % je zástupný znak pro libovolný počet libovolných znaků pro řetězce v SQL

-- ORDER BY - Slouží k řazení výstupu dle zvoleného sloupce tabulky

-- Ascending [ASC] - Vzestupně = 1,2,3,4, ... -- Descending [DESC] - Sestupně = 10,9,8,7, ...

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) jmeno, prijmeni FROM uzivatel WHERE prijmeni LIKE 'Novak%' **ORDER BY** prijmeni **ASC**;

-- /\* 1. Vyhledejte všechny názvy žánrů, které mají ID 5 a méně \*/

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) name FROM genre WHERE idgenre <= 5;

-- /\* 2. Vypište všechna ID žánrů, které jsou 2 cif. a seřaďte od nejmenšího \*/

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) idgenre FROM genre  WHERE (idgenre >= 10 AND idgenre <= 99) ORDER BY idgenre ASC;

-- /\* 3. Vypište všechny názvy žánrů, které začínají na písmeno 'M' \*/

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) name FROM genre WHERE name LIKE 'M**%**';

-- SELECT: -- PODMÍNKY - WHERE, LIKE, SIMILAR TO, ORDER BY, BETWEEN;

 -- AGREGAČNÍ KLAUZULE (Funkce):

-- Count(), MIN(), MAX(), AVG(), SUM(); - Číselné

        -- DISTINCT, GROUP BY, CONCAT(), UNION; - Řetězcové;

-- [JOINy](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6328" \o "JOINy) - JOIN, LEFT JOIN, RIGHT JOIN, INNER JOIN, ON;

-- /\* Spočítej kolik žánrů her je v DB \*/

SELECT **Count**(idgenre) FROM genre; -- /\* = 10 \*/

SELECT **SUM(**idgenre) FROM genre;

-- ----------------------------------------------------------------------------------------------

**ÚVOD DO VÝUKY SQL PRO ZAČÁTEČNÍKY**

**DROP** **DATABASE** IF EXISTS Skola;

**CREATE DATABASE** IF NOT EXISTS Skola

    USE Skola;

   CREATE TABLE IF NOT EXISTS `Skola`.`Studenti` (

  `Id` INT NOT NULL **PRIMARY KEY**,

  `Jmeno` VARCHAR(20) ,

  `Prijmeni` VARCHAR(20) ,

  `DatumNarozeni` DATE ,

  `MaRozsirenouVyuku` VARCHAR(80),

   `Rocnik` INT,

   `Skolne` INT);

INSERT INTO `Skola`.`Studenti`( `Id`,`Jmeno` ,`Prijmeni`  ,`DatumNarozeni` ,`MaRozsirenouVyuku`,`Rocnik`, `Skolne`) VALUES (1,'Petra','Tyrkysová', '2003-10-12','nema', 2, 14100);

 INSERT INTO `Skola`.`Studenti`( `Id`,`Jmeno` ,`Prijmeni`  ,`DatumNarozeni` ,`MaRozsirenouVyuku`,`Rocnik`, `Skolne`)

 VALUES (2,'Petr','Tyrkyso', '2006-11-12','nema', 2, 16100), (3,'Jan','Novák', '2001-10-12','ma', 2, 15200), (4,'Petra','Tyrkysová', '2098-05-06','nema', 4, 24100);

   -- Nyní vložíme do tabulky nové věty pomocí sdružených příkazů:

**START TRANSACTION;**

USE `Skola`;

 INSERT INTO `Skola`.`Studenti`( `Id`,`Jmeno` ,`Prijmeni`  ,`DatumNarozeni` ,`MaRozsirenouVyuku`,`Rocnik`, `Skolne`)

 VALUES (5,'Petr','Tyrkyso', '2006-11-12','nema', 2, 16100), (6,'Jan','Novák', '2001-10-12','ma', 2, 15200), (7,'Petra','Tyrkysová', '2098-05-06','nema', 4, 24100);

INSERT INTO `Skola`.`Studenti`( `Id`,`Jmeno` ,`Prijmeni`  ,`DatumNarozeni` ,`MaRozsirenouVyuku`,`Rocnik`, `Skolne`)

 VALUES (8,'Petr','Tyrkyso', '2006-11-12','nema', 2, 16100), (9,'Jan','Novák', '2001-10-12','ma', 2, 15200), (10,'Petra','Tyrkysová', '2098-05-06','nema', 4, 24100);

**COMMIT;**

INSERT INTO `Skola`.`Studenti`( `Id`,`Jmeno` ,`Prijmeni`  ,`DatumNarozeni` ,`MaRozsirenouVyuku`,`Rocnik`, `Skolne`) [SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) NULL ,`Jmeno` ,`Prijmeni`  ,`DatumNarozeni` ,`MaRozsirenouVyuku`,`Rocnik`, `Skolne`FROM `Skola`.`Studenti` WHERE `Jmeno` = 'Jan' AND  `Prijmeni` = 'Novák';

-- Vytvoření kopie tabulky

 DROP TABLE IF EXISTS `Skola`.`KopirStudenti`;

**CREATE TABLE KopieStudenti AS**[**SELECT**](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329)**\* FROM `Skola`.`Studenti`;**

DROP TABLE IF EXISTS `Skola`.`DruhaKopieStudenti`;

-- Vytvořte kopii tabulky „Studenti“, tentokrát ale do kopie přeneste pouze sloupce, nikoli data:

CREATE TABLE DruhaKopieStudenti AS [SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) \* FROM `Skola`.`Studenti` **WHERE 1 = 2**;

-- -----------------------------------------------------------------------------------------------**VÝUKA SQL PRO ZAČÁTEČNÍKY**

-- příkaz [SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329):

[**SELECT**](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329)**\* FROM skola.studenti;**

-- výběr z dat:

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni FROM skola.studenti;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) 'Student', Prijmeni, Jmeno FROM skola.studenti;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) 'Student' AS DruhOsoby, Prijmeni, Jmeno FROM skola.studenti;

-- složené výběry:

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik FROM skola.studenti WHERE Rocnik <> 1;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik FROM skola.studenti WHERE Rocnik != 1;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik FROM skola.studenti WHERE ((Rocnik != 1) AND (Prijmeni LIKE 'Tyrkys%'));

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik FROM skola.studenti WHERE Prijmeni LIKE 'Tyrkys\_';

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik FROM skola.studenti WHERE NOT Rocnik = 1;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik FROM skola.studenti WHERE ((Rocnik != 1) AND (Prijmeni NOT LIKE 'Tyrkys%'));

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik FROM skola.studenti WHERE Prijmeni NOT LIKE 'Tyrkys\_';

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik, MaRozsirenouVyuku FROM skola.studenti WHERE ((Rocnik != 1) AND (MaRozsirenouVyuku LIKE 'ma%'));

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik, Skolne FROM skola.studenti WHERE  Rocnik = 2 AND Skolne < 16000 ;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) \* FROM skola.studenti;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik, Skolne FROM skola.studenti WHERE  Rocnik = 2 OR Rocnik = 4 ;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik, Skolne FROM skola.studenti WHERE  Rocnik = 2 OR 4 ;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik FROM skola.studenti WHERE  Rocnik IN (2,4) ;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Skolne FROM skola.studenti WHERE  Skolne <= 16000  AND  Skolne >= 10000;

-- rozmezí:

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Skolne FROM skola.studenti WHERE  Skolne **BETWEEN 16000 AND 40000**;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, MaRozsirenouVyuku, Skolne, Rocnik from skola.studenti

where (MaRozsirenouVyuku = "ma%" or Skolne > 13000) and Rocnik = 4;

-- řazení:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Prijmeni, Rocnik, DatumNarozeni from skola.studenti **order by** Prijmeni, Rocnik;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Prijmeni, DatumNarozeni from skola.studenti order by DatumNarozeni;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Prijmeni, DatumNarozeni from skola.studenti order by DatumNarozeni **desc**;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Skolne, Prijmeni, Jmeno, Rocnik from skola.studenti where Rocnik <= 2 order by Skolne, Prijmeni, Jmeno;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni as PrijmeniStudenta, DatumNarozeni from skola.studenti order by Prijmeni, DatumNarozeni;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Skolne/12 as MesicniSkolne, Jmeno, Prijmeni from skola.studenti order by MesicniSkolne;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) 'Student' AS DruhOsoby, Prijmeni, Jmeno FROM skola.studenti;

-- přímý výpočet:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) 1+2+3;

-- CASE:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, **case Rocnik when 1 then "Prvni" when 2 then "Druhy" when 3 then "Treti" when 4 then "Ctvrty" else "Neznamy" end** as Rocnik from skola.studenti;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Skolne, case when (Skolne <=16000)  then "Nizke"  when ( Skolne <=17000 ) then "Stredni" else "Vysoke" end as KategorieSkolneho from skola.studenti;

-- if:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Skolne, **if (Skolne < 17000, "Malo", "Hodne")** as Mnozstvi from skola.studenti;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Skolne, if ((Skolne < 17000) and (Jmeno = "Petra"), "Malo", "Hodne") as MnozstviPokus from skola.studenti;

-- hodnota null:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) \* from skola.studenti where DatumNarozeni  is **null;**

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) \* from skola.studenti where DatumNarozeni is **not null**;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Prijmeni, Jmeno, isnull(1) as JednaNeniNull from skola.studenti;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Prijmeni, Jmeno, isnull(1/0) as DeleniNulouJeNull from skola.studenti;

-- převod na velká písmena:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **upper(Prijmeni)** as Prijmeni, Jmeno from skola.studenti;

-- výběr části textu:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **substr(Prijmeni,1,2)** as PrvniDvePismena, Prijmeni as CeleJmeno from skola.studenti;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) DatumNarozeni, substr((DatumNarozeni), 1, 4),  substr((DatumNarozeni), 6, 2),  substr((DatumNarozeni), 9, 2) from skola.studenti;

-- počet znaků:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Prijmeni, if**(length(Prijmeni)**<=5, "Kratke prijmeni", "Dlouhe prijmeni") as "Kratke nebo dlouhe prijmeni" from skola.studenti;

-- přepsání znaků:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) DatumNarozeni, **replace((DatumNarozeni), "-", ".")**,  replace(Jmeno,"P", "p" ) from skola.studenti;

-- spojení dvou sloupců v jeden:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, **concat(Jmeno, ' ', Prijmeni)** as CeleJmeno from skola.studenti;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) concat('       ', Jmeno, ' ', Prijmeni,'                            ') as CeleJmeno from skola.studenti;

-- ořezávání mezer:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) trim(Jmeno), ltrim(Jmeno), rtrim(Jmeno) from skola.studenti;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) concat('       ', Jmeno, ' ', Prijmeni,'                            '),

**trim**(concat('       ', Jmeno, ' ', Prijmeni,'                            ')),

**ltrim**(concat('       ', Jmeno, ' ', Prijmeni,'                            ')),

**rtrim**(concat('       ', Jmeno, ' ', Prijmeni,'                            ')) from skola.studenti;

-- práce s datem:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **dayofyear(20230201), dayname(20231224), dayofmonth(20231224**);

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) CURDATE(),  dayofyear(CURDATE()), dayname(CURDATE()), dayofmonth(CURDATE());

-- aktuální datum a čas:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **now(), curtime(), curdate();**

-- rozdíl datumů:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **datediff('2023-12-31','2023-12-29');**

-- složky datumu:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Id, Jmeno, Prijmeni from skola.studenti where month(DatumNarozeni) = **month(now()**)  and day(DatumNarozeni) = **day(now())**;

-- práce s čísly - dělení:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Jmeno, Prijmeni, Rocnik/4\*100 as "Hotovo v %" from skola.studenti;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) 4/4, 4%4, 5/4, 5%4, **round(4/4, 2)**, ceiling(5/4), floor(5/4);

-- práce s čísly - zaokrouhlování:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **format(Skolne, 2)**, format(Rocnik, 3) from skola.studenti;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) replace(format(Skolne, 0), ',', ' ') as FormatovaneSkolne, format(Rocnik,2) as FormatovanyRocnik from skola.studenti;

-- souhrny (počty všech):

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **count(\*)** as PocetVsech from skola.studenti;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) count(\*) as PocetDruhaku from skola.studenti where Rocnik = 2;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) count(\*) as PocetCtrtaku from skola.studenti where Rocnik = 4;

