MySQL

Typy tabulek – Storage Engines

- **MyISAM** – **defaultní**, neumí transakce, umí **fulltext**
- **InnoDB** – **transakce**, **cizí klíče**, neumí fulltext (a nebo už ano?)
- **MEMORY** (HEAP) – v paměti; neumí transakce
- **ARCHIVE** – velké množství dat bez indexů
- **CSV** – v textovém souboru ve formátu hodnot oddělených čárkou
- **FEDERATED** – v databázi na jiném serveru; místní server se připojí ke vzdálenému serveru a pomocí SQL čte nebo upravuje vzdálenou databázi
- **MERGE** (MRG_MYISAM) – neumí transakce; práce s několika MyISAM tabulkami stejné struktury jakoby to byla jedna; umožňuje z takto spojené tabulky číst ale pro zápis musí mít tabulka nastaveno do které skutečné tabulky zapisovat
- BDB (BerkeleyDB) – transakce; odstraněno ve verzi 5.1
- EXAMPLE – šablona pro vývojáře nových typů tabulek
- BLACKHOLE – černá díra; data přijímá, vrací vždy prázdný výsledek
- ISAM – původní; nahrazeno MyISAM

Tabulky **ne**podporující transakce (MyISAM) jsou **rychlejší**, zabírají **méně místa na disku** a potřebují **méně paměti** pro UPDATE.

MyISAM na data, z nichž se **často vybírá** pomocí SELECT a **InnoDB** na data, která se **často mění**.

```
CREATE TABLE t (i INT) ENGINE = INNODB;
ALTER TABLE t ENGINE = MYISAM;
```

Místo *ENGINE* se dříve používalo *TYPE* a tento termín je zachován z důvodu zpětné kompatibility.

Indexy

Zrychlení přístupu k datům.

Používat u všech sloupců, podle kterých se **vyhledává**, **třídí** nebo podle kterých se **spojují** tabulky.

Nepoužívat u tabulek, do kterých se **převážně vkládá** a jen výjimečně se z nich čte (např. logy).

- **Běžný** index.
- **Unikátní** index – nedovolí do tabulky vložit více záznamů se stejnou hodnotou sloupce, nad kterým je index definován.
- **Primární klíč** – označuje sloupec, který jednoznačně identifikuje libovolný záznam v tabulce; může být jen jeden v tabulce.

Transakce – Transactions

Provedení **několika činností dohromady** jako jedné.

- Nelze provést jen část – transakce se vždy musí provést celá nebo vůbec.
- Když je potřeba v půlce skončit, musí se vrátit všechny dosavadní změny.

```
START TRANSACTION
```

Zahájí neboli **odstartuje transakci**. Veškeré příkazy zadané později jsou součástí transakce a navenek se tedy budou jevit jako jediný příkaz.

```
COMMIT
```

Aktuální transakce je **potvrzena**. Změny jsou zapsány do databáze.

```
ROLLBACK
```

Aktuální transakce je **zamítnuta**. Všechny provedené změny jsou zrušeny a databáze se vrátí do stavu, v němž byla před zahájením transakce.

```
START TRANSACTION;
UPDATE platy SET plat=plat+2000 WHERE plat < 15000;
```

```
UPDATE platy SET plat=plat*1.1;
COMMIT;
```

Při použití InnoDB je transakce **každý samostatný příkaz**. Tzn. okolo transakce složené z jednoho příkazu není třeba psát START TRANSACTION a COMMIT.

```
SELECT * FROM cosi;
```

Předchozí příkaz provede ve skutečnosti toto:

```
START TRANSACTION; SELECT * FROM cosi; COMMIT;
```

Během provádění transakce všechny ostatní pokusy o práci nad stejnými daty **čekají.**

Cizí klíče – Foreign keys

V databázi máme následující tabulky:

- **osoby** se sloupci **osoba_id** a **jméno**
- **psi** se sloupci **pes_id**, **majitel** a **rasa**

Každý záznam psa má uvedené id majitele. Proto označíme v tabulce psi **sloupec majitel jako cizí klíč, vztažený k sloupci osoba_id** v tabulce osoby.

Když je poté přidán záznam pro psa, databázový engine bude **vyžadovat**, aby číslo v poli majitel nabývalo některé z existujících hodnot osoba_id tabulky osoby. Zároveň můžeme určit, zda se při smazání osoby **smažou** i záznamy všech vlastněných psