SQL

SQL znamená *Structured Query Language*, tedy jazyk pro strukturované dotazy. Jazyk slouží pro dotazování se na data v relačních databázích. Umožňuje základní operace nad daty: vytvoření, získání, změnu a odstranění dat. Také slouží k definici toho, jak data mají vypadat a umožňuje pokročilé programování.

T-SQL

Jazyk SQL je standardizovaný, (má svoji normu). Reálné databázové systémy (MariaDB, PostgreSQL, Microsoft SQL, Oracle atd.) tuto normu z větší části dodržují, existují však drobné odchylky.

Velká většina textu platí obecně, někde však použijeme speciality z jazyka pro Microsoft SQL s názvem T-SQL (Transact-SQL). Jeho dokumntace je na adrese https://learn.microsoft.com/en-us/sql/t-sql/.

Zdroje k výuce

Odkazy na zdroje, které ve výuce používáme a/nebo obsahují informace k výuce.

Vzorové databáze Microsoft

Repozitář se vzorovými databázemi: https://github.com/microsoft/sql-server-samples/tree/master/samples/databases

V tomto repozitáři je i skript pro vytvoření vzorové databáze Pubs: https://github.com/microsoft/sql-server-samples/blob/master/samples/databases/northwind-pubs/instpubs.sql

Microsoft Visual Studio

Vývojové prostředí Microsoft Visual Studio Community Edition je ke stažení zdarma na adrese https://visualstudio.microsoft.com/cs/vs/community. Při standardní instalaci je nainstalována i lokální databáze. V rámci vývojového IDE je dostupný i SQL server browser.

Kurz jazyka SQL

Na internetu v případě zájmu dohledáte mnoho kurzů SQL. Web w3schools nabízí kromě stručného tutoriálu i vzorovou databázi a online editor SQL příkazů.

Odkaz: https://www.w3schools.com/sql/default.asp

Microsoft Management Studio

Pokročilý nástroj na správu MSSQL databází. Odkaz: https://learn.microsoft.com/en-us/ssms/install/install

Relační databáze

Relační databáze je jedním z typů databází. Databáze obecně je počítačový systém k ukládání a následnému získávání údajů, tedy dat.

V relačních databázích jsou data ukládána do tabulek, které si můžeme představit jako formátované tabulky Excelu s několika dalšími vlastnostmi:

- každá tabulka se skládá z definice sloupců (představíme si jako nadpis sloupce v Excelu) a samotných dat (řádky tabulky),
- definice sloupců obsahuje nejen název sloupce, ale i datový typ sloupce a případné omezení pro data (např. ve sloupci *Narozen* bude uložena hodnota typu datum, ve sloupci *Mohsova tvrdost* bude celé kladné číslo menší nebo rovno 10).

Teoretické pojmy:

- *Relace* je tabulka v databázi,
- *Atribut* je definice sloupce v tabulce, má určen *datový typ* (číslo, řetězec, datum, ...) a *doménu* (povolené hodnoty),
- *Záznam* je řádek dat tabulky.

Příklad

Za použití teoretických pojmů:

Relace *Studenti* má **atribut** *Příjmení* **datového typu** nvarchar (znaky) o maximální délce 100, tedy např. hodnota "Novák" patří do **domény** tohoto datového typu, hodnota 1.1.1980 nepatří do **domény** tohoto datového typu.

Přeloženo znamená:

V **tabulce** *Studenti* je **sloupec** *Příjmení*: může obsahovat nejvýše 100 znaků. "Novák" mohu zapsat, 1.1.1980 ne (to je datum)

Příklad

Relace *Materiály* obsahuje **atribut** *Tvrdost* **datového typu** celé číslo, jeho **doména** je množina {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}.

Přeloženo znamená:

V **tabulce** *Materiály* je **sloupec** *Tvrdost*, povolené hodnoty jsou celá čísla od 1 do 10.

Klíč

Klíč slouží jako jednoznačná identifikace záznamu v tabulce. To znamená, že když známe klíč, můžeme podle něj najít správný řádek v tabulce. Také to znamená, že v tabulce nejsou dva záznamy se stejným klíčem.

Příklad

V tabulce *Studenti* může být klíčem např. sloupec *Číslo ISIC*. Když znám číslo ISIC, mohu podle něj najít studenta.

Klíč může být tvořen více sloupci v tabulce.

Příklad

V tabulce *Bankovní účty* může být klíčem dvojice sloupců: *Číslo účtu* a *kód banky*.

Příklad

V tabulce *Adresy* může být klíčem pětice sloupců *Stát*, *Město*, *Ulice*, *Číslo popisné*, *Číslo bytu*.

V relačních databázích rozeznáváme několi druhů klíčů.

Kandidátní klíč (Candidate key)

Kandidátní klíč jsou jakékoli sloupce, které jednoznačně identifikují záznam. Jedna tabulka může obsahovat více kandidátních klíčů.

Příklad

Tabulka *Studenti* obsahuje kandidátní klíč *Rodné číslo*, dalším kandidátním klíčem je sloupec *Číslo ISIC*. (Studenta mohu najít podle rodného čísla, studenta také mohu najít podle čísla ISIC).

Primární klíč (Primary key)

Primární klíč je jeden z kandidátních klíčů, který si vybereme a budume nadále používat jako "vyvolený".

Příklad

Tabulka *Studenti* má primární klíč *Rodné číslo*

nebo také:

Příklad

Tabulka *Studenti* má primární klíč *Číslo ISIC*

Je na autorovi návrhu databáze, pro který z kandidátních klíčů se rozhodne. Měl by brát v úvahu případné budoucí komplikace (např. zahraniční student nemá rodné číslo nebo student ztratí kartu ISIC a je mu vystavena nová s jiným číslem apod.).