-- agregační funkce:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **avg(Skolne)** as PromerSkolne, **max(Skolne), sum(Skolne), min(Skolne),** sum(Skole)/count(Skolne)  as KontorlaPrumeru from skola.studenti;

-- řazení dat (=order) a počet jedinečných hodnot (=distinct):

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **distinct Skolne** as JedinecneHodnotySkolneho from skola.studenti order by Skolne;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) count( distinct Skolne) as PocetJedinecnychHodnotSkolneho from skola.studenti;

-- seskupování dat:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Rocnik, count(Rocnik) as PocetDruhaku from skola.studenti where Rocnik = 2;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Rocnik, count(Rocnik) as AutomatickeSeskupovaniRocniku from skola.studenti group by Rocnik  order by Rocnik;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Rocnik, sum(Skolne) as AutomatickeScitaniSkolneho from skola.studenti group by Rocnik order by Rocnik;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Rocnik, sum(Skolne) as AutomatickeScitaniSkolneho from skola.studenti where  Skolne > 20000 group  by Rocnik order by Rocnik;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Rocnik, sum(Skolne) as AutomatickeScitaniSkolneho from skola.studenti where  Skolne <= 20000 group  by Rocnik order by Rocnik;

-- vkládání nových záznamů do tabulky:

insert into skola.studenti (Id, Jmeno, Prijmeni, DatumNarozeni, MaRozsirenouVyuku, Rocnik, Skolne) values

(20, 'Jan', 'Novák', '2001-10-12', 'ma', '2', '15200');

insert into skola.studenti (Id,Jmeno, Prijmeni, DatumNarozeni, MaRozsirenouVyuku, Rocnik, Skolne)  values

(23, "Lenka", "Nováková",'2001-12-24',  "ma", 1, 18500),

(24, "Lenka", "Jedináková",'2001-12-24',  "ma", 1, 18500),

(25, "Petr", "Veselý",'1994-12-24',  "ma", 3, 28500);

-- vložení výsledku SELECTu:

insert into skola.studenti (Id,Jmeno, Prijmeni, DatumNarozeni, MaRozsirenouVyuku, Rocnik, Skolne) **[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329" \o "SELECT)**Id,"Jan", "Hep", DatumNarozeni, MaRozsirenouVyuku, Rocnik, Skolne from skola.studenti where Rocnik = 1;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) \* from skola.studenti where Id is not null order by Id limit 50;

-- kopie celé tabulky:

**insert into skola.kopiestudenti [select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329" \o "SELECT) \* from skola.studenti;**

-- další kopie celé tabulky:

CREATE TABLE IF NOT EXISTS skola.superkopiestudenti AS [SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) \* FROM skola.studenti;

-- změny databází:

**SET SQL\_SAFE\_UPDATES = 0;**

UPDATE skola.superkopiestudenti set  Prijmeni = "Veselý" where Id = 20;

update skola.superkopiestudenti set skola.superkopiestudenti.Skolne = 10000;

-- mazání řádků databází:

delete from skola.superkopiestudenti where Rocnik = 4;

-- další změny databází:

update skola.superkopiestudenti set Rocnik = Rocnik + 1;

update skola.superkopiestudenti set Id = Id + 1;

update skola.superkopiestudenti set Id = 2 where (Rocnik = 2 and Skolne = 10000);

update skola.superkopiestudenti set Id = 4 where Rocnik = 4 and Prijmeni = "Hepová";

-- vytvoření nové tabulky:

CREATE TABLE IF NOT EXISTS skola.superkopiestudenti2 AS [SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) \* FROM skola.studenti;

-- vkládání dat do nové tabulky:

insert into skola.superkopiestudenti2 (Id,Jmeno, Prijmeni, DatumNarozeni, MaRozsirenouVyuku, Rocnik, Skolne)  values (122, "Lenka", "Nováková",'2001-12-24',  "ma", 1, 18500);

-- smazání celé tabulky, **zbude jen hlavička tabulky**:

SET SQL\_SAFE\_UPDATES = 0;

**TRUNCATE table skola.superkopiestudenti2;**

-- vytvoření nové tabulky:

CREATE TABLE IF NOT EXISTS skola.superkopiestudenti3 AS [SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) \* FROM skola.studenti;

-- vkládání dat do tabulky:

insert into skola.superkopiestudenti3 (Id,Jmeno, Prijmeni, DatumNarozeni, MaRozsirenouVyuku, Rocnik, Skolne) values (22, "Lenka", "Nováková",'2001-12-24',  "ma", 1, 18500);

-- další smazání celé tabulky, **zbude jen hlavička tabulky:**

SET SQL\_SAFE\_UPDATES = 0;

**DELETE from skola.superkopiestudenti3;**

-- vytvoření tabulky s definicí sloupců:

CREATE TABLE skola.student2029 (Id INT (11) NOT NULL AUTO\_INCREMENT,

Jmeno varchar(20),

Prijmeni varchar(20),

DatumNarozeni date,

MaRozsirenouVyuku varchar(80),

Rocnik int(11),

Skolne int (11),

**PRIMARY KEY (Id));**

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) \* from skola.student2029;

-- jiné vytvoření tabulky s definicí sloupců:

CREATE TABLE skola.student2030 (Id INT (11) NOT NULL **primary key** AUTO\_INCREMENT,

Jmeno varchar(20),

Prijmeni varchar(20),

DatumNarozeni date,

MaRozsirenouVyuku varchar(80),

Rocnik int(11),

Skolne int (11));

-- přidání sloupců do tabulky:

**ALTER table skola.student2030 add JeExternistou bit, add Poznamka text;**

insert into skola.student2030 (Jmeno, Prijmeni, DatumNarozeni, MaRozsirenouVyuku, Rocnik, Skolne)

 values ("Lenka", "Nováková",'2001-12-24',  "ma", 1, 18500);

-- změna sloupců tabulky:

update skola.student2030 set JeExternistou = 1 where Id =1;

-- **vymazání sloupců  tabulky:**

**alter table skola.student2030 drop JeExternistou;**

-- **vymazání celé tabulky včetně hlavičky tabulky:**

**DROP table skola.student2030;**

-- nyní již neexistuje student2030   [select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329" \o "SELECT) \* from skola.student2030;

**-- Integritní omezení Constraint:**

**-- omezení hodnot CHECK():**

**ALTER TABLE skola.student2029 ADD CONSTRAINT CK\_Studenti\_Rocnik CHECK (Rocnik BETWEEN 1 AND 4);**

**-- omezení jedinečnost UNIQUE():**

alter table skola.student2029 add RodneCislo int;

ALTER TABLE skola.student2029 ADD CONSTRAINT CK\_Studenti\_RodneCislo **unique** (RodneCislo);

insert into skola.student2029 (Jmeno, Prijmeni, DatumNarozeni, MaRozsirenouVyuku, Rocnik, Skolne)

 values ("Lenka", "Nováková",'2001-12-24',  "ma", 1, 18500);

-- **omezení výchozí hodnoty DEFAULT**  (okamžik vytvoření záznamu)  bez Constraint:

ALTER TABLE skola.student2029 add OkamzikVytvoreni TIMESTAMP **DEFAULT** CURRENT\_TIMESTAMP;

-- odstranění integritních omezení:

alter table skola.student2029 **drop constraint** CK\_Studenti\_Rocnik;

-- vytvoření databáze: **česká řady utf8\_czech\_ci** určuje, jak jsou data řazena a porovnávána:

CREATE DATABASE IF NOT EXISTS voska **COLLATE 'utf8\_czech\_ci';**

-- **odstranění databáze:**

**DROP DATABASE** voska;

INSERT přidá do tabulky [relační databáze](https://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze) nový záznam.

**INSERT** **INTO** <tabulka> [(<sloupec>[,...n])] **VALUES** (<hodnota>[,...n]);

**INSERT** **INTO** <tabulka> [(<sloupec>[,...n])] <**SELECT** prikaz>;

<tabulka> je název tabulky, do které se má nový záznam uložit. <sloupec> je jmenovitý seznam sloupců, do kterých se hodnoty ukládají.

<hodnota> je vkládaná hodnota. Každá hodnota se uloží do sloupce, jež má stejnou pozici ve výčtu sloupců jako tato hodnota.

<SELECT prikaz> je poddotaz pro výběr záznamů z tabulky ([SELECT](https://cs.wikipedia.org/wiki/SELECT)). V této syntaxi se do tabulky vloží výsledek tohoto výběru.

Seznam sloupců v závorkách může být vynechán; v tom případě se předpokládá seznam všech sloupců tabulky. Počet sloupců a hodnot musí být stejný. U sloupců vynechaných v daném seznamu se použije implicitní hodnota. Tu obsahuje definice dané tabulky.

Hodnoty zadané při INSERT dotazu musí splňovat všechny podmínky pro sloupce (např. [primární klíč](https://cs.wikipedia.org/wiki/Prim%C3%A1rn%C3%AD_kl%C3%AD%C4%8D), podmínky CHECK a NOT NULL).

**Příklad 1**

|  |  |  |  |
| --- | --- | --- | --- |
| **jmeno** | **cislo** | **ulice** | **mesto** |
| Jan Novák | 257125474 | Wikipedistická 28 | Pastoriovice |
| Jana Nováková | 574125474 | Luční 6 | Praha |

**INSERT** **INTO** telefonni\_seznam (jmeno, cislo) **VALUES** ('John Doe', '555-1212');

|  |  |  |  |
| --- | --- | --- | --- |
| **jmeno** | **cislo** | **ulice** | **mesto** |
| Jan Novák | 257125474 | Wikipedistická 28 | Pastoriovice |
| Jana Nováková | 574125474 | Luční 6 | Praha |
| John Doe | 555–1212 | NULL | NULL |

Pokud jsou zadány hodnoty všech sloupečků, můžeme použít zkrácenou verzi:

**INSERT** **INTO** tabulka **VALUES** ('hodnota1', [''hodnota2, ...'']);

**Příklad 2**

**INSERT** **INTO** telefonni\_seznam **VALUES** ('John Doe', '555-1212', 'Pařížská 6','Aš');

|  |  |  |  |
| --- | --- | --- | --- |
| **jmeno** | **cislo** | **ulice** | **mesto** |
| Jan Novák | 257125474 | Wikipedistická 28 | Pastoriovice |
| Jana Nováková | 574125474 | Luční 6 | Praha |
| John Doe | 555–1212 | Pařížská 6 | Aš |

**Vložení více záznam**

Některé databáze povolují vložení více záznamů za sebou. V takovém případě se hodnoty pro druhý, … další záznam vloží za ty první a oddělí ,

**INSERT** **INTO** telefonni\_seznam (jmeno, cislo) **VALUES**

('John Doe', '555-1212'),

('Leona Lewis', '555-112777'),

('Joe King', '555-1213');

Většinou je v případě vložení více záznamů povoleno použít i kratší formu s vynechaným seznamem sloupců.

**Vložení výběru**

Povolená je i forma vložení záznamu (nebo záznamů) jejichž sloupce jsou vybrány poddotazem [SELECT](https://cs.wikipedia.org/wiki/SELECT). Poddotaz může použít většinu klauzulí jako pro samostatný dotaz typu SELECT (např. výrazy místo jen názvů sloupců, klauzuli WHERE, klauzuli LIMIT, klauzuli [JOIN](https://cs.wikipedia.org/wiki/JOIN), [agregační funkce](https://cs.wikipedia.org/wiki/Agrega%C4%8Dn%C3%AD_funkce)).

**INSERT** **INTO** telefonni\_seznam (jmeno, cislo)

**SELECT** jmeno, cislo **FROM** telefonni\_seznam\_kopie **WHERE** cislo **LIKE** "555-%" **LIMIT** 3;

**INSERT** **INTO** tabulka\_a (sloupec1a, sloupec2a, …)

**SELECT** sloupec1b, sloupec2b, … **FROM** tabulka\_b [**WHERE** …];

**SELECT** vrací [množinu](https://cs.wikipedia.org/wiki/Mno%C5%BEina) záznamů z jedné anebo více [tabulek](https://cs.wikipedia.org/wiki/Tabulka)

**SELECT**

[**ALL** | **DISTINCT**]

{[tabulka. | **alias**. | pohled.]{\* | sloupec | sloupec **AS** **alias**}

| **AVG**([tabulka. | **alias**. | pohled.]<sloupec>) [**AS** <**alias**>]

| **MIN**([tabulka. | **alias**. | pohled.]<sloupec>) [**AS** <**alias**>]

| **MAX**([tabulka. | **alias**. | pohled.]<sloupec>) [**AS** <**alias**>]

| **COUNT**([tabulka. | **alias**. | pohled.]<\* | sloupec>) [**AS** <**alias**>]

}[,...n]

[**INTO** jméno\_nové\_tabulky]

**FROM** <tabulka> [**AS** <**alias**>][,... n]

[[**INNER** | **FULL**] **JOIN** <tabulka> **ON** <spojovaci podminka>

| <**LEFT** | **RIGHT**> **OUTER** **JOIN** <tabulka> **ON** <spojovaci podminka>

| **CROSS** **JOIN** <sloupce>

[**AS** <**alias**>]

[,... n]]

[**WHERE** <podmínky>

| <sloupec> <**operator**> <sloupec | hodnota>

| <sloupec> <**operator**> <sloupec | hodnota> <**AND** | **OR** | **NOT**> <sloupec> <**operator**> <sloupec | hodnota>

| <sloupec> **BETWEEN** <hodnota> **AND** <hodnota>

| <sloupec> **LIKE** <regularni vyraz>

| <sloupec> **IN** <vycet hodnot>

| <sloupec | vyraz> <**operator**> **ANY** | **SOME** (poddotaz)

| **EXISTS** (poddotaz)]

[**GROUP** **BY** <nazev sloupce>[,... n]]

[**HAVING** <omezujici podminka postavena na vysledcich klauzule **GROUP** **BY**>]

[**ORDER** **BY** <sloupec>[,... n] [**ASC** | **DESC**]]

[**UNION** <**SELECT** dotaz>]

Míra [implementace](https://cs.wikipedia.org/wiki/Implementace) [SQL](https://cs.wikipedia.org/wiki/SQL) dotazů se liší u každého [SŘBD](https://cs.wikipedia.org/wiki/S%C5%98BD), proto je třeba mít při psaní konkrétních dotazů na zřeteli konkrétní SŘBD.

**SELECT** id, zakaznik, cena **FROM** smlouvy **WHERE** cena>10000 **AND** se\_slevou=1 **ORDER** **BY** cena **DESC**

**DISTINCT**

Klíčové slovo DISTINCT z výpisu odstraní záznamy, které se v dané hodnotě pole opakují. Výsledkem pro daný sloupec bude seznam všech hodnot (vyhovující případné podmínce ); každé zastoupené jen jednou. SELECT s DISTINCT vypisuje z logických důvodů většinou jen jedno pole.

**Omezení počtu zobrazených řádků**

Databázové stroje většinou umožňují pomocí nějakého klíčového slova v SQL omezit počet vybraných řádků na určitou hodnotu.

**LIMIT a OFFSET**

Databáze [MySQL](https://cs.wikipedia.org/wiki/MySQL" \o "MySQL), [PostgreSQL](https://cs.wikipedia.org/wiki/PostgreSQL" \o "PostgreSQL) mají klauzuli LIMIT, která kromě maximálního počtu zobrazených řádků umožňuje určit i od jakého místa (ofsetu) z výsledných řádků dotazu (pomyslného celkového výběru) má vracení výsledku začít. Například dotaz

**SELECT** jmeno\_skladby **FROM** zebricek\_skladeb **ORDER** **BY** poslouchanost **LIMIT** 5,10;

by zobrazil záznamy na 5. až 15. místě. Je možné použít

LIMIT s jedním parametrem – maximálním počtem vypsaných řádků

LIMIT se dvěma parametry – první je maximální počet vypsaných řádků a druhý pozice, od které má výpis začínat (offset)

LIMIT v kombinaci s klíčovým slovem OFFSET – místo varianty dvou čísel oddělených čárkou

**SELECT** jmeno\_skladby **FROM** zebricek\_skladeb **ORDER** **BY** poslouchanost **LIMIT** 10 **OFFSET** 5;

**UPDATE** upravuje data v [relační databázi](https://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze). Může být upraven jediný záznam, nebo i více záznamů najednou.

**UPDATE** <tabulka>

**SET** <nazev\_sloupecku> = <hodnota> [,...n]

[**FROM** <zdrojova tabulka (tabulky)>]

[**WHERE** <podminka>]

Aby se příkaz UPDATE úspěšně provedl, musí mít uživatel práva na manipulaci dat v databázi. Nové hodnoty také nesmí kolidovat s podmínkami (např. [primární klíč](https://cs.wikipedia.org/wiki/Prim%C3%A1rn%C3%AD_kl%C3%AD%C4%8D), jedinečný [index](https://cs.wikipedia.org/wiki/Index_(datab%C3%A1ze)), podmínky CHECK a NOT NULL).

|  |  |  |
| --- | --- | --- |
| **C1** | **C2** | **C3** |
| 100 | a | x |
| 105 | b | x |
| 110 | c | x |

Pro nastavení sloupečku *C1* v tabulce *T* na hodnotu 1, ale pouze za podmínky že hodnota sloupečku *C2* je „a“.

**UPDATE** T **SET** C1 = 1 **WHERE** C2 = 'a'

|  |  |  |
| --- | --- | --- |
| **C1** | **C2** | **C3** |
| 1 | a | x |
| 105 | b | x |
| 110 | c | x |

Zvýšení hodnoty sloupečku *C1* o 1 pokud *C2* je „a“.

**UPDATE** T **SET** C1 = C1 + 1 **WHERE** C2 = 'a'

|  |  |  |
| --- | --- | --- |
| **C1** | **C2** | **C3** |
| 2 | a | x |
| 105 | b | x |
| 110 | c | x |

Přidání ke sloupečku *C3* řetězec „text“ pokud *C2* je „a“.

**UPDATE** T **SET** C3 = CONCAT( 'text' , C1 ) **WHERE** C2 = 'a'

|  |  |  |
| --- | --- | --- |
| **C1** | **C2** | **C3** |
| 2 | a | text2 |
| 105 | b | x |
| 110 | c | x |

U příkazu UPDATE se rozlišuje, které hodnoty jsou konstantní po dobu jeho vykonávání a které se mění s každým řádkem. Pokud hodnoty (nebo [výrazy](https://cs.wikipedia.org/wiki/V%C3%BDraz) z nich), které příkaz UPDATE sloupcům přiřazuje, obsahují jména sloupců dané tabulky (nebo tabulek spojených přes [JOIN](https://cs.wikipedia.org/wiki/JOIN)), pak je [systém řízení báze dat](https://cs.wikipedia.org/wiki/Syst%C3%A9m_%C5%99%C3%ADzen%C3%AD_b%C3%A1ze_dat) bere z každého aktuálního řádku. Například pro tabulku produktů se sloupečky cen bez a s DPH příkaz:

**UPDATE** produkty **SET** cena\_s\_DPH = cena\_bez\_DPH \* 1.20;

aktualizuje cenu s DPH na základě ceny bez DPH u každého jednotlivého záznamu.

DELETE sloužící k odstranění záznamů z tabulky [relační](https://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze) [databáze](https://cs.wikipedia.org/wiki/Datab%C3%A1ze).

DELETE FROM <tab\_name> [WHERE <condition>

<tab\_name> je jméno tabulky, ze které budou záznamy odstraněny <condition> je logická podmínka, kterou mají splňovat odstraňované záznamy

DELETE FROM t\_employee WHERE emp\_date\_to < '1.1.2006'

Tento příkaz smaže z tabulky zaměstnanců všechny záznamy zaměstnanců, kteří ukončili pracovní poměr do konce roku 2005.

DELETE FROM t\_loaded\_files\_temp

Tento příkaz smaže všechny záznamy v tabulce t\_loaded\_files\_temp.

**SQL příkazy se dělí na čtyři základní skupiny:**

**Příkazy pro manipulaci s daty**

příkazy pro získání dat z databáze a pro jejich úpravy. **DML** – *Data Manipulation Language*

[SELECT](https://cs.wikipedia.org/wiki/SELECT) – vybírá data z databáze, umožňuje výběr podmnožiny a řazení dat. [INSERT](https://cs.wikipedia.org/wiki/Insert_(SQL)) – vkládá do databáze nová data.

[UPDATE](https://cs.wikipedia.org/wiki/UPDATE) – mění data v databázi (editace).

[MERGE](https://cs.wikipedia.org/wiki/MERGE) – kombinace INSERT a UPDATE – data buď vloží (pokud neexistuje odpovídající [klíč](https://cs.wikipedia.org/wiki/Index_(datab%C3%A1ze))), pokud existuje, pak je upraví ve stylu

[DELETE](https://cs.wikipedia.org/wiki/DELETE) – odstraňuje data (záznamy) z databáze. [SHOW](https://cs.wikipedia.org/wiki/SHOW) - umožňující zobrazit databáze, tabulky

[EXPLAIN](https://cs.wikipedia.org/w/index.php?title=EXPLAIN&action=edit&redlink=1) – speciální příkaz, který zobrazuje postup zpracování SQL příkazu. Pomáhá uživateli optimalizovat příkazy tak, aby byly rychlejší.

**Syntaxe jednoduchého SQL dotazu do tabulky a logické zpracování dat**

Pokud se chceme dotázat SQL skriptem do nějaké tabulky pro určitá data, tak kromě zmíněného DML příkazu [SELECT](https://cs.wikipedia.org/wiki/SELECT) využíváme tzv klauzule. Mezi tyto klauzule patří

[WHERE](https://cs.wikipedia.org/wiki/WHERE) - zadání podmínek a filtrů dat

[GROUP BY](https://cs.wikipedia.org/wiki/GROUP_BY) - seskupování záznamů

[HAVING](https://cs.wikipedia.org/wiki/HAVING) - zadání podmínek nad agregační funkcí

[ORDER BY](https://cs.wikipedia.org/wiki/ORDER_BY) - seřazení dat

**Syntaxe SQL**[**dotazu**](https://biportal.cz/zakladni-sql-dotazy/)[

1. [SELECT](https://cs.wikipedia.org/wiki/SELECT) [sloupec\_1] AS Alias\_1, [sloupec\_2] AS Alias\_2, SUM([sloupec\_3]) AS [Soucet\_sloupec\_3]
2. FROM [tabulka]
3. [WHERE](https://cs.wikipedia.org/wiki/WHERE) [sloupec\_1] like ('%text%')
4. [GROUP BY](https://cs.wikipedia.org/wiki/GROUP_BY) [sloupec\_1], [sloupec\_2]
5. [HAVING](https://cs.wikipedia.org/wiki/HAVING) SUM([sloupec\_3]) > [hodnota]
6. [ORDER BY](https://cs.wikipedia.org/wiki/ORDER_BY) [sloupec\_1]

**Příkazy pro definici dat**

vytvářejí tabulky, [indexy](https://cs.wikipedia.org/wiki/Index_(datab%C3%A1ze)), [pohledy](https://cs.wikipedia.org/wiki/Pohled_(datab%C3%A1ze)),.. lze je i upravovat, mazat. **DDL**–*Data Definition Language*

[CREATE](https://cs.wikipedia.org/wiki/CREATE) – vytváření nových objektů.

[ALTER](https://cs.wikipedia.org/wiki/ALTER) – změny existujících objektů. [DROP](https://cs.wikipedia.org/wiki/DROP) – odstraňování objektů.

**Příkazy pro řízení dat**

nastavování práv přístupu, řízení transakcí **DCL**–*Data Control Language* nebo **TCC** – *Transaction Control Commands*

[GRANT](https://cs.wikipedia.org/wiki/GRANT) – příkaz pro přidělení oprávnění uživateli k určitým objektům.

[REVOKE](https://cs.wikipedia.org/wiki/REVOKE) – příkaz pro odnětí práv uživateli.

[START TRANSACTION](https://cs.wikipedia.org/wiki/START_TRANSACTION) – zahájení transakce. [COMMIT](https://cs.wikipedia.org/wiki/COMMIT) – potvrzení transakce. [ROLLBACK](https://cs.wikipedia.org/wiki/ROLLBACK) – zrušení transakce, návrat do původního stavu.

**příkazy pro řízení přístupových práv** ([GRANT](https://cs.wikipedia.org/wiki/GRANT), [REVOKE](https://cs.wikipedia.org/wiki/REVOKE))

**ostatní** nebo speciální příkazy (číslovače, schémata, ...)

MariaDB je open-source relační databázový systém (RDBMS) vyvinutý jako fork MySQL. Nabízí vysoký výkon, škálovatelnost a rozšiřitelnost. MariaDB je plně kompatibilní s MySQL, což znamená, že většina dotazů a příkazů bude fungovat v obou systémech.

**Charakteristika MariaDB**

1. **Open Source**: MariaDB je open-source, což znamená, že je zdarma k použití a má aktivní komunitu přispěvatelů.
2. **Kompatibilita s MySQL**: MariaDB je navržena tak, aby byla plně kompatibilní s MySQL. To umožňuje snadnou migraci aplikací z MySQL do MariaDB.
3. **Výkon a škálovatelnost**: MariaDB nabízí vylepšení výkonu a škálovatelnosti, což ji činí vhodnou pro použití v rozsáhlých a náročných prostředích.
4. **Podpora pro NoSQL**: MariaDB podporuje hybridní úložiště, což umožňuje kombinovat relační a NoSQL data.
5. **Bezpečnostní funkce**: MariaDB zahrnuje různé bezpečnostní funkce, včetně uživatelských oprávnění, SSL a šifrování.

**Syntaxe a Základní Dotazy CRUD**

**Vytvoření Databáze** CREATE DATABASE example\_db;

**Použití Databáze** USE example\_db;

**Vytvoření Tabulky**

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL,

email VARCHAR(100) NOT NULL,

created\_at TIMESTAMP

);

**Create (INSERT)**

**Vložení Dat** INSERT INTO users (username, email) VALUES ('john\_doe', 'john@example.com');

**Read (SELECT)**

**Výběr Všech Řádků** SELECT \* FROM users;

**Výběr Specifických Sloupců** SELECT username, email FROM users;

**Použití WHERE Klauzule** SELECT \* FROM users WHERE username = 'john\_doe';

**Řazení Výsledků** SELECT \* FROM users ORDER BY created\_at DESC;

**Update (UPDATE)**

**Aktualizace Dat** UPDATE users SET email = 'john\_doe@example.com' WHERE username = 'john\_doe';

**Delete (DELETE)**

**Smazání Dat** DELETE FROM users WHERE username = 'john\_doe';

**Příklad Vytvoření a Použití Databáze a Tabulky**

1. **Vytvoření Databáze**: CREATE DATABASE company\_db;
2. **Použití Databáze**: USE company\_db;
3. **Vytvoření Tabulky**:

CREATE TABLE employees (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

position VARCHAR(50),

salary DECIMAL(10, 2),

hire\_date DATE

);

1. **Vložení Dat**:

INSERT INTO employees (name, position, salary, hire\_date) VALUES

('Alice', 'Manager', 70000, '2022-01-15'),

('Bob', 'Developer', 60000, '2023-03-10'),

('Charlie', 'Analyst', 50000, '2024-05-20');

1. **Výběr Všech Zaměstnanců**: SELECT \* FROM employees;
2. **Aktualizace Platu Zaměstnance**: UPDATE employees SET salary = 65000 WHERE name = 'Bob';
3. **Smazání Zaměstnance**: DELETE FROM employees WHERE name = 'Charlie';

**Příklad s JOIN**

Tabulka departments:

| **id** | **department\_name** |
| --- | --- |
| 1 | IT |
| 2 | HR |
| 3 | Sales |

Tabulka employees:

| **id** | **name** | **department\_id** |
| --- | --- | --- |
| 1 | Alice | 1 |
| 2 | Bob | 2 |
| 3 | Charlie | 3 |

**Výběr Zaměstnanců s Názvy Oddělení**

SELECT employees.name, departments.department\_name FROM employees

JOIN departments ON employees.department\_id = departments.id;

**Příklad s VIEW**

**Vytvoření Pohledu**

CREATE VIEW high\_salary\_employees AS SELECT name, salary

FROM employees WHERE salary > 60000;

**Použití Pohledu** SELECT \* FROM high\_salary\_employees;

**20. SQL - skupina příkazů JOIN (LEFT, RIGHT, atd.) a VIEW, použití a příklady**

**Pohled** (anglicky **View**) je [databázový](https://cs.wikipedia.org/wiki/Datab%C3%A1ze) objekt, který uživateli poskytuje data ve stejné podobě jako [tabulka](https://cs.wikipedia.org/wiki/Tabulka_(datab%C3%A1ze)). Na rozdíl od tabulky, kde jsou data přímo uložena, obsahuje pohled pouze předpis, jakým způsobem mají být data získána z tabulek a jiných pohledů. Z toho vyplývají některé základní rozdíly v chování tabulky a pohledu:

1. Data tabulky lze přímo modifikovat pomocí příkazů [DML SQL](https://cs.wikipedia.org/wiki/SQL) ([INSERT](https://cs.wikipedia.org/wiki/Insert_(SQL)), [UPDATE](https://cs.wikipedia.org/wiki/UPDATE), [DELETE](https://cs.wikipedia.org/wiki/DELETE_(SQL))). Data poskytovaná pohledem nelze obecně vzato přímo modifikovat - výsledek, který pohled poskytuje, se změní v případě, že se změní data v tabulkách, ze kterých pohled čerpá. V některých případech lze pohled modifikovat, pak se mu říká aktualizovatelný pohled. Změny se ovšem v takovém případě promítnou do tabulky (tabulek) na nichž je pohled založen.
2. Na rozdíl od tabulky pohled nezabírá v paměti téměř žádné místo, protože neobsahuje data, ale pouze předpis pro získání dat (obvykle příkaz [SELECT](https://cs.wikipedia.org/wiki/SELECT)).
3. Získání výsledku především u komplikovanějších pohledů může být časově výrazně náročnější než u přímého přístupu k tabulce, neboť data musí být při každém použití pohledu získána z podkladových pohledů a tabulek (výpočet může být často dost složitý). Tento problém lze řešit vytvořením vhodných [indexů](https://cs.wikipedia.org/wiki/Index_(datab%C3%A1ze)) na podkladových tabulkách.
4. Stejně jako ostatní databázové objekty, je i pohled modifikován pomocí příkazů [DDL SQL](https://cs.wikipedia.org/wiki/SQL) ([CREATE](https://cs.wikipedia.org/wiki/CREATE) VIEW…, [ALTER](https://cs.wikipedia.org/wiki/ALTER) VIEW…, [DROP](https://cs.wikipedia.org/wiki/DROP) VIEW…).

|  |
| --- |
|  |

**JOIN** je [syntaktická konstrukce](https://cs.wikipedia.org/wiki/Syntaxe) jazyka [SQL](https://cs.wikipedia.org/wiki/SQL). Slouží ke spojování výsledku dotazu [SELECT](https://cs.wikipedia.org/wiki/SELECT). Pomocí JOINů spojujeme 2 a více tabulek k sobě, abychom mohli vzájemně používat vazby a hodnoty z několika tabulek na jeden výpis. Máme celkem 4 druhy JOINů:

* INNER JOIN - Vypíše pouze obsah společné množiny A a B
* LEFT JOIN - Vypíše celou tabulku A a společný obsah s tabulkou B
* RIGHT JOIN -  Vypíše celou tabulku B a společný obsah s tabulkou A
* FULL OUTER JOIN - Vypíše obě množiny A a B

Tabulka A = uzivatel (protože ji voláme jako první z klauzule FROM)

Tabulka B = role (protože ji voláme jako druhou z klauzule JOIN)

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) uzivatel.jmeno, uzivatel.prijmeni, uzivatel.idrole, role.jmeno FROM uzivatel

**JOIN** role ON uzivatel.idrole = role.idrole;

 - uzivatel.jmeno - stejné jako v objektovém programování. Voláme entitu/tabulku uzivatel a z ní vlastnost jmeno. Kdybychom takto neoznačili správně vlastnosti, SŘBD neví, které jméno k jaké tabulce přiřadit.

- Klauzule ON nám přiřazuje cizí klíč z tabulky uživatel k primárnímu klíči tabulky role, tak aby byla jasná shoda klíčů. Kdybychom takto neprovedli, každý uživatel by ve výpisu měl takový počet řádků se svými hodnotami, kolik je řádků rolí.

-----------------------------------------------------------------------------------------------

**VÝUKA PRO POKROČILÉ**

-- PROPOJENÉ TABULKY -- dvě tabulky:

alter table lenkavojtechova.studenti add IdOboru int;

SET SQL\_SAFE\_UPDATES = 0;

update lenkavojtechova.studenti set IdOboru = (case when Rocnik =1 then  1  when Rocnik =2 then 2

 else 3 end);

 CREATE TABLE IF NOT EXISTS lenkavojtechova.obory (Id INT (11) NOT NULL primary key AUTO\_INCREMENT ,

Nazev varchar(20),

Kod varchar(20),

AkreditacePlatnaDo date,

constraint UQ\_Obory\_Nazev unique(Nazev));

insert into lenkavojtechova.obory (Nazev, Kod, AkreditacePlatnaDo)

 values ("Informatika", "Inf",'2025-12-24'),

 ("Robotika", "Ro",'2025-11-20'),

 ("Databaze", "Da",'2027-05-14');

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) \* from lenkavojtechova.obory;

-- omezení **constraint** cizího klíče **foreingn key**, které hlídá, aby všechny cizí klíče vždy odpovídaly nějakému primárnímu klíči:

**alter table** lenkavojtechova.studenti **add constraint** FK\_Studenti\_IdOboru **foreign key** (IdOboru) **references**  lenkavojtechova.obory(Id);

-- tahání dat z propojených tabulek = [select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452" \o "SELECT) na dvě tabulky:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno, lenkavojtechova.obory.Nazev  from lenkavojtechova.studenti

**join** lenkavojtechova.obory on lenkavojtechova.studenti.IdOboru = lenkavojtechova.obory.Id order by studenti.Prijmeni, studenti.Jmeno;

-- join je úplně to samé jako inner join:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno, lenkavojtechova.obory.Nazev  from lenkavojtechova.studenti

**inner join** lenkavojtechova.obory on lenkavojtechova.studenti.IdOboru = lenkavojtechova.obory.Id order by studenti.Prijmeni, studenti.Jmeno;

-- jak join funguje: 1. kombinace každý s každým  2. ponechání pouze těch kombinací, které splňují podmínku za ON

-- 1. kombinace každý s každým

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) St.Prijmeni, St.Jmeno, Ob.Nazev as Obor, St.IdOboru as StCiziKlic, Ob.Id as ObPrimKlic

from lenkavojtechova.studenti as St, lenkavojtechova.obory as Ob order by St.Prijmeni, St.Jmeno, Ob.Nazev;

-- 2. ponechání pouze těch kombinací, které splňují podmínku za ON (where)     on St.Oboru = Ob.Id

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) St.Prijmeni, St.Jmeno, Ob.Nazev as Obor, St.IdOboru as StCiziKlic, Ob.Id as ObPrimKlic

from lenkavojtechova.studenti as St, lenkavojtechova.obory as Ob where St.IdOboru = Ob.Id order by St.Prijmeni, St.Jmeno, Ob.Nazev;

-- NESPÁROVANÉ ZÁZNAMY: -- vysvětlení joinů = vytvořím řádek bez naplnění IdOboru:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) \* from lenkavojtechova.studenti order by IdOboru;

insert into lenkavojtechova.studenti (Id,Jmeno, Prijmeni, DatumNarozeni, MaRozsirenouVyuku, Rocnik, Skolne)

 values (30, "Lenka", "BezIdOboru",'2001-12-24',  "ma", 1, 18500);

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) \* from lenkavojtechova.studenti order by IdOboru;

-- rozdíl mezi JOIN (NEvypíše řádky, které nemají přiřazený obor):

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno, lenkavojtechova.obory.Nazev  from lenkavojtechova.studenti

**join** lenkavojtechova.obory on lenkavojtechova.studenti.IdOboru = lenkavojtechova.obory.Id order by studenti.Prijmeni, studenti.Jmeno;

-- rozdíl mezi LEFT JOIN (vypíše i řádky, které nemají přiřazený obor):

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno, lenkavojtechova.obory.Nazev  from lenkavojtechova.studenti

**left join** lenkavojtechova.obory on lenkavojtechova.studenti.IdOboru = lenkavojtechova.obory.Id order by studenti.Prijmeni, studenti.Jmeno;

-- oproti bez JOIN (NEvypíše řádky, které nemají přiřazený obor):

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) St.Prijmeni, St.Jmeno, Ob.Nazev as Obor, St.IdOboru as StCiziKlic, Ob.Id as ObPrimKlic

from lenkavojtechova.studenti as St, lenkavojtechova.obory as Ob where St.IdOboru = Ob.Id order by St.Prijmeni, St.Jmeno, Ob.Nazev;

-- symetrické spojení: -- nejdříve vytvořím v číselníku oborů řádek s novým oborem, který dosud neexistoval:

insert into lenkavojtechova.obory (Nazev, Kod, AkreditacePlatnaDo)

 values ("Nový obor1", "N1",'2025-12-24'),

 ("Nový obor2", "N2",'2025-11-20');

-- rozdíl mezi LEFT JOIN (vypíše i řádky, které nemají přiřazený obor, ale nevypíše nové obory):

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno, lenkavojtechova.obory.Nazev  from lenkavojtechova.studenti

**LEFT join** lenkavojtechova.obory on lenkavojtechova.studenti.IdOboru = lenkavojtechova.obory.Id order by studenti.Prijmeni, studenti.Jmeno;

-- MariaDB ve svých některých verzích nemusí přímo podporovat klíčové slovo FULL JOIN.

