

SQL (Structured Query Language) je nejrozšířenější dotazovací databázový jazyk. Jeho znalost zajišťuje uživateli poměrně jednoduchý přístup k datům, je ale určen programátorům, kteří jej používají pro konstrukci přístupových cest k datům a nabízejí je tak uživatelům pro různé manipulace podle jejich způsobu oborového myšlení. Přestože je SQL již poměrně podrobně standardizován (viz [ISOSQL2003]), při práci s ním zvláště programátoři často používají jeho části, které ve standardu obsaženy nejsou. Následující přehled dotazů je pouze základním seznamem a jejich zde uváděná podoba striktně podléhá standardu. V dále uvedeném tvaru jsou zcela použitelné v libovolném databázovém stroji, který standard splňuje.

Každá databáze disponuje řádkově orientovaným přístupem. V oknově orientovaných operačních systémech se jedná o aplikaci, která v okně umožní řádkovou komunikaci s obsahem databáze. Řádkově orientovaná komunikace znamená, že uživatel na řádku píše některý z dotazů a teprve po stisknutí klávesy Enter je tento dotaz odeslán databázovému stroji. Databázový stroj dotaz zpracuje a výsledky vypíše na několika řádcích za text uživatelova dotazu. Uživatel píše dotaz na základě výzvy ke komunikaci, který nazýváme prompt, např.:

SQL>

Uživatel píše na řádek dotaz a odesílá jej klávesou Enter. Dotaz je formálně ale ukončen znakem ; (středník), který uživatel napíše před stiskem klávesy Enter. Uživatel ale může pokračovat v zápisu dotazu na několika řádcích a dokud nenapíše znak ; následovaný klávesou Enter, dotaz není databázovým stroje zpracován. Např.

```
SQL> SELECT * FROM PRACOVIŠTĚ;  
ID_PRACOVIŠTĚ  NÁZEV
```

```
-----  
1              kotelna  
2              výpočetní středisko  
3              ústředí
```

(tučně psaný text je odezvou databázového stroje).

V dalším textu uváděné formální definice používají obvyklý definiční metajazyk. Význam použitých symbolů je následující:

- velkými písmeny uvádíme text jazyka SQL, text malými písmeny uživatel substituuje jménem tabulky, sloupce, uvedením konkrétního vztahu atp.,
- část uvedená v hranatých závorkách [ a ] je nepovinná,

**SELECT Dotaz** SELECT má formální definici

```
SELECT sloupec,sloupec...
FROM tabulka,tabulka...
[WHERE podmínka ]
[GROUP BY výstup,výstup...]
[HAVING podmínka ]
[{UNION | UNION ALL | INTERSECT | MINUS} SELECT ...]
[ORDER BY kritérium_řídění]
```

Např.

```
SQL> SELECT NÁZEV FROM PRACOVIŠTĚ;
NÁZEV
```

```
-----
```

```
kotelna
výpočetní středisko
ústředí
```

```
SQL> SELECT NÁZEV FROM PRACOVIŠTĚ WHERE ID_PRACOVIŠTĚ > 2;
NÁZEV
```

```
-----
```

```
ústředí
```

byl proveden výpis názvů (položka `NÁZEV`) všech záznamů tabulky `PRACOVIŠTĚ` a všech záznamů tabulky `PRACOVIŠTĚ`, jejichž položka `ID_PRACOVIŠTĚ` (jednoznačná číselná identifikace záznamu) je větší než 2.

**INSERT Dotaz** INSERT vloží nový záznam do databáze, má formální definici

```
INSERT INTO tabulka
[(sloupec[ ,sloupec][ ,...])]
{ SELECT... | VALUES (hodnota[ ,hodnota][ ,hodnota]...) }
```

Např.

```
SQL> INSERT INTO PRACOVIŠTĚ (ID_PRACOVIŠTĚ,NÁZEV) VALUES (4,'obchodní
oddělení');
```

```
SQL>
```

je totéž jako

```
SQL> INSERT INTO PRACOVIŠTĚ VALUES (4,'obchodní oddělení');
```

a dotazem SELECT můžeme prověřit, zda se stalo, co jsme chtěli:

```
SQL> SELECT * FROM PRACOVIŠTĚ;
```

```
ID_PRACOVIŠTĚ  NÁZEV
```

```
-----
```

```
1              kotelna
```

**Dotaz DELETE** odstraní data z databáze, formální definice je

**DELETE**

```
DELETE FROM tabulka
[WHERE podmínka]
```

Např.

```
SQL> DELETE FROM PRACOVISTĚ WHERE ID_PRACOVISTĚ=4;
```

odstraní každý záznam, jehož ID\_PRACOVISTĚ je 4, ale bez uvedení podmínky

```
SQL> DELETE FROM PRACOVISTĚ;
```

odstraní všechny záznamy tabulky PRACOVISTĚ (!).