Poznámka

Velmi často se do tabulek uměle přidává sloupec s názvem *ID* (identifikátor) a datovým typen GUID (Globally Unique Identifier). V jazyce T-SQL mají sloupce obsahující GUID datový typ s názvem *uniqueidentifier*. Jedná se o číslo délky 16 bytů (většinou se zapisuje jako 32 hexadecimálních číslic). Databáze obsahují

funci pro přidělení nového GUID takovým způsobem, aby bylo "jedinečné v celém vesmíru" - je téměř nemožné standardním postupem získat dva stejné GUID. V T-SQL nový GUID vrací funkce *newid()*.

Cizí klíč (Foreign key)

Cizí klíč je takový sloupec tabulky, který obsahuje hodnotu primárního klíče z jiné tabulky.

Příklad

V databázi centrální evidence automobilů jsou tabulky:

- tabulka *Provozovatele* se sloupci *Jméno*, *Příjmení*, *Rodné číslo* ,
- tabulka *Vozidla* se sloupci *SPZ*, *Značka vozidla*, *Rodné číslo provozovatele*.

Sloupec *Rodné číslo* v tabulke *Provozovatele* je primární klíč. Sloupec *Rodné číslo provozovatele* v tabulce *Vozidla* je cizí klíč do tabulky *Provozovatele*.

SELECT (jedna tabulka)

Příkaz SELECT umožňuje vybírat záznamy z více tabulek. V této části budeme pro jednoduchost popisovat výběr z jedné tabulky, později probereme i výběr z více tabulek.

Struktura příkazu

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

SELECT	jaké sloupce vybírám, jaké výpočty provádím
FROM	z jaké tabulky (tabulek) vybírám
WHERE	podmínka, kterou musí záznamy splňovat PŘED sdružením záznamů (GROUP BY)
GROUP BY	podle jakých sloupečků sdružuji záznamy při výpočtech
HAVING	podmínka, kterou musí záznamy splňovat PO sdružení záznamů (GROUP BY)
ORDER BY	Jak seřadím výsledek

SELECT, FROM

Jaké sloupečky (SELECT) ze které tabulky nebo tabulek (FROM) vybírám.

Příklad

```
SELECT kontakty.jmeno, kontakty.prijmeni
FROM kontakty
```

Pokud je jméno sloupce jednoznačné, je možné použít zkrácený tvar bez jména tabulky:

```
SELECT jmeno, prijmeni
FROM kontakty
```

DISTINCT

Vynechá z výsledné tabulky duplicitní řádky.

Příklad

Pokud tabulka kontakty obsahuje záznamy.

Id	*jmeno*	*prijmeni*
1	Jan	Novák
2	Petr	Novák
3	Jan	Dvořák

Příklad

Dotaz BEZ použití DISTINCT

```
SELECT jmeno
FROM kontakty
```

vrátí záznamy

Jan
Petr
Jan

Dotaz s použitím DISTINCT

```
SELECT DISTINCT jmeno
FROM kontakty
```

vrátí záznamy

Jan

Petr

TOP

Vrátí pouze tolik řádků, kolik je uvedeno.

Příklad

```
SELECT TOP 5 jmeno, prijmeni
FROM kontakty
```

Téměř vždy se používá spolu s ORDER BY (viz níže), jinak je výsledek nepředvídatelný.

CASE

Používá se pro výpočet nové hodnoty ve sloupci na základě podkladové hodnoty.

Příklad

```
SELECT
    misto,
    datum,
    cas,
    oblacnost,
    CASE
        WHEN oblacnost <= 1/8.0 THEN 'jasno'
        WHEN oblacnost <= 2/8.0 THEN 'skoro jasno'
        WHEN oblacnost <= 3/8.0 THEN 'malá oblačnost'
        WHEN oblacnost <= 4/8.0 THEN 'polojasno'
        WHEN oblacnost <= 5/8.0 THEN 'oblačno'
        WHEN oblacnost <= 6/8.0 THEN 'velká oblačnost'
        WHEN oblacnost <= 7/8.0 THEN 'skoro zataženo'
        ELSE 'zataženo'
    END AS oblacnost_slovy,
    teplota,
FROM predpoved
```

AS (ALIAS)

Slouží k dočasnému přejmenování tabulky a/nebo přejmenování sloupce výsledku dotazu.

Příklad (alias sloupce):

```
SELECT jmeno1 AS krestni,  prijmeni
FROM kontakty
```

Příklad (alias tabulky):

```
SELECT kon_cr.jmeno, kon_cr.prijmeni
FROM kontakty_na_zakazniky_z_ceske_republiky AS kon_cr
```

Příklad (alias sloupce i tabulky):

```
SELECT kon_cr.jmeno AS krestni, kon_cr.prijmeni
FROM kontakty_na_zakazniky_z_ceske_republiky AS kon_cr
```

Ve většině implementací SQL je možné klíčové slovo AS vypustit, předchozí příklad lze zapsat:

```
SELECT kon_cr.jmeno krestni, kon_cr.prijmeni
FROM kontakty_na_zakazniky_z_ceske_republiky kon_cr
```

WHERE

Příklad

```
SELECT jmeno, prijmeni
FROM kontakty
WHERE prijmeni = 'Novák'
```

Podmínky se dají spojovat operátory AND, OR, NOT

```
SELECT jmeno, prijmeni
FROM kontakty
WHERE prijmeni = 'Novák' AND jmeno = 'Jan'
```

Základní operátory

Mohu použít operátory:

=	rovná se
<>	nerovná se, v MSSQL lze použít i !=
>	větší
<	menší
>=	větší nebo rovno
<=	menší nebo rovno

Všimněte si:

- řetězce uzavíráme do jednoduchých uvozovek, tedy ‘Novák’, nikoli “Novák”
- rovnost se zapisuje jedním symbolem, nikoli dvěma, tedy prijmeni = ‘Novák’, nikoli prijmeni == ‘Novák’

Pozor na prioritu podmínek: NOT má přednost před AND, AND má přednost před OR. Pokud chci např. všechny učitele s příjmením Novák i studenty s příjmením Novák, tento dotaz je ŠPATNĚ (vybere všechny studenty s příjmením Novák a k nim i všechny učitele s libovolným příjmením):

```
SELECT jmeno, prijmeni, role
FROM kontakty
WHERE prijmeni = 'Novák' AND role = 'student' OR role = 'učitel'
```

je totiž shodný s dotazem:

```
SELECT jmeno, prijmeni, role
FROM kontakty
WHERE (prijmeni = 'Novák' AND role = 'student') OR (role = 'učitel')
```

správný dotaz by byl:

```
SELECT jmeno, prijmeni, role
FROM kontakty
WHERE prijmeni = 'Novák' AND (role = 'student' OR role = 'učitel')
```

IS NULL, IS NOT NULL

Hodnota null znamená „neznámá hodnota". Pokud je v podmínce WHERE použito porovnání s null hodnotou, je výsledek nedefinován (undefined) a podmínka není splněna.

Příklad

Uvažujme tabulku studenti:

jmeno	*prijmeni*	*datum_slozeni_zkousky*
Jan	Novák	2024-06-03
Jiří	Novotný	null
František	Rovný	null

Oba následující dotazy NEVRÁTÍ ani jeden záznam.

```
SELECT jmeno, prijmeni,
FROM studenti
WHERE datum_slozeni_zkousky = null
```

```
SELECT jmeno, prijmeni,
FROM studenti
WHERE datum_slozeni_zkousky <> null
```

Porovnání na null hodnotu se správně provádí **IS NULL** a **IS NOT NULL**.

Příklad

```
SELECT jmeno, prijmeni,
FROM studenti
WHERE datum_slozeni_zkousky IS NULL
```

```sql
SELECT jmeno, prijmeni,
FROM studenti
WHERE datum_slozeni_zkousky IS NOT NULL
```

BETWEEN

Vybírá záznamy v daném rozmezí, včetně hraničních hodnot.

Příklad

```sql
SELECT nazev, cena
FROM zbozi
WHERE cena BETWEEN 100 AND 180
```

Vybere všechno zboží s cenou od 100 včetně do 180 včetně. Následující dotaz vrátí stejné výsledky:

```sql
SELECT nazev, cena
FROM ZBOZI
WHERE cena >= 100 AND cena <= 180
```

IN

Vybírá záznamy, které jsou shodné s uvedenými hodnotami.

Příklad

```sql
SELECT vyrobce, model, cena
FROM mobilni_telefony
WHERE vyrobce IN ('Apple', 'Google', 'Samsung')
```

Následující dotaz vrátí stejné výsledky jako předchozí:

```sql
SELECT vyrobce, model, cena
FROM mobilni_telefony
WHERE vyrobce = 'Apple' OR vyrobce = 'Google' OR vyrobce = 'Samsung'
```

Použití vnořeného SELECT V podmínce IN může být jako seznam použit i vnořený SELECT, který vrací právě jeden sloupec s hodnotami, které se dají porovnat s hodnotami sloupce, podle kterého se řídí IN podmínka.

Příklad

```sql
SELECT nazev, cena
FROM zbozi
WHERE id_vyrobce IN (SELECT id FROM vyrobce WHERE stat = 'ČR')
```

LIKE

Vybírá záznamy, které odpovídají masce (šabloně). Používá se pouze pro řetězce. V masce se dají použít zástupné znaky:

% (procento)	nahrazuje libovolný počet znaků (i nula znaků)
_ (podtržítko)	nahrazuje právě jeden znak

Příklad

Vybere obce s názvem 'Horní Lhota', 'Dolní Lhota', 'Lhota', ale NEVYBERE 'Lhota u Brodu'.

```
SELECT nazev, kraj, pocet_obyvatel
FROM obce
WHERE nazev LIKE '%Lhota'
```

Příklad

Vybere obce s názvem 'Horní Lhota', 'Dolní Lhota', 'Lhota' a VYBERE i 'Lhota u Brodu'.

```
SELECT nazev, kraj, pocet_obyvatel
FROM obce
WHERE nazev LIKE '%Lhota%'
```

Příklad

Vybere záznamy s kódem např. 'C135', 'CF35', ale nevybere 'CE45' ani 'C1358'.

```
SELECT nazev, kod
FROM vyrobky
WHERE kod LIKE 'C_35'
```

Masky se dají kombinovat.

Příklad

Vybere záznamy s kódem např. 'AQ12345CC359876', 'AQC135', ale nevybere 'BAQC35', 'AQC1235'.

```
SELECT nazev, kod
FROM vyrobky
WHERE kod LIKE 'AQ%C_35%'
```

ANY

Splňuje podmínku, pokud je vybrán alespoň jeden záznam vnořeného SELECT.

Příklad

Vybere všechny makléře, kteří uzavřeli alespoň jednu objednávku za milion nebo více.

```
SELECT nazev
FROM makleri
WHERE id = ANY (SELECT id_makler FROM objednavky WHERE cena_celkem >= 1000000)
```

ALL

Splňuje podmínku, pokud všechny záznamy vnořeného SELECT splňují podmínku.

Příklad

Vybere všechny mobily, které jsou levnější než všechny mobily od Apple.

```
SELECT vyrobce, model, cena
FROM mobily
WHERE cena < ALL (SELECT cena FROM mobily where vyrobce = 'Apple')
```

GROUP BY

Podle jakých sloupců se budou provádět výpočty. Výpočty se nazývají **agregační funkce**, protože seskupují (aggregate) záznamy.

Pomůcka: GROUP BY je podle těch sloupců, podle kterých se NEPOČÍTÁ.