-- V MariaDB můžete použít kombinaci LEFT JOIN a RIGHT JOIN ve spojení s UNION, aby se simulovalo chování FULL JOIN. -- zapsáno pomocí AS:

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) St.Prijmeni, St.Jmeno, Ob.Nazev

FROM lenkavojtechova.studenti AS St **LEFT JOIN** lenkavojtechova.obory AS Ob ON St.IdOboru = Ob.Id

**UNION**

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) St.Prijmeni, St.Jmeno, Ob.Nazev FROM lenkavojtechova.studenti AS St

**RIGHT JOIN** lenkavojtechova.obory AS Ob ON St.IdOboru = Ob.Id ORDER BY Prijmeni, Jmeno;

-- pravé spojení:

-- rozdíl mezi RIGHT JOIN (NEvypíše řádky, které nemají přiřazený obor, ale vypíše i nové obory):

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno, lenkavojtechova.obory.Nazev  from lenkavojtechova.studenti

**RIGHT join** lenkavojtechova.obory on lenkavojtechova.studenti.IdOboru = lenkavojtechova.obory.Id order by lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno;

-- křížové spojení (= každý s každým) = CROSS JOIN:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno, lenkavojtechova.obory.Nazev  from lenkavojtechova.studenti

**CROSS join** lenkavojtechova.obory order by lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno;

-- křížové spojení (= každý s každým)  omezené ON lenkavojtechova.studenti.IdOboru = lenkavojtechova.obory.Id :

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno, lenkavojtechova.obory.Nazev  from lenkavojtechova.studenti

**CROSS join** lenkavojtechova.obory on lenkavojtechova.studenti.IdOboru = lenkavojtechova.obory.Id  order by lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno;

-- ZMĚNY DAT DLE DRUHÉ TABULKY:  -- mazání dle kritéria z připojené tebulky:

  DELETE lenkavojtechova.studenti from lenkavojtechova.studenti **left** **join** lenkavojtechova.obory on  lenkavojtechova.studenti.IdOboru = lenkavojtechova.obory.Id where lenkavojtechova.obory.Nazev = "Robotika";

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452" \o "SELECT) \* from lenkavojtechova.studenti;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452" \o "SELECT) St.Id, St.Jmeno, St.Prijmeni, St.Rocnik, St.DatumNarozeni, St.Skolne, St.IdOboru, Ob.Nazev

   from lenkavojtechova.studenti as St **LEFT join** lenkavojtechova.obory as Ob on St.Idoboru = Ob.Id order by St.Id;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) lenkavojtechova.studenti.Prijmeni, lenkavojtechova.studenti.Jmeno, lenkavojtechova.obory.Nazev  from lenkavojtechova.studenti

**join** lenkavojtechova.obory on lenkavojtechova.studenti.IdOboru = lenkavojtechova.obory.Id order by studenti.Prijmeni, studenti.Jmeno;

  -- vkládádní do jedné tabulky řádků z druhé tabulky:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452" \o "SELECT) \* from lenkavojtechova.student2029;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452" \o "SELECT) \* from lenkavojtechova.studenti;

  INSERT INTO lenkavojtechova.studenti (Id, Jmeno, Prijmeni, DatumNarozeni, MaRozsirenouVyuku, Rocnik, Skolne, IdOboru)

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) Id, Jmeno, Prijmeni, DatumNarozeni, MaRozsirenouVyuku, 2, Skolne, 2

FROM lenkavojtechova.student2029;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452" \o "SELECT) \* from lenkavojtechova.student2029;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452" \o "SELECT) \* from lenkavojtechova.studenti;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452" \o "SELECT) St.Id, St.Jmeno, St.Prijmeni, St.Rocnik, St.DatumNarozeni, St.Skolne, St.IdOboru, Ob.Nazev

   from lenkavojtechova.studenti as St LEFT join lenkavojtechova.obory as Ob on St.Idoboru = Ob.Id order by St.Id;

-- VKLÁDÁNÍ POZMĚNĚNÝCH ÚDAJU: -- jen návrh:

INSERT INTO lenkavojtechova.obory (Nazev, Kod, AkreditacePlatnaDo)

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) CONCAT(Nazev, '\_N'), Kod, date\_add(AkreditacePlatnaDo, INTERVAL 1 YEAR)

FROM lenkavojtechova.obory where year(AkreditacePlatnaDo)<2026;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) \* from lenkavojtechova.obory;

-- řešení:

alter table lenkavojtechova.obory add IdPuvodnihoOboru int null;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) \* from lenkavojtechova.obory;

alter table lenkavojtechova.obory add constraint FK\_Obory\_IdPuvodnihoOboru foreign key (IdPuvodnihoOboru) references lenkavojtechova.obory(Id);

-- dokončení návrhu:

INSERT INTO lenkavojtechova.obory (Nazev, Kod, AkreditacePlatnaDo, IdPuvodnihoOboru)

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) CONCAT(Nazev, '\_Now'), Kod, date\_add(AkreditacePlatnaDo, INTERVAL 1 YEAR),Id

FROM lenkavojtechova.obory where year(AkreditacePlatnaDo)<2026;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) \* from lenkavojtechova.obory;

-- aktualizace propojené tabulky:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) \* from lenkavojtechova.studenti;

SET SQL\_SAFE\_UPDATES = 0;

update lenkavojtechova.studenti  join lenkavojtechova.obory on lenkavojtechova.obory.IdPuvodnihoOboru = lenkavojtechova.studenti.IdOboru set lenkavojtechova.studenti.IdOboru = lenkavojtechova.obory.Id;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7452) \* from lenkavojtechova.studenti;

V MariaDB, stejně jako v jiných SQL databázových systémech, se příkazy JOIN používají k kombinaci záznamů z dvou nebo více tabulek na základě souvisejících sloupců. Kromě toho lze použít pohledy (views) pro vytvoření virtuálních tabulek, které usnadňují práci s komplexními dotazy

**Typy JOIN a Příklady**

**1. INNER JOIN** vrací pouze ty řádky, které mají odpovídající záznamy v obou tabulkách.

**Příklad:**

Tabulka customers:

| **customer\_id** | **customer\_name** |
| --- | --- |
| 1 | John Doe |
| 2 | Jane Smith |
| 3 | Bob Brown |

Tabulka orders:

| **order\_id** | **customer\_id** | **amount** |
| --- | --- | --- |
| 1 | 1 | 100 |
| 2 | 2 | 150 |
| 3 | 1 | 200 |
| 4 | 4 | 250 |

Dotaz:

SELECT customers.customer\_name, orders.amount FROM customers **INNER JOIN** orders ON customers.customer\_id = orders.customer\_id;

**Výsledek:**

| **customer\_name** | **amount** |
| --- | --- |
| John Doe | 100 |
| John Doe | 200 |
| Jane Smith | 150 |

**2. LEFT JOIN** vrací všechny řádky z levé tabulky a odpovídající řádky z pravé tabulky. Pokud není nalezena žádná shoda, vrací NULL pro pravou tabulku.

**Příklad:**

SELECT customers.customer\_name, orders.amount FROM customers

**LEFT JOIN** orders ON customers.customer\_id = orders.customer\_id;

**Výsledek:**

| **customer\_name** | **amount** |
| --- | --- |
| John Doe | 100 |
| John Doe | 200 |
| Jane Smith | 150 |
| Bob Brown | NULL |

**3. RIGHT JOIN** vrací všechny řádky z pravé tabulky a odpovídající řádky z levé tabulky. Pokud není nalezena žádná shoda, vrací NULL pro levou tabulku.

**Příklad:**

SELECT customers.customer\_name, orders.amount FROM customers

**RIGHT JOIN** orders ON customers.customer\_id = orders.customer\_id;

**Výsledek:**

| **customer\_name** | **amount** |
| --- | --- |
| John Doe | 100 |
| John Doe | 200 |
| Jane Smith | 150 |
| NULL | 250 |

**4. FULL JOIN** MariaDB nativně nepodporuje FULL JOIN, ale lze jej simulovat pomocí kombinace LEFT JOIN a RIGHT JOIN s UNION.

**Příklad:**

SELECT customers.customer\_name, orders.amount FROM customers

**LEFT JOIN** orders ON customers.customer\_id = orders.customer\_id

**UNION**

SELECT customers.customer\_name, orders.amount FROM customers

**RIGHT JOIN** orders ON customers.customer\_id = orders.customer\_id;

**5. CROSS JOIN**

CROSS JOIN vrací kartézský produkt dvou tabulek. To znamená, že každý řádek z první tabulky je kombinován s každým řádkem z druhé tabulky.

**Příklad:**

SELECT customers.customer\_name, orders.amount FROM customers **CROSS JOIN** orders;

**Výsledek:**

| **customer\_name** | **amount** |
| --- | --- |
| John Doe | 100 |
| John Doe | 150 |
| John Doe | 200 |
| John Doe | 250 |
| Jane Smith | 100 |
| Jane Smith | 150 |
| Jane Smith | 200 |
| Jane Smith | 250 |
| Bob Brown | 100 |
| Bob Brown | 150 |
| Bob Brown | 200 |
| Bob Brown | 250 |

**VIEW (Pohledy)** Pohledy jsou virtuální tabulky založené na výsledcích SQL dotazů. Pohledy zjednodušují složité dotazy a poskytují abstrakci nad skutečnými daty.

**Vytvoření Pohledu**

**Syntaxe:**

CREATE VIEW view\_name AS SELECT column1, column2, ... FROM table\_name WHERE condition;

**Příklad:**

-- Vytvoření a naplnění tabulky:

CREATE TABLE employees (

    employee\_id INT PRIMARY KEY,

    name VARCHAR(50),

    department VARCHAR(50),

    salary DECIMAL(10, 2)

);

INSERT INTO employees (employee\_id, name, department, salary) VALUES

(1, 'Alice', 'IT', 5000.00), (2, 'Bob', 'HR', 3000.00),

(3, 'Charlie', 'IT', 6000.00), (4, 'Dave', 'HR', 4000.00), (5, 'Eve', 'Sales', 4500.00);

Tabulka employees:

| **employee\_id** | **name** | **department** | **salary** |
| --- | --- | --- | --- |
| 1 | Alice | IT | 5000 |
| 2 | Bob | HR | 3000 |
| 3 | Charlie | IT | 6000 |
| 4 | Dave | HR | 4000 |
| 5 | Eve | Sales | 4500 |
|  |  |  |  |

Vytvoření pohledu pro zaměstnance v IT oddělení:

**CREATE VIEW** IT\_employees AS SELECT employee\_id, name, salary FROM employees WHERE department = 'IT';

**Použití Pohledu:** Pohledy se používají stejně jako běžné tabulky:

SELECT \* FROM IT\_employees;

**Aktualizace Pohledu** Některé pohledy mohou být aktualizovatelné, což znamená, že můžete vkládat, aktualizovat nebo mazat řádky prostřednictvím pohledu. To závisí na složitosti dotazu a použitých funkcích.Pokud je pohled jednoduchý a bez agregačních funkcí nebo poddotazů, můžete provádět aktualizace:

**UPDATE** IT\_employees SET salary = 7000 WHERE employee\_id = 1;

**Odstranění Pohledu** Pokud již pohled nepotřebujete, můžete jej odstranit:

**DROP VIEW** IF EXISTS IT\_employees;

**Shrnutí**

* **JOIN** příkazy umožňují kombinaci řádků z dvou nebo více tabulek na základě souvisejících sloupců. Typy zahrnují INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN (simulovaný) a CROSS JOIN.
* **VIEW** poskytují způsob, jak zjednodušit a abstraktně pracovat s komplexními dotazy. Pohledy mohou být také použity pro bezpečnost a zlepšení správy dat.