**Dotaz UPDATE** mění obsah položek záznamů, formální definice je

**UPDATE**

```
UPDATE tabulka
SET sloupec=hodnota [,sloupec=hodnota][,sloupec=hodnota]...
[WHERE podmínka]
```

Např.

```
SQL> UPDATE PRACOVISTĚ SET NÁZEV='ústředí organizace' WHERE ID_PRACOVISTĚ=3;
```

nebo

```
SQL> UPDATE PRACOVISTĚ SET NÁZEV='ústředí organizace' WHERE NÁZEV='ústředí';
```

Při práci s definováním datového modelu používáme dotazy pro vytváření tabulek, jejich odstraňování, doplňování atd. Novou tabulku vytvoříme pomocí dotazu CREATE, tabulku zrušíme pomocí DROP. Oba dotazy se ale používají také na další prvky struktury datového modelu, jakým je např. *index*, *uživatel*, *pohled* aj.

**Dotaz CREATE** vytvoří novou strukturu dat (např. tabulku), formální definice pro vytvoření tabulky je

**CREATE**

```
CREATE TABLE tabulka
(sloupec typ [omezení_sloupce] [,sloupec typ [omezení_sloupce]...]
omezení_tabulky)
```

Např.

```
SQL> CREATE TABLE PRACOVISTĚ
      (ID_PRACOVISTĚ DECIMAL(2,0) PRIMARY KEY, NÁZEV CHAR(20) );
```

Standardizované datové typy, které musíme uvádět při definici položek tabulky, jsou především číselné (obsahují číselné údaje), znakové (obecné texty) a časové (datum a čas).

Číselné datové typy jsou:

INTEGER            položka bude obsahovat celá čísla (kladná i záporná),  
                      pro položky, kdy víme, že se bude jednat o malá celá čísla,

**REAL** malé desetinné číslo, v parametru i zde počet platných číslic, nejvýše 18,  
**BIT** binární data, není číselný typ, jedná se o manipulaci s daty na bitové úrovni, může tedy obsahovat cokoli (viz Příloha A), v parametru celé číslo, udávající délku v bajtech, obvykle omezenou na 255.

Znakové datové typy:

**CHAR** textový řetězec určené max. délky (nejvýše 255), v parametru uvedena délka,  
**VARCHAR** text proměnlivé délky, v parametru se uvádí max. délka, např. `VARCHAR(2000)`.

Datové typy pro datum a čas:

**DATE** položka bude obsahovat údaje o dnu, měsíci a roku, formát není standardizován a často jej lze v konkrétní databázi měnit na jiný,  
**TIME** čas, údaje o hodině, minutě a vteřině,  
**TIMESTAMP** datum a čas.

Dotazem `CREATE` vytváříme i jiné struktury dat datového modelu, jako jsou `INDEX`, `SEQUENCE`, `USER`, `ROLE`, `VIEW`, `TRIGGER`.

**INDEX** `INDEX` Obsah tabulky lze indexovat, tj. záznamy mají přiřazeno své umístění v databázi a jsou takto rychleji dostupné. Kritéria pro umístění stanovujeme ve chvíli definice indexu, a to tak, že určíme, na které sloupce tabulky má být index zaměřen. Používáním indexů se výrazně zvýší čtení (především dotazem `SELECT`) dat z tabulky. Formální definice pro vznik indexu je

```
CREATE [UNIQUE] INDEX index
ON tabulka
(sloupec, sloupec ...)
```

Index se doporučuje znovu vytvářet vždy po nárůstu počtu záznamů v tabulce (zrušíme jej pomocí `DROP INDEX` a nově vytvoříme pomocí `CREATE INDEX`). Každá tabulka je indexována implicitně podle primárního klíče (viz Enjoy It III.3). Např.

```
SQL> CREATE INDEX IX_Prac_NÁZEV ON PRACOVIŠTĚ (NÁZEV);
```

umožní rychlejší čtení, pokud vyhledáváme podle názvu pracoviště.