Příklad

```
SELECT id_ridic, sum(trestne_body) as soucet_bodu, sum(pokuta) as pokuta_celkem, count(\*) a
FROM prestupky
GROUP BY id_ridic
```

Agregační funkce

MIN	minimum
MAX	maximum
AVG	průměr
SUM	součet
COUNT	počet

Příklad operace (výpočet průměrné známky, pokud známky nemají váhy):

```
SELECT id_student, trida, AVG(znamka) as prumer
FROM znamky
GROUP BY id_student, trida
```

Ve výpočtech lze používat i matematické operace (výpočet průměrné známky, pokud známky mají váhy):

```
SELECT id_student, trida, SUM(znamka \* vaha) / SUM(vaha) as prumer
FROM znamky
GROUP BY id_student, trida
```

HAVING

Podmínka, která se uplatní PO agregačních funkcích.

Příklad (vyber jedničkáře):

```
SELECT id_student, trida, sum(znamka \* vaha) / sum(vaha) as prumer
FROM znamky
GROUP BY id_student, trida
HAVING sum(znamka \* vaha) / sum(vaha) < 1.5
```

Místo výrazu pro výpočet lze použít název sloupce:

```
SELECT id_student, trida, sum(znamka \* vaha) / sum(vaha) as prumer
FROM znamky
GROUP BY id_student, trida
HAVING prumer < 1.5
```

Podmínky WHERE a HAVING lze kombinovat (jedničkáři z 3.D):

```
SELECT id_student, trida, sum(znamka \* vaha) / sum(vaha) as prumer
FROM znamky
WHERE id_trida = '3D'
GROUP BY id_student, id_trida
HAVING prumer < 1.5
```

ORDER BY

Podle jakých sloupců bude výsledný dotaz seřazen.

Příklad

```
SELECT id_student, trida, avg(znamka) as prumer
FROM znamky
GROUP BY id_student, trida
ORDER BY trida, prumer
```

Pro řazení od největšího k nejmenšímu se uvede klauzule DESC (seřaď po třídách od nejmenší k největší, v rámci třídy seřaď podle známky od nejhorší k nejlepší):

```
SELECT id_student, trida, avg(znamka) as prumer
FROM znamky
GROUP BY id_student, trida
ORDER BY trida, prumer DESC
```

TOP

Patří k části SELECT (viz výše). Téměř vždy se používá v kombinaci s ORDER BY

Příklad

```
SELECT TOP 5 vyrobce, model, maximalni_rychlost
FROM auta
ORDER BY maximalni_rychlost
```

UNION

Příkazem UNION je možné spojit výsledky dvou a více příkazů SELECT dohromady.

Příklad

```
SELECT datum, -castka as castka
FROM faktury
UNION
SELECT datum, castka
FROM platby
```

Použití příkazu má několik omezení:

- jednotlivé příkazy SELECT musí vracet stejný počet sloupců,
- odpovídající sloupce musí mít odpovídající (kompatibilní) datový typ.

Lze tedy pomocí UNION spojit dva SELECTy, pokud např. prvý v prvém sloupci vrací datový typ numeric (číslo), a druhý v prvém sloupci vrací money (peníze). Pokud by vraceli např. date (datum) a numeric (číslo), spojení není možné a dotaz skončí chybou.

Poznámky:

- část ORDER BY je možné uvést pouze jednou **na konci** příkazu, v jednotlivých SELECT se ORDER BY nemůže uvést (řadíme až výsledek po sloučení),
- UNION automaticky odstraní ty řádky, které jsou společné pro jednotlivé SELECTy,
- duplicitní řádky se zachovají při použití UNION ALL.

Příklad

Dotaz

```
SELECT 1
UNION
SELECT 1
```

vrátí záznam

1

Dotaz

```
SELECT 1
UNION ALL
SELECT 1
```

vrátí záznamy

1
1

SELECT z více tabulek

Nyní popíšeme hlavně ty části příkazu SELECT, které se týkají výběru z více tabulek. Před samotným výkladem je potřeba pochopit pojem klíč - hlavně primární klíč a cizí klíč.

Databáze

V následujících ukázkách budeme pracovat se vzorovou databází, skripty pro její vytvoření jsou k dispozici v části Vzorová databáze.

Tabulka vyucujici

obsahuje data:

id	jmeno	prijmeni
1	Jan	Rychtařík
2	Martin	Svázaný
3	Tomáš	Plácal
4	Milan	Chrastil

Sloupec *id* je primární klíč.

Tabulka ucebny

obsahuje data:

id	oznaceni	patro	id_vyucujici_zodpovedny
1	ICT1	2	1
2	ICT2	2	1
3	ICT3	2	4
4	ICT4	2	NULL
5	ICT5	2	2

Sloupec *id* je primární klíč. Sloupec *id_vyucujici_zodpovedny* je cizí klíč do tabulky *Vyucujici*.

V prvém řádku je zachyceno, že zodpovědným vyučujícím za učebnu ICT1 je Jan Rychtařík (*id_vyucujici_zodpovedny* má hodnotu 1, v tabulce *Vyucujici* má primární klíč hodnotu 1 v řádku Jan Rychtařík).

Ve třetím řádku je zachyceno, že zodpovědným vyučujícím za učebnu ICT3 je Milan Chrastil (*id_vyucujici_zodpovedny* má hodnotu 4, v tabulce *Vyucujici* má primární klíč hodnotu 4 v řádku Milan Chrastil).

Ve čtvrtém řádku je zachyceno, že učebna ICT4 nemá přiděleného zodpovědného vyučujícího.

Krátce řečeno: Za učebnyICT1 a ICT2 zodpovídá Jan Rychtařík, za učebnu ICT3 zodpovídá Milan Chrastil, učebna ICT4 nemá zodpovědného vyučujícího a za učebnu ICT5 zodpovídá Martin Svázaný. Tomáš Plácal nezodpovídá za žádnou učebnu.

Tabulka vybaveni_uceben

obsahuje data:

id	id_ucebny	nazev	evidencni_kod
1	1	'projektor'	'p42'
2	1	'tabule'	't58'
3	2	'televizor'	'tv15'
4	2	'televizor'	'tv17'
5	3	'projektor'	'p12'
6	3	'tabule'	't57'
7	4	'projektor'	'p38'
8	4	'tabule'	't51'
9	5	'projektor'	'p89'
10	5	'tabule'	't46'

Sloupec *id* je primární klíč. Sloupec *id_ucebny* je cizí klíč do tabulky *Ucebny*.

V prvém řádku je zachyceno, že učebna ICT1 je vybavena projektorem s evidenčním kódem 'p42'.

Ve čtvrtém řádku je zachyceno, že učebna ICT2 je vybavena televizorem s evidenčním kódem 'tv17'.

Tabulka predmety

obsahuje data:

id	nazev	zkratka
1	Programování	PRG
2	Vývoj aplikací	VAP
3	Operační systémy	OPS
4	Aplikační software	ASW

Sloupec *id* je primární klíč.

Tabulka vyucujici_predmety

obsahuje data:

id	id_vyucujici	id_predmety
1	1	4
2	2	1
3	2	2
4	3	1
5	3	2
6	4	3

Sloupec *id* je primární klíč. Sloupec *id_vyucujici* je cizí klíč do tabulky *Vyucujici*, slopec *id_predmety* je cizí klíč do tabulky *Predmety*.

V tabulce *vyucujici_predmety* je zachyceno, že:

- prvý záznam: vyučující číslo 1 (Jan Rychtařík) učí předmět číslo 4 (Aplikační software)
- druhý záznam: vyučující číslo 2 (Martin Svázaný) učí předmět číslo 1 (Programování)
- třetí záznam: vyučující číslo 2 (Martin Svázaný) učí předmět číslo 2 (Vývoj aplikací)
- atd.

Tabulka rozvrh

obsahuje data:

id	id_vyucujici	id_predmety	id_ucebny	den	hodina
1	1	4	5	čt	1
2	2	2	5	út	3
3	2	1	1	út	4
4	2	2	5	st	2
5	2	2	5	st	5

id	id_vyucujici	id_predmety	id_ucebny	den	hodina
6	2	2	5	st	7
7	2	2	5	st	8
8	2	2	5	st	9
9	3	2	5	út	5
10	3	2	5	út	6b
11	3	2	5	st	1
12	3	1	3	st	2
13	3	1	3	st	3
14	3	1	2	st	4
15	3	2	5	st	6a
16	4	3	3	út	4
17	4	3	3	út	5
18	4	3	3	st	4
19	4	3	3	st	5

Sloupec *id* je primární klíč. Sloupec *id_vyucujici* je cizí klíč do tabulky *Vyucujici*, slopec *id_predmety* je cizí klíč do tabulky *Predmety*, sloupec id_ucebny je cizí klíč do tabulky *Ucebny*.

V tabulce **rozvrh** je zachyceno, že:

- prvý záznam: vyučující číslo 1 (Jan Rychtařík) vyučuje předmět číslo 4 (Aplikační software) v učebně číslo 5 (ICT5) ve čtvrtek 1. hodinu.
- druhý záznam: vyučující číslo 2 (Martin Svázaný) vyučuje předmět číslo 2 (Vývoj aplikací) v učebně číslo 5 (ICT5) v úterý 3. hodinu.
- atd.

SELECT, FROM

V části FROM můžeme uvést libovolný počet tabulek. Výsledkem je kombinace všech řádků všech tabulek (kartézký součin).

Příklad

Výsledkem příkazu

```
SELECT vyucujici.prijmeni, ucebny.oznaceni
FROM vyucujici, ucebny
```

je tabulka s celkem 25 řádky (4 řádky tabulky *vyucujici* krát 5 řádků tabulky *ucebny*):

prijmeni	oznaceni
Rychtařík	ICT1
Rychtařík	ICT2

prijmeni	oznaceni
Rychtařík	ICT3
Rychtařík	ICT4
Rychtařík	ICT5
Svázaný	ICT1
Svázaný	ICT2
Svázaný	ICT3
Svázaný	ICT4
Svázaný	ICT5
Plácal	ICT1
Plácal	ICT2
Plácal	ICT3
Plácal	ICT4
Plácal	ICT5
Chrastil	ICT1
Chrastil	ICT2
Chrastil	ICT3
Chrastil	ICT4
Chrastil	ICT5

Můžeme si představit, že v části FROM při uvedení více tabulek vznikne "obří" tabulka, která obsahuje všechny možné kombinace řádků všech tabulek. Tato obří tabulka je pak zpracována dalšími částmi příkazu tak, jak to již známe z části SELECT - jedna tabulka: část WHERE vybere jen řádky "obří" tabulky splňující podmínku, část GROUP BY záznamy sdruží, část HAVING vybere sdružené záznamy splňující podmínky, část ORDER BY výsledek seřadí a část SELECT do výsledku vybere jen uvedené sloupce.

Jinými slovy:

> Výběr z více tabulek je stejný jako výběr z jedné "obří" tabulky vzniklé po spojení těchto tabulek.

Pokud bychom chtěli předchozí dotaz upravit tak, abychom dostali informaci o tom, který vyučující je zodpovědný za jakou učebnu, doplníme část WHERE.

Příklad

```
SELECT vyucujici.prijmeni, ucebny.oznaceni
FROM vyucujici, ucebny
WHERE vyucujici.id = ucebny.id_vyucujici_zodpovedny
```

Výsledkem je tabulka

prijmeni	oznaceni
Rychtařík	ICT1

prijmeni	oznaceni
Rychtařík	ICT2
Chrastil	ICT3
Svázaný	ICT5

Všimněte si: ve WHERE části dotazu je podmínka, kdy požadujeme, aby se primární klíč tabulky *vyucujici* rovnal cizímu klíči *id_vyucujici_zodpovedny* tabulky *ucebny* do tabulky *vyucujici*. To je jedna z hlavních myšlenek relačních databází.

INNER JOIN

INNER JOIN slouží ke spojení dvou tabulek stejným způsobem, jako bychom uvedli jména tabulek v části FROM a spojovací podmínku v části WHERE.

Příklad

```
SELECT vyucujici.prijmeni, ucebny.oznaceni
FROM vyucujici
    INNER JOIN ucebny ON vyucujici.id = ucebny.id_vyucujici_zodpovedny
```

vrátí stejný výsledek jako

```
SELECT vyucujici.prijmeni, ucebny.oznaceni
FROM vyucujici, ucebny
WHERE vyucujici.id = ucebny.id_vyucujici_zodpovedny
```

INNER JOIN je možné řetězit, to znamená použít víc příkazů INNER JOIN za sebou.

Příklad

```
SELECT vyucujici.prijmeni, ucebny.oznaceni, vybaveni_uceben.nazev, vybaveni_uceben.evidencni
FROM vyucujici
    INNER JOIN ucebny ON vyucujici.id = ucebny.id_vyucujici_zodpovedny
    INNER JOIN vybaveni_uceben ON vybaveni_uceben.id_ucebny = ucebny.id
```

Nejdříve se provede spojení tabulek *vyucujici* a *ucebny*. Můžeme si představit, že výsledkem je "velká" tabulka, která obsahuje všechny sloupce z tabulky *vyucujici* i tabulky *ucebny*. Tato tabulka se v dalším kroku spojí s tabulkou *vybaveni_uceben* a vznikne "ještě větší" tabulka, která obsahuje všechny slouce z tabulek *vyucujici*, *ucebny* a *vybaveni_uceben*. Nakonec se z této "ještě větší" tabulky vyberou sloupce *prijmeni*, *oznaceni*, *nazev* a *evidencni kod*.

Výsledkem posledního dotazu tedy budou záznamy:

prijmeni	oznaceni	nazev	evidencni_kod
'Rychtařík'	'ICT1'	'projektor'	'p42'

prijmeni	oznaceni	nazev	evidencni_kod
‘Rychtařík’	‘ICT1’	‘tabule’	‘t58’
‘Rychtařík’	‘ICT2’	‘televizor’	‘tv15’
‘Rychtařík’	‘ICT2’	‘televizor’	‘tv17’
‘Chrastil’	‘ICT3’	‘projektor’	‘p12’
‘Chrastil’	‘ICT3’	‘tabule’	‘t57’
‘Svázaný’	‘ICT5’	‘projektor’	‘p89’
‘Svázaný’	‘ICT5’	‘tabule’	‘t46’

Dotaz vrátí seznam vybavení za které jsou jednotliví učitelé zodpovědní včetně informace o tom, v jaké učebně se vybavení nachází.

Předchozí dotaz odpovídá dotazu bez použití INNER JOIN:

```
SELECT vyucujici.prijmeni, ucebny.oznaceni,
    vybaveni_uceben.nazev, vybaveni_uceben.evidencni_kod
FROM vyucujici, ucebny, vybaveni_uceben
WHERE
    vyucujici.id = ucebny.id_vyucujici_zodpovedny
    AND vybaveni_uceben.id_ucebny = ucebny.id
```

Proč tedy vůbec používat INNER JOIN? Je to hlavně kvůli čitelnosti dotazů. Při použití INNER JOIN je lépe vidět, jak se postupně tabulky spojují.

Lepší čitelnost vynikne při použití další podmínky WHERE.

Příklad

```
SELECT vyucujici.prijmeni, ucebny.oznaceni,
    vybaveni_uceben.nazev, vybaveni_uceben.evidencni_kod
FROM vyucujici
    INNER JOIN ucebny ON vyucujici.id = ucebny.id_vyucujici_zodpovedny
    INNER JOIN vybaveni_uceben ON vybaveni_uceben.id = ucebny.id
WHERE vyucujici.prijmeni = N'Rychtařík'
```

Všimněte si, že podmínky, na základě kterých se tabulky spojují, jsou v části INNER JOIN, podmínka na další omezení záznamů je v části WHERE.

Poznámka: V části WHERE je při porovnání uveden řetězen N’Rychtařík’. Tento zápis znamená, že řetězev je typu NVARCHAR, tedy v kódování Unicode. Pokud by byl řetězec uveden jako ‘Rychtařík’, jednalo by se o datový typ VARCHAR, tedy ANSI kódování a při konverzi by mohlo podle nastavení databáze dojít ke ztrátě znaků.

Bez použití INNER JOIN by dotaz vypadal:

```
SELECT vyucujici.prijmeni, ucebny.oznaceni,
    vybaveni_uceben.nazev, vybaveni_uceben.evidencni_kod
FROM vyucujici, ucebny, vybaveni_uceben
```

```
WHERE
    vyucujici.id = ucebny.id_vyucujici_zodpovedny
    AND vybaveni_uceben.id = ucebny.id
    AND vyucujici.prijmeni = N'Rychtařík'
```

Zde se již přehlednost ztrácí.

LEFT JOIN

V některých případech je potřeba do JOIN zahrnout i ty záznamy, které by jinak "vypadly", protože neexistují odpovídající záznamy v obou tabulkách.

Příklad:

```
SELECT vyucujici.prijmeni, ucebny.oznaceni
FROM vyucujici, ucebny
WHERE vyucujici.id = ucebny.id_vyucujici_zodpovedny
```

Výsledkem je tabulka všech vyučujících a učeben, za které jsou zodpovědní. Pokud však některý z vyučujících nezodpovídá za žádnou učebnu, není ve výsledném dotazu uveden. V tomto případě chybí ve výsledku vyučující Plácal.

prijmeni	oznaceni
Rychtařík	ICT1
Rychtařík	ICT2
Chrastil	ICT3
Svázaný	ICT5

Řešením je použití LEFT JOIN, které spojuje tabulky stejným způsobem jako INNER JOIN a navíc přidá ze záznamů z tabulky stojící vlevo od LEFT JOIN ty, které nemají žádný odpovídající záznam v tabulce stojící vpravo od LEFT JOIN.