21. SQL - agregační klauzule, podmínky, řazení výstupu, příklady

**Agregační funkce** jsou v [SQL](https://cs.wikipedia.org/wiki/SQL) [statistické](https://cs.wikipedia.org/wiki/Statistika) funkce, pomocí kterých [systém řízení báze dat](https://cs.wikipedia.org/wiki/Syst%C3%A9m_%C5%99%C3%ADzen%C3%AD_b%C3%A1ze_dat) umožňuje seskupit vybrané řádky dotazu (získané příkazem [SELECT](https://cs.wikipedia.org/wiki/SELECT)) a spočítat nad nimi výsledek určité [aritmetické](https://cs.wikipedia.org/wiki/Aritmetika) nebo statistické funkce. Agregační funkce se v SQL používají s konstrukcí [GROUP BY](https://cs.wikipedia.org/wiki/GROUP_BY). Nejčastěji je to [aritmetický průměr](https://cs.wikipedia.org/wiki/Aritmetick%C3%BD_pr%C5%AFm%C4%9Br), [minimum](https://cs.wikipedia.org/wiki/Minimum), [maximum](https://cs.wikipedia.org/wiki/Maximum), složitější, tzv. populační funkce, [směrodatné odchylky](https://cs.wikipedia.org/wiki/Sm%C4%9Brodatn%C3%A1_odchylka) a variance; nebo naopak obyčejný [součet](https://cs.wikipedia.org/wiki/Sou%C4%8Det) či počet řádků, které byly pod každou jednotlivou hodnotou seskupeny. [GROUP BY](https://cs.wikipedia.org/wiki/GROUP_BY) lze výjimečně vynechat, např. pro vypsání výsledku agregační funkce jednoho sloupce nad celou tabulkou.

**Vícenásobné seskupení** Databázové stroje většinou podporují i vícenásobné seskupení / seskupení podle více sloupců (nebo výrazů z nich složených). V takovém případě se názvy sloupců za klíčovým slovem GROUP BY oddělují čárkou. Sloupec (nebo výraz) uvedený jako první má nejvyšší prioritu, poslední sloupec nebo výraz má prioritu nejnižší.

**Nejčastější agregační funkce**

|  |  |
| --- | --- |
| AVG() | [Aritmetický průměr](https://cs.wikipedia.org/wiki/Aritmetick%C3%BD_pr%C5%AFm%C4%9Br) |
| SUM() | Součet |
| COUNT() | Počet |
| MIN() | [Minimum](https://cs.wikipedia.org/wiki/Minimum) |
| MAX() | [Maximum](https://cs.wikipedia.org/wiki/Maximum) |

* Uvnitř závorek se předpokládá použití některého z polí z tabulky, popř. hvězdička (\*), reprezentující celý řádek.
* Povolené jsou také výrazy z názvů polí i literálů, interních a [uživatelských funkcí](https://cs.wikipedia.org/wiki/User_Defined_Function).

GROUP\_CONCAT() je speciální agregační funkce, **která se používá k seskupování hodnot z více řádků do jednoho řetězce. Jejím výsledkem je nikoli počet ale výčet nalezených hodnot, oddělených čárkou nebo jiným oddělovačem.**

**-- VELKÝ TRÉNINK SQL 1** -- AGREGAČNÍ FUNKCE

-- aktuální počet výpůjček:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **count(\*)** as AktualniPocetVypujcek from knihovna.vypujcky where DatunVraceni is null;

-- vypište statistiku čtenářů dle obcí:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Obec, **count(\*)** as Ctenari from knihovna.ctenari group by Obec order by Obec;

-- vypište průměrnou dobu výpůjček:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **AVG(DATEDIFF(DatunVraceni, DatumPujceni ))** as PrumernaDobaVypujcky from knihovna.vypujcky where DatunVraceni is not null;

-- zaokrouhlete na dvě desetiny:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329)  **ROUND(AVG(DATEDIFF(DatunVraceni, DatumPujceni)), 2)** as PrumernaDobaVypujcky from knihovna.vypujcky where DatunVraceni is not null;

-- spočítejte průměrnou dobu výpůjčky bez AVG:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **sum(DATEDIFF(DatunVraceni, DatumPujceni)) / count(DatunVraceni)** as PrumernaDobaVypujcky from knihovna.vypujcky;

-- zaokrouhlete na dvě desetiny:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **round(sum(DATEDIFF(DatunVraceni, DatumPujceni)) / count(DatunVraceni), 2)**  as PrumernaDobaVypujcky from knihovna.vypujcky;

-- PROMĚNNÉ -- zobrazte nejmladšího čtenáře pomocí vnořeného selectu:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Prijmeni, Jmeno, Id from knihovna.ctenari where DatumNarozeni =  ([SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **MAX(DatumNarozeni)** FROM knihovna.ctenari);

-- vlož hodnotu max do proměnné @nej

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) MAX(DatumNarozeni) INTO **@nej** FROM knihovna.ctenari;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Prijmeni, Jmeno, Id, @nej from knihovna.ctenari where DatumNarozeni =  @nej;

-- najděte datum narození nejmladšího a nejstaršího čternáře pomocí proměných:

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) MAX(DatumNarozeni) INTO @nejmladsi FROM knihovna.ctenari;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) **min(DatumNarozeni)** INTO @nejstarsi FROM knihovna.ctenari;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) @nejmladsi, @nejstarsi;

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) \* from knihovna.ctenari;

-- DVĚ TABULKY -- vypiš všechny exempláře konkrétního titulu "%gert%"

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.exemplare.Id as IdExemplare, knihovna.tituly.Nazev, knihovna.tituly.Id as IdTitulu

from knihovna.exemplare **join** knihovna.tituly on exemplare.IdTitulu = tituly.Id  where tituly.nazev like "%gert%"

order by tituly.nazev, tituly.Id, exemplare.Id;

-- vypište všechny autory zadané knihy, když známe IdTitulu knihy 16 :

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.autorititulu.IdAutora from knihovna.autorititulu where IdTitulu = 16 order by Poradi;

-- vypište je jmenovitě:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.autori.Prijemni, knihovna.autori.Jmeno from knihovna.autorititulu

**join** knihovna.autori on knihovna.autorititulu.IdAutora = knihovna.autori.Id  where knihovna.autorititulu.IdTitulu  = 16 order by knihovna.autorititulu.Poradi;

-- jeden autor IdTitulu=1

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.autori.Prijemni, knihovna.autori.Jmeno from knihovna.autorititulu

**join** knihovna.autori on knihovna.autorititulu.IdAutora = knihovna.autori.Id  where knihovna.autorititulu.IdTitulu  = 1 order by knihovna.autorititulu.Poradi;

-- nula autorů Idtitulu=35

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.autori.Prijemni, knihovna.autori.Jmeno from knihovna.autorititulu

**join** knihovna.autori on knihovna.autorititulu.IdAutora = knihovna.autori.Id  where knihovna.autorititulu.IdTitulu  = 35 order by knihovna.autorititulu.Poradi;

-- vypište všechny zaevidované tituly jednoho autota, IdAutora = 16 :

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.tituly.Nazev from knihovna.autorititulu **join** knihovna.tituly on knihovna.autorititulu.IdTitulu = knihovna.tituly.Id  where knihovna.autorititulu.IdAutora  = 16 order by knihovna.tituly.Nazev;

-- VÍCE TABULEK -- výpis knih zapůjčených čtenářem Id = 2 :

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) \* from knihovna.vypujcky where knihovna.vypujcky.IdCtenare = 2 and knihovna.vypujcky.DatunVraceni is null;

-- řešení úplné:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.tituly.Nazev, knihovna.vypujcky.DatumPujceni, knihovna.vypujcky.PujceniDo from knihovna.vypujcky

**join** knihovna.exemplare on knihovna.vypujcky.IdExemplare = knihovna.exemplare.Id

**join** knihovna.tituly on knihovna.exemplare.IdTitulu = knihovna.tituly.Id where knihovna.vypujcky.IdCtenare = 2 and knihovna.vypujcky.DatunVraceni is null order by knihovna.vypujcky.DatumPujceni;

-- zobrazte všechny čtenáře, kteří četli knihu s Idtitulu  = 29 :

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329)  knihovna.ctenari.Prijmeni, knihovna.ctenari.Jmeno, knihovna.ctenari.CisloPrukazu from knihovna.ctenari

**join** knihovna.vypujcky on knihovna.ctenari.Id = knihovna.vypujcky.IdCtenare

**join** knihovna.exemplare on knihovna.vypujcky.IdExemplare = knihovna.exemplare.Id where knihovna.exemplare.IdTitulu = 29  order by knihovna.ctenari.Prijmeni, knihovna.ctenari.Jmeno;

-- řešení úplné (=bez duplikace):

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) distinct knihovna.ctenari.Prijmeni, knihovna.ctenari.Jmeno, knihovna.ctenari.CisloPrukazu from knihovna.ctenari

**join** knihovna.vypujcky on knihovna.ctenari.Id = knihovna.vypujcky.IdCtenare

**join** knihovna.exemplare on knihovna.vypujcky.IdExemplare = knihovna.exemplare.Id where knihovna.exemplare.IdTitulu = 29  order by knihovna.ctenari.Prijmeni, knihovna.ctenari.Jmeno;

-- zobrazte všechny čtenáře, kteří četli knihu s Idtitulu  = 32 :

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) distinct knihovna.ctenari.Prijmeni, knihovna.ctenari.Jmeno, knihovna.ctenari.CisloPrukazu from knihovna.ctenari

**join** knihovna.vypujcky on knihovna.ctenari.Id = knihovna.vypujcky.IdCtenare

**join** knihovna.exemplare on knihovna.vypujcky.IdExemplare = knihovna.exemplare.Id where knihovna.exemplare.IdTitulu = 32  order by knihovna.ctenari.Prijmeni, knihovna.ctenari.Jmeno;

-- zobrazte čtenáře, kteří četli knihu daného autora, tj. známe Id autora :

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) distinct knihovna.ctenari.Prijmeni, knihovna.ctenari.Jmeno, knihovna.ctenari.CisloPrukazu from knihovna.ctenari

**join** knihovna.vypujcky on knihovna.ctenari.Id = knihovna.vypujcky.IdCtenare

**join** knihovna.exemplare on knihovna.vypujcky.IdExemplare = knihovna.exemplare.Id

**join** knihovna.tituly on knihovna.exemplare.IdTitulu = knihovna.tituly.Id

**join** knihovna.autorititulu on knihovna.tituly.Id = knihovna.autorititulu.IdTitulu

where knihovna.autorititulu.IdAutora = 12  order by knihovna.ctenari.Prijmeni, knihovna.ctenari.Jmeno;

**-- VELKÝ TRÉNINK SQL 2** -- SESKUPOVÁNÍ POD LUPOU-- vypište, kolik je od každého titulu v knihovně exemplářů:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.tituly.Nazev, knihovna.tituly.Id, count(knihovna.exemplare.Id) as PocetExemplaru from knihovna.exemplare

**join** knihovna.tituly on knihovna.exemplare.IdTitulu = knihovna.tituly.Id group by knihovna.tituly.Nazev order by knihovna.tituly.Nazev;

-- vypište, kolik je od každého titulu v knihovně exemplářů včetně titulů, které nemají v databázi žádné exempláře:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.tituly.Nazev, knihovna.tituly.Id, count(knihovna.exemplare.Id) as PocetExemplaru from knihovna.exemplare

**right join** knihovna.tituly on knihovna.exemplare.IdTitulu = knihovna.tituly.Id group by knihovna.tituly.Nazev order by knihovna.tituly.Nazev;

-- REDUKCE PŘDRUŽENÝCH ZÁZNAMU -- vypište všechny knihy a jejich autory:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.tituly.Nazev, knihovna.tituly.Id, knihovna.autori.Prijemni, knihovna.autori.Jmeno from knihovna.tituly

**join** knihovna.autorititulu on knihovna.tituly.Id = knihovna.autorititulu.IdTitulu

**join** knihovna.autori on knihovna.autorititulu.IdAutora = knihovna.autori.Id order by knihovna.tituly.Nazev;

-- vypište všechny knihy a jejich autory, i knihy bez autorů:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.tituly.Nazev,knihovna.tituly.Id, knihovna.autori.Prijemni, knihovna.autori.Jmeno from knihovna.tituly