**SEQUENCE** `SEQUENCE` definuje sekvenci. Sekvence udržuje naposledy použitou číselnou hodnotu používané číselné řady, např. pro jednoznačné označování postupně

Např.

```
SQL> CREATE SEQUENCE SQ_ID_PRACOVIŠTĚ START WITH 1 INCREMENT BY 1 NOCYCLE;
```

USER K databázi mohou přistupovat uživatelé různých oprávnění pro čtení, vytváření nebo aktualizaci dat různých částí datového modelu. Výchozí nastavení databáze pracuje s nejméně jedním uživatelem, který má neomezená práva přístupu k databázi. Pomocí vytváření různých uživatelů a pomocí dotazu GRANT lze uživatele omezovat v přístupu k tabulkám, jejich sloupcům, atd. Vytvořit nového uživatele lze příkazem

**USER, GRANT,  
REVOKE**

```
CREATE USER uživatel
```

jehož formát není stále striktně standardizován.

Např.

```
SQL> CREATE USER KOTELNÍK;
```

```
SQL> GRANT SELECT ON PRACOVIŠTĚ TO KOTELNÍK;
```

```
SQL> REVOKE DELETE ON PRACOVIŠTĚ FROM KOTELNÍK;
```

vzniká uživatel se jménem KOTELNÍK, který má přístup k tabulce PRACOVIŠTĚ dotazem SELECT, nemůže ale v tabulce odstraňovat záznamy. Bude-li KOTELNÍK zlobit, lze mu odejmout i právo pro SELECT pomocí REVOKE takto

```
SQL> REVOKE SELECT ON PRACOVIŠTĚ FROM KOTELNÍK;
```

ROLE Pro vypracování skupin různě silných uživatelů vzhledem k pravidlům přístupových metod k datovému modelu slouží role. Definujeme postupně různé role a pak je pomocí dotazu GRANT přiřazujeme vytvořeným uživatelům (nebo je pomocí REVOKE uživatelům odebíráme). Formát je

**ROLE**

```
CREATE ROLE role
```

Např.

```
SQL> CREATE ROLE VEDENÍ;
```

```
SQL> GRANT ALL ON PRACOVIŠTĚ TO VEDENÍ;
```

```
SQL> CREATE USER ŘEDITEL;
```

```
SQL> GRANT VEDENI TO ŘEDITEL;
```

VIEW je pohled. Pomocí něj můžeme usnadnit uživateli přístup k datům. Jedná se také o bezpečnostní opatření, protože přístup k původní tabulce nebo tabulkám, nad kterými je pohled vytvořen, uživateli odpojíme a přiřadíme mu pouze přístup k pohledu (i zde pomocí GRANT a REVOKE). Pohled definujeme a vzniká tak zástupné jméno pro virtuální tabulku. Definovat pohled znamená určit, které položky kterých tabulek budou sloučeny v rámci jednoho pohledu VIEW. Formát je

**VIEW**

```
CREATE VIEW pohled
```

a následně můžeme použít

```
SQL> SELECT PŘÍJMENÍ, NÁZEV FROM kdo_kde;
```

**TRIGGER** TRIGGER je vnitřní procedura v databázi, která je aktivována tehdy, nastane-li v dané tabulce vytvoření nového záznamu, změna záznamu nebo zrušení záznamu. Jedná se již o programování vnitřních procedur, které je věcí nikoliv obyčejných uživatelů. Navíc dosavadní definice vnitřních procedur a programování uvnitř databáze není standardizována a výrazně se liší napříč různými implementacemi SQL.

**ALTER** Dotaz ALTER mění dříve definovanou strukturu dat, upraví strukturu tabulky, databázového uživatele, sekvence atd. Formální definice pro změnu v tabulce je

```
ALTER TABLE tabulka
[ ADD (sloupec [, sloupec] [...]) ]
[ MODIFY (sloupec [, sloupec] [...]) ]
[ DROP { sloupec | CONSTRAINT omezení } ]
```

Např.

```
SQL> ALTER TABLE PRACOVIŠTĚ ADD (POČET_ŽIDLÍ DECIMAL(2,0));
```

Doplnění sloupce nezpůsobí žádnou ztrátu dat v již existujících záznamech. Obsah nových sloupců v již existujících záznamech není dle standardu definován, obvykle ale bývá databázovým strojem naplněn hodnotou NULL (prázdné hodnoty).

**DROP** Dotaz DROP odstraní dříve definovaná data, např. celou tabulku (zmizí i její obsah), formáty:

```
DROP TABLE tabulka
DROP INDEX index
DROP VIEW
DROP USER uživatel
DROP ROLE role
DROP SEQUENCE sekvence
```

Např.

```
SQL> DROP VIEW kdo_kde;
```

```
SQL> DROP TABLE PRACOVIŠTĚ;
```