Pokud takové záznamy z tabulky vlevo existují a ve výsledném dotazu mají být uvedeny i sloupce z tabulky vpravo, jejich hodnota je NULL.

Příklad:

```
SELECT vyucujici.prijmeni, ucebny.oznaceni
FROM vyucujici
    LEFT JOIN ucebny ON vyucujici.id = ucebny.id_vyucujici_zodpovedny
```

vrátí výsledek

prijmeni	oznaceni
Rychtařík	ICT1
Rychtařík	ICT2
Svázaný	ICT5

prijmeni	oznaceni
Plácal	NULL
Chrastil	ICT3

Vyučující Plácal je zde uveden s tím, že hodnota sloupce *oznaceni* je NULL.

RIGHT JOIN

Jak již název napovídá, RIGHT JOIN je obdobný LEFT JOIN jen s tím rozdílem, že do výsledného výběru jsou zahrnuty i všechny řádky tabulky vpravo od RIGHT JOIN, které nemají žádné odpovídající záznamy v tabulce vlevo od RIGHT JOIN.

Příklad

```
SELECT vyucujici.prijmeni, ucebny.oznaceni
FROM vyucujici
    RIGHT JOIN ucebny ON vyucujici.id = ucebny.id_vyucujici_zodpovedny
```

vrátí výsledek

prijmeni	oznaceni
Rychtařík	ICT1
Rychtařík	ICT2
Chrastil	ICT3
NULL	ICT4
Svázaný	ICT5

Učebna ICT4 je zde uvedena s tím, že hodnota *prijmeni* je NULL.

FULL JOIN

FULL JOIN spojí obě tabulky tak, že ve výsledku jsou uvedeny i ty záznamy z obou tabulek, které nemají odpovídající záznamy v druhé z tabulek.

Příklad

```
SELECT vyucujici.prijmeni, ucebny.oznaceni
FROM vyucujici
    FULL JOIN ucebny ON vyucujici.id = ucebny.id_vyucujici_zodpovedny
```

vrátí výsledek

prijmeni	oznaceni
Rychtařík	ICT1

prijmeni	oznaceni
Rychtařík	ICT2
Svázaný	ICT5
Plácal	NULL
Chrastil	ICT3
NULL	ICT4

Ve výsledném dotazu můžeme vidět jak informaci o tom, že vyučující Plácal nezodpovídá za žádnou učebnu, tak informaci o tom, že učebna ICT4 nemá přiděleného zodpovědného vyučujícího.

Vzorová databáze

Skripty pro vzorovou databázi uvádíme ve dvou verzích:

- pro MSSQL server s Unicode znaky,
- podle SQL normy, v MSSQL server se použije ANSI kódování znaků.

Skripty pro MSSQL

```
SET ANSI_NULLS ON;

SET QUOTED_IDENTIFIER ON;

-- Tabulka vyucujici
CREATE TABLE vyucujici(
    id int NOT NULL primary key,
    jmeno nvarchar(100) NOT NULL,
    prijmeni nvarchar(100) NOT NULL);

INSERT vyucujici (id, jmeno, prijmeni)
VALUES
(1, N'Jan', N'Rychtařík'),
(2, N'Martin', N'Svázaný'),
(3, N'Tomáš ', N'Plácal'),
(4, N'Milan', N'Chrastil');

-- Tabulka ucebny
CREATE TABLE ucebny(
    id int NOT NULL primary key,
    oznaceni nvarchar(5) NOT NULL,
    patro numeric(18, 0) NOT NULL,
    id_vyucujici_zodpovedny int NULL);
```

```sql
INSERT ucebny
(id, oznaceni, patro, id_vyucujici_zodpovedny)
VALUES
(1, N'ICT1', 2, 1),
(2, N'ICT2', 2, 1),
(3, N'ICT3', 2, 4),
(4, N'ICT4', 2, NULL),
(5, N'ICT5', 2, 2);

-- Tabulka vybaveni_uceben
CREATE TABLE vybaveni_uceben(
    id int NOT NULL primary key,
    id_ucebny int NOT NULL,
    nazev nvarchar(100) NOT NULL,
    evidencni_kod nvarchar(10) NOT NULL);

-- Tabulka vybaveni_uceben
INSERT vybaveni_uceben
(id, id_ucebny , nazev, evidencni_kod )
VALUES
(1, 1, N'projektor', N'p42'),
(2, 1, N'tabule', N't58'),
(3, 2, N'televizor', N'tv15'),
(4, 2, N'televizor', N'tv17'),
(5, 3, N'projektor', N'p12'),
(6, 3, N'tabule', N't57'),
(7, 4, N'projektor', N'p38'),
(8, 4, N'tabule', N't51'),
(9, 5, N'projektor', N'p89'),
(10, 5, N'tabule', N't46');

-- Tabulka predmety
CREATE TABLE predmety(
    id int NOT NULL primary key,
    nazev nvarchar(100) NOT NULL,
    zkratka nchar(3) NOT NULL);

INSERT predmety
(id, nazev, zkratka)
VALUES
(1, N'Programování', N'PRG'),
(2, N'Vývoj aplikací', N'VAP'),
(3, N'Operační systémy', N'OPS'),
(4, N'Aplikační software', N'ASW');

-- Tabulka vyucujici_predmety
```

```
CREATE TABLE vyucujici_predmety(
    id int NOT NULL primary key,
    id_predmety int NOT NULL,
    id_vyucujici int NOT NULL);

INSERT vyucujici_predmety
(id, id_predmety, id_vyucujici)
VALUES
(1, 1, 4),
(2, 2, 1),
(3, 2, 2),
(4, 3, 1),
(5, 3, 2),
(6, 4, 3);

-- Tabulka rozvrh
CREATE TABLE rozvrh(
    id int NOT NULL primary key,
    id_vyucujici int NOT NULL,
    id_predmety int NOT NULL,
    id_ucebny int NOT NULL,
    den nchar(2) NULL,
    hodina nvarchar(2) NULL);

INSERT rozvrh (id, id_vyucujici, id_predmety, id_ucebny, den, hodina)
VALUES
(1, 1, 4, 5, N'čt', N'1'),
(2, 2, 2, 5, N'út', N'3'),
(3, 2, 2, 5, N'út', N'4'),
(4, 2, 2, 5, N'st', N'2'),
(5, 2, 2, 5, N'st', N'5'),
(6, 2, 2, 5, N'st', N'7'),
(7, 2, 2, 5, N'st', N'8'),
(8, 2, 2, 5, N'st', N'9'),
(9, 3, 2, 5, N'út', N'5'),
(10, 3, 2, 5, N'út', N'6b'),
(11, 3, 2, 5, N'st', N'1'),
(12, 3, 1, 3, N'st', N'2'),
(13, 3, 1, 3, N'st', N'3'),
(14, 3, 1, 2, N'st', N'4'),
(15, 3, 2, 5, N'st', N'6a'),
(16, 4, 3, 3, N'út', N'4'),
(17, 4, 3, 3, N'út', N'5'),
(18, 4, 3, 3, N'st', N'4'),
(19, 4, 3, 3, N'st', N'5');
```