**left join** knihovna.autorititulu on knihovna.tituly.Id = knihovna.autorititulu.IdTitulu

**left join** knihovna.autori on knihovna.autorititulu.IdAutora = knihovna.autori.Id order by knihovna.tituly.Nazev;

-- vypište všechny knihy a jejich autory, i knihy bez autorů     ALE u každé knihy vypište jen prvního autora:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.tituly.Nazev, knihovna.tituly.Id, knihovna.autori.Prijemni, knihovna.autori.Jmeno from knihovna.tituly

**left join** knihovna.autorititulu on knihovna.tituly.Id = knihovna.autorititulu.IdTitulu

**left join** knihovna.autori on knihovna.autorititulu.IdAutora = knihovna.autori.Id

where knihovna.autorititulu.Poradi = 1 or knihovna.autorititulu.Poradi is null order by knihovna.tituly.Nazev;

-- jiný zápis téhož (složená podmínka v JOIN):

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.tituly.Nazev, knihovna.tituly.Id, knihovna.autori.Prijemni, knihovna.autori.Jmeno from knihovna.tituly

**left join** knihovna.autorititulu on knihovna.tituly.Id = knihovna.autorititulu.IdTitulu and  knihovna.autorititulu.Poradi = 1

**left join** knihovna.autori on knihovna.autorititulu.IdAutora = knihovna.autori.Id order by knihovna.tituly.Nazev;

-- předpokládali jsme, že každý záznam začíná jedničkou -- NIKDY    x   ANI JEDNOU:

-- vypište všechny tituly, které nemají přidružený exemplář:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) Nazev, Id from knihovna.tituly where NOT exists

([select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329" \o "SELECT) 1 from knihovna.exemplare where knihovna.exemplare.IdTitulu = knihovna.tituly.Id)

order by Nazev, Id;

-- řešení pomocí join:

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) knihovna.tituly.Nazev, knihovna.tituly.Id from knihovna.tituly

**left join** knihovna.exemplare  on knihovna.exemplare.IdTitulu = knihovna.tituly.Id

where knihovna.exemplare.Id IS null order by knihovna.tituly.Nazev, knihovna.tituly.Id;

-- vypište tituly, které jsou momentálně vypůjčené (bez duplicit):

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) distinct knihovna.tituly.Nazev, knihovna.tituly.Id from knihovna.tituly

**join** knihovna.exemplare  on knihovna.exemplare.IdTitulu = knihovna.tituly.Id

**join** knihovna.vypujcky on knihovna.vypujcky.IdExemplare = knihovna.exemplare.Id

where knihovna.vypujcky.DatunVraceni IS null order by knihovna.tituly.Nazev, knihovna.tituly.Id;

-- jiný zápis téhož (složená podmínka v JOIN):

[select](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=6329) distinct knihovna.tituly.Nazev, knihovna.tituly.Id from knihovna.tituly

**join** knihovna.exemplare  on knihovna.exemplare.IdTitulu = knihovna.tituly.Id

**join** knihovna.vypujcky on knihovna.vypujcky.IdExemplare = knihovna.exemplare.Id and knihovna.vypujcky.DatunVraceni IS null

 order by knihovna.tituly.Nazev, knihovna.tituly.Id;

**COUNT** Funkce COUNT() se od ostatních agregačních funkcí v několika věcech odlišuje:

* Zatímco ostatní svůj výsledek vrací nad specifikovanými sloupci nebo výrazy z nich, COUNT() **vrací počet záznamů**, které vyhovují zadané podmínce resp. seskupení, a proto je jedno, který ze sloupců má jako argument (a často se používá hvězdičková konvence).
* Výjimkou výše uvedeného bodu je případ, kdy je potřeba vrátit počet unikátních hodnot určitého sloupce použitého v tabulce. Pak se ke COUNT() přidává klíčové slovo DISTINCT:

Příklad:

**SELECT** **COUNT**(**DISTINCT** zeme) **FROM** navstevnici;

*-- Z kolika zemí přicházejí návštěvníci?*

* V případech, že nebyl nalezen ani jeden odpovídající řádek, COUNT() vrátí nulu (ostatní agregační funkce vrací hodnotu [NULL](https://cs.wikipedia.org/wiki/NULL)).
* COUNT() může být v SQL voláno bez konstrukce [GROUP BY](https://cs.wikipedia.org/wiki/GROUP_BY). Touto výjimkou je SQL dotaz nad celou tabulkou (např. dotaz na maximální hodnotu určitého sloupce z celé tabulky), který vrací jeden řádek:

**SELECT** **COUNT**(\*) **FROM** zamestnanci;

*-- Vrať počet řádků v tabulce `zamestnanci`.*

**SUM** Funkce SUM() **vrací**[**sumaci**](https://cs.wikipedia.org/wiki/Sumace)**vybraných řádků**. Je-li jejich počet nulový, je vrácena nikoli nula, ale hodnota [NULL](https://cs.wikipedia.org/wiki/NULL). Pro SUM() umožňuje řada databázových systémů použít také logické (pravdivostní) výrazy (z [Booleovy algebry](https://cs.wikipedia.org/wiki/Booleova_algebra" \o "Booleova algebra) – např. výsledky rovnosti, nerovnosti, apod.), u kterých vrací jedničku pro „pravdu“ a nulu pro „nepravdu“. Takto lze zjistit počet řádků v tabulce, jejichž prvky splňují zadanou podmínku.

**GROUP\_CONCAT** GROUP\_CONCAT() je speciální agregační funkce, **která se používá k seskupování hodnot z více řádků do jednoho řetězce. Jejím výsledkem je nikoli počet ale výčet nalezených hodnot, oddělených čárkou nebo jiným oddělovačem.** Pro různé číselníky apod. tak může být GROUP\_CONCAT() velice užitečná – bez ní by bylo potřeba hodnoty vybrat jiným SQL dotazem, výsledek projít záznam po záznamu a hodnoty zapsat jednu za druhou do pomocné řetězcové proměnné. U příliš obsáhlých tabulek může ovšem výsledek přesahovat maximum toho, co databázový systém může vrátit, a je třeba na to dávat pozor. Jako u COUNT() i u této funkce lze GROUP\_CONCAT() kombinovat s klíčovým slovem DISTINCT pro eliminaci vícekrát se vyskytujících hodnot. Navíc lze výčet seřadit (vložením klauzule [ORDER BY](https://cs.wikipedia.org/wiki/ORDER_BY)) a též si přizpůsobit formát výpisu specifikováním jiného oddělovače než výchozí čárky (uvedeným v klauzuli SEPARATOR).

**SELECT** student\_name,

GROUP\_CONCAT(**DISTINCT** test\_score **ORDER** **BY** test\_score **DESC** SEPARATOR ' ')

**FROM** student

**GROUP** **BY** student\_name;

*-- Z tabulky studentů vypíše jméno a unikátní skóre, seřazené sestupně, oddělené mezerou.*

**Konstrukce HAVING, za kterou následují omezující podmínky, umožňuje omezit řádky, které jsou zpracovány agregační funkcí. Je tedy možné ji použít jako náhradu za konstrukci WHERE, kde agregační funkce nejsou povoleny.** Návrh [SQL](https://cs.wikipedia.org/wiki/SQL) používá dvě různé konstrukce pro omezující podmínky podle přítomnosti agregačních funkcí proto, že agregovaný výběr se od toho běžného liší (použitím rozdílných postupů, [algoritmů](https://cs.wikipedia.org/wiki/Algoritmus), výkonem, atd.). Omezující podmínky konstrukce HAVING obsahující alespoň jednu agregační funkci musejí být uvedeny až za GROUP BY.

**Syntaxe HAVING:**

**SELECT** zeme, **COUNT**(\*) **AS** Pocet

**FROM** zamestnanci

**GROUP** **BY** zeme

**HAVING** **COUNT**(\*)>1

**Agregační funkce a NULL**

* Hodnota [NULL](https://cs.wikipedia.org/wiki/NULL) je absorpční prvek pro statistické agregační funkce jako Min(), Max(), Avg(), a další. To znamená, že **pokud se ve výběru, ze kterého se agregační funkce počítá, vyskytne NULL, výsledek je pak vždy NULL.**
* Pro funkci Count() je důležitý počet řádků, nikoli, zdali hodnota jejího parametru nabývá NULL, vrací tedy vždy číslo.
* Agregační funkce s výjimkou Count() vracejí NULL, pokud počet řádků, které pro ně byl vybrán, je nulový.
* NULL se objevuje i na pozici agregovaného sloupce nebo výrazu v dotazu s konstrukcí GROUP BY (výraz) [WITH ROLLUP](https://cs.wikipedia.org/wiki/WITH_ROLLUP), kde NULL reprezentuje všechny předtím vybrané případy dohromady

**Agregační Klauzule**

Agregační klauzule slouží k provádění výpočtů na více řádcích a vracení jedné hodnoty. Běžné agregační funkce zahrnují **COUNT(), SUM(), AVG(), MIN(), MAX() a GROUP\_CONCAT()**.

**Příklad:**

Tabulka orders:

| **order\_id** | **customer** | **amount** |
| --- | --- | --- |
| 1 | John | 150 |
| 2 | Alice | 200 |
| 3 | John | 250 |
| 4 | Bob | 100 |
| 5 | Alice | 300 |

Dotaz pro zjištění celkových objednávek od každého zákazníka:

SELECT customer, **SUM(amount)** AS total\_amount FROM orders GROUP BY customer;

**Podmínky**

Podmínky slouží k filtrování řádků, které mají být zahrnuty ve výsledku dotazu. Existují klauzule WHERE a HAVING.

**WHERE** Klauzule WHERE se používá k filtrování řádků před agregací.

Příklad:

SELECT customer, SUM(amount) AS total\_amount FROM orders **WHERE amount > 100** GROUP BY customer;

**HAVING**Klauzule HAVING se používá k filtrování výsledků po agregaci.

Příklad:

SELECT customer, SUM(amount) AS total\_amount FROM orders GROUP BY customer

**HAVING total\_amount > 300**;

**Řazení Výstupu**

K řazení výsledků dotazu se používá klauzule ORDER BY. Výsledky lze řadit vzestupně (ASC) nebo sestupně (DESC).

**Příklad:**

SELECT customer, SUM(amount) AS total\_amount FROM orders GROUP BY customer

**ORDER BY** total\_amount **DESC**;

**Kompletní Příklad** Kombinace všech klauzulí v jednom dotazu.

Tabulka employees:

| **employee\_id** | **department** | **salary** |
| --- | --- | --- |
| 1 | IT | 5000 |
| 2 | HR | 3000 |
| 3 | IT | 6000 |
| 4 | HR | 4000 |
| 5 | Sales | 4500 |

Dotaz, který najde průměrný plat v každém oddělení, jen pro oddělení s průměrným platem vyšším než 4000, a seřadí výsledky podle průměrného platu sestupně:

SELECT department, **AVG(salary)** AS average\_salary FROM employees GROUP BY department

HAVING AVG(salary) > 4000 ORDER BY average\_salary DESC;

**Výsledek:**

| **department** | **average\_salary** |
| --- | --- |
| IT | 5500 |
| Sales | 4500 |

Tento dotaz:

1. **Seskupí řádky podle oddělení**: GROUP BY department.
2. **Vypočítá průměrný plat pro každé oddělení**: AVG(salary).
3. **Filtruje výsledky, aby zobrazil pouze oddělení s průměrným platem vyšším než 4000**: HAVING AVG(salary) > 4000.
4. **Seřadí výsledky podle průměrného platu v sestupném pořadí**: ORDER BY average\_salary DESC.

**Další Příklady**

**Počet objednávek od každého zákazníka**

Tabulka orders:

SELECT customer, **COUNT(\*)** AS number\_of\_orders FROM orders GROUP BY customer;

**Celkový a průměrný plat v každém oddělení, seřazený podle celkového platu**

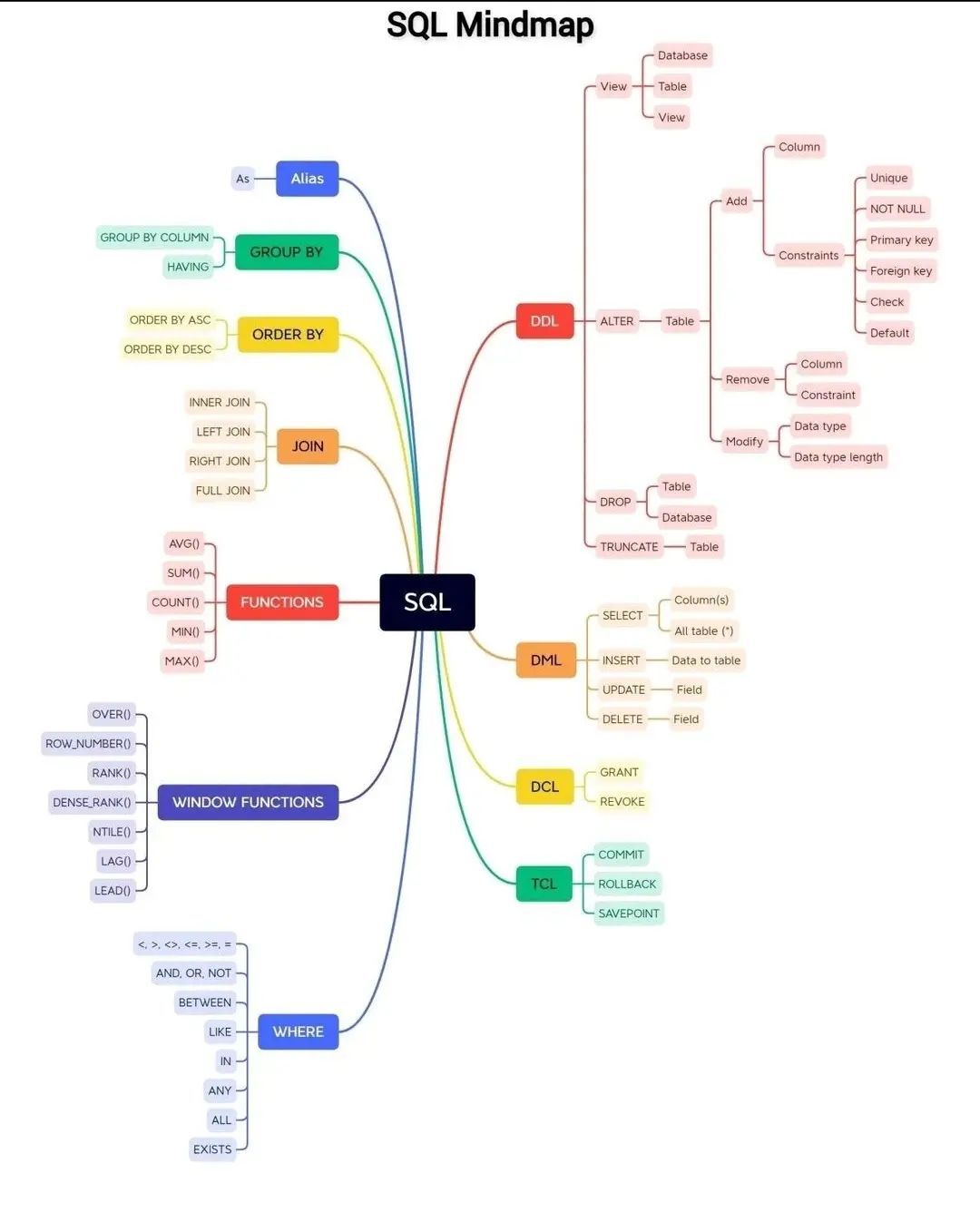
Tabulka employees:

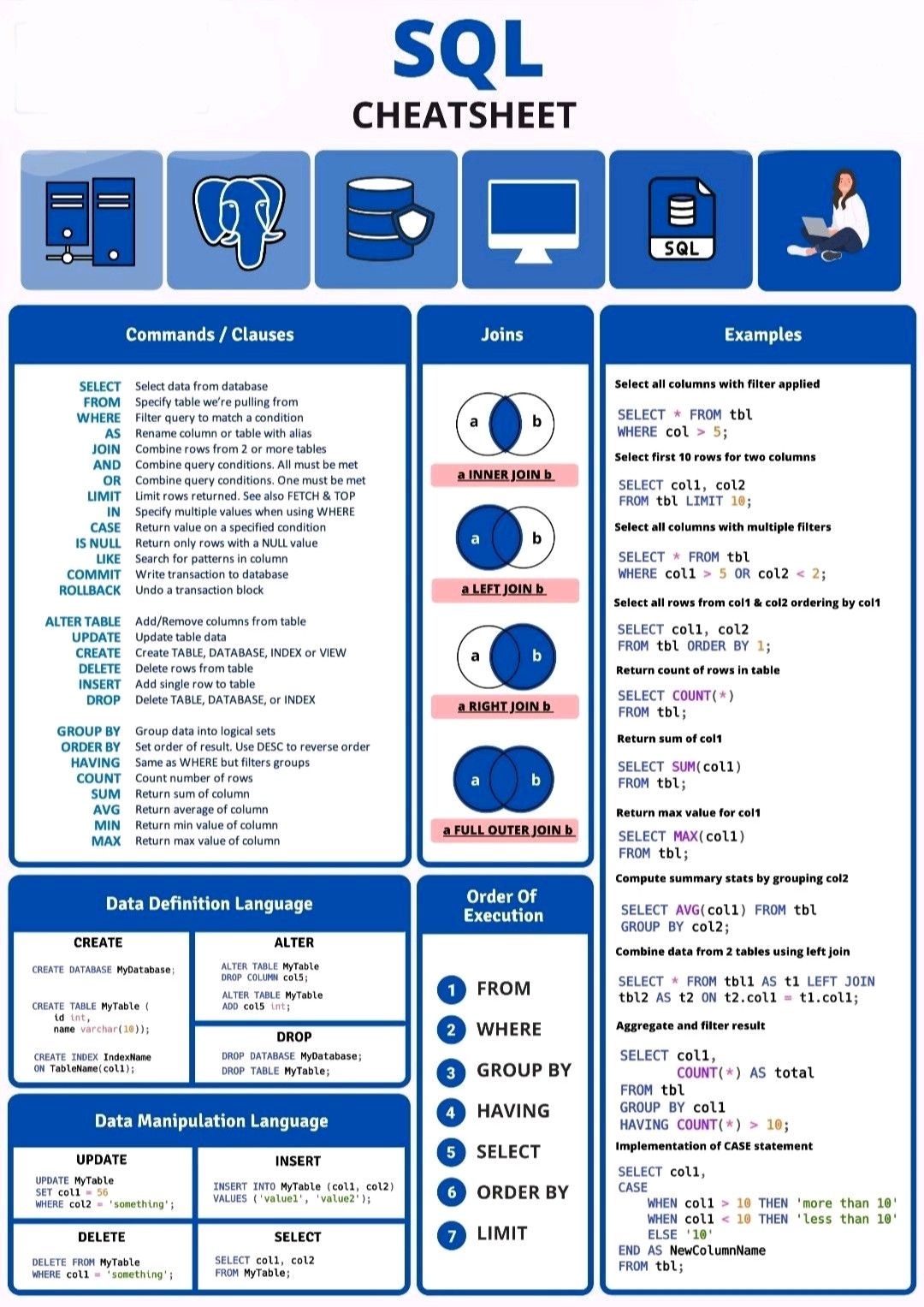
SELECT department, SUM(salary) AS total\_salary, AVG(salary) AS average\_salary

FROM employees GROUP BY department ORDER BY total\_salary DESC;

**Výsledek:**

| **department** | **total\_salary** | **average\_salary** |
| --- | --- | --- |
| IT | 11000 | 5500 |
| Sales | 4500 | 4500 |
| HR | 7000 | 3500 |





**Výuka agregačních funkcí 1** -- Vytvoření databáze:

CREATE DATABASE sales; USE sales;

-- Vytvoření tabulky a naplnění daty

CREATE TABLE FactInternetSales (

    SalesOrderID INT PRIMARY KEY,

    OrderDate DATETIME,

    SalesAmount DECIMAL(10, 2));

INSERT INTO FactInternetSales (SalesOrderID, OrderDate, SalesAmount) VALUES

(1, '2023-01-01 10:00:00', 100.00), (2, '2023-01-01 12:00:00', 150.00), (3, '2023-01-02 14:00:00', 200.00),

(4, '2023-01-02 16:00:00', 250.00), (5, '2023-01-03 18:00:00', 300.00);

-- Vytvoření dotazu s agregacemi

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7619)

-- funkce CAST přecede hodnoty OrderDate z typu DATETIME (obsahuje datum a čas) na typ DATE (obsahuje pouze datum):

    CAST(OrderDate AS DATE) AS Datum\_Objednavky,

-- agregační funkce:

    SUM(SalesAmount) AS Soucet\_trzeb, AVG(SalesAmount) AS Prumer\_trzeb, COUNT(SalesAmount) AS Pocet\_trzeb, MAX(SalesAmount) AS Maximalni\_trzba, MIN(SalesAmount) AS Minimalni\_trzba FROM FactInternetSales GROUP BY Datum\_Objednavky ORDER BY Datum\_Objednavky;

[SELECT](https://moodle.skolakrizik.cz/mod/lesson/view.php?id=7619)

-- Převádí hodnotu sloupce OrderDate z DATETIME na DATE, pojmenuje tento nový sloupec jako Datum\_Objednavky

    CAST(OrderDate AS DATE) AS Datum\_Objednavky,

    SUM(SalesAmount) AS Soucet\_trzeb,

    AVG(SalesAmount) AS Prumer\_trzeb,

    COUNT(SalesAmount) AS Pocet\_trzeb,

    MAX(SalesAmount) AS Maximalni\_trzba,

    MIN(SalesAmount) AS Minimalni\_trzba

FROM FactInternetSales

-- WHERE filtruje záznamy před skupinováním; vybíráme pouze záznamy, kde OrderDate je po 1. lednu 2023

WHERE OrderDate > '2023-01-01'

-- GROUP BY seskupuje záznamy podle převažujícího datového sloupce Datum\_Objednavky

GROUP BY Datum\_Objednavky

-- HAVING filtruje skupiny po skupinování; vybíráme pouze ty skupiny, kde součet tržeb je větší než 400

HAVING SUM(SalesAmount) > 400

-- ORDER BY seřazuje výsledky podle Datum\_Objednavky

ORDER BY Datum\_Objednavky;

**Výuka agregačních funkcí 2 - GROUP\_CONCAT()**

CREATE DATABASE company;

USE company;

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

department VARCHAR(50),

salary DECIMAL(10, 2));

INSERT INTO employees (employee\_id, first\_name, last\_name, department, salary) VALUES

(1, 'Alice', 'Smith', 'IT', 5000.00),

(2, 'Bob', 'Johnson', 'HR', 3000.00),

(3, 'Charlie', 'Smith', 'IT', 6000.00),

(4, 'Dave', 'Williams', 'HR', 4000.00),

(5, 'Eve', 'Brown', 'Sales', 4500.00);

-- funkce DISTINCT() slouží k odstranění duplicitních záznamů v výsledku dotazu:

SELECT DISTINCT

-- funkce CONCAT() slouží ke zřetězení first\_name a last\_name s mezerou a dává tomuto výsledku alias full\_name:

CONCAT(first\_name, ' ', last\_name) AS full\_name, department FROM employees

-- Seřazuje výsledky podle full\_name:

ORDER BY full\_name;

-- Funkce CONCAT() a GROUP\_CONCAT() jsou obě používány ke spojení (zřetězení) hodnot do jednoho řetězce, ale mají odlišné použití a chování:

-- funkce CONCAT() se používá k zřetězení řetězců v jednom řádku:

SELECT CONCAT(first\_name, ' ', last\_name) AS full\_name FROM employees;

-- Funkce GROUP\_CONCAT() je používána ve spojení s funkcí skupinování (GROUP BY) a slouží k sloučení hodnot ze skupiny do jednoho řetězce. -- Může být použita pouze s agregovanými funkcemi (jako je GROUP BY).-- Implicitně odděluje hodnoty čárkou. -- Používáme ji k vytvoření jediného řetězce ze všech hodnot ve skupině:

SELECT department, GROUP\_CONCAT(employee\_id) AS employee\_ids FROM employees GROUP BY department;

-- Použití GROUP\_CONCAT() ke zřetězení jedinečných (distinct) kombinací first\_name a last\_name, oddělených , :

SELECT department, GROUP\_CONCAT(DISTINCT CONCAT(first\_name, ' ', last\_name) ORDER BY last\_name SEPARATOR ', ') AS employee\_names FROM employees GROUP BY department ORDER BY department;