Skripty podle SQL normy pro většinu systémů

```sql
-- Tabulka vyucujici
CREATE TABLE vyucujici(
    id int NOT NULL primary key,
    jmeno varchar(100) NOT NULL,
    prijmeni varchar(100) NOT NULL);

INSERT INTO vyucujici (id, jmeno, prijmeni)
VALUES
(1, 'Jan', 'Rychtařík'),
(2, 'Martin', 'Svázaný'),
(3, 'Tomáš ', 'Plácal'),
(4, 'Milan', 'Chrastil');

-- Tabulka ucebny
CREATE TABLE ucebny(
    id int NOT NULL primary key,
    oznaceni varchar(5) NOT NULL,
    patro numeric(18, 0) NOT NULL,
    id_vyucujici_zodpovedny int NULL);

INSERT INTO ucebny
(id, oznaceni, patro, id_vyucujici_zodpovedny)
VALUES
(1, 'ICT1', 2, 1),
(2, 'ICT2', 2, 1),
(3, 'ICT3', 2, 4),
(4, 'ICT4', 2, NULL),
(5, 'ICT5', 2, 2);

-- Tabulka vybaveni_uceben
CREATE TABLE vybaveni_uceben(
    id int NOT NULL primary key,
    id_ucebny int NOT NULL,
    nazev varchar(100) NOT NULL,
    evidencni_kod varchar(10) NOT NULL);

-- Tabulka vybaveni_uceben
INSERT INTO vybaveni_uceben
(id, id_ucebny , nazev, evidencni_kod )
VALUES
(1, 1, 'projektor', 'p42'),
(2, 1, 'tabule', 't58'),
(3, 2, 'televizor', 'tv15'),
(4, 2, 'televizor', 'tv17'),
```

```
(5, 3, 'projektor', 'p12'),
(6, 3, 'tabule', 't57'),
(7, 4, 'projektor', 'p38'),
(8, 4, 'tabule', 't51'),
(9, 5, 'projektor', 'p89'),
(10, 5, 'tabule', 't46');

-- Tabulka predmety
CREATE TABLE predmety(
    id int NOT NULL primary key,
    nazev varchar(100) NOT NULL,
    zkratka nchar(3) NOT NULL);

INSERT INTO predmety
(id, nazev, zkratka)
VALUES
(1, 'Programování', 'PRG'),
(2, 'Vývoj aplikací', 'VAP'),
(3, 'Operační systémy', 'OPS'),
(4, 'Aplikační software', 'ASW');

-- Tabulka vyucujici_predmety
CREATE TABLE vyucujici_predmety(
    id int NOT NULL primary key,
    id_predmety int NOT NULL,
    id_vyucujici int NOT NULL);

INSERT INTO vyucujici_predmety
(id, id_predmety, id_vyucujici)
VALUES
(1, 1, 4),
(2, 2, 1),
(3, 2, 2),
(4, 3, 1),
(5, 3, 2),
(6, 4, 3);

-- Tabulka rozvrh
CREATE TABLE rozvrh(
    id int NOT NULL primary key,
    id_vyucujici int NOT NULL,
    id_predmety int NOT NULL,
    id_ucebny int NOT NULL,
    den nchar(2) NULL,
    hodina varchar(2) NULL);
```

```sql
INSERT INTO rozvrh (id, id_vyucujici, id_predmety, id_ucebny, den, hodina)
VALUES
(1, 1, 4, 5, 'čt', '1'),
(2, 2, 2, 5, 'út', '3'),
(3, 2, 2, 5, 'út', '4'),
(4, 2, 2, 5, 'st', '2'),
(5, 2, 2, 5, 'st', '5'),
(6, 2, 2, 5, 'st', '7'),
(7, 2, 2, 5, 'st', '8'),
(8, 2, 2, 5, 'st', '9'),
(9, 3, 2, 5, 'út', '5'),
(10, 3, 2, 5, 'út', '6b'),
(11, 3, 2, 5, 'st', '1'),
(12, 3, 1, 3, 'st', '2'),
(13, 3, 1, 3, 'st', '3'),
(14, 3, 1, 2, 'st', '4'),
(15, 3, 2, 5, 'st', '6a'),
(16, 4, 3, 3, 'út', '4'),
(17, 4, 3, 3, 'út', '5'),
(18, 4, 3, 3, 'st', '4'),
(19, 4, 3, 3, 'st', '5');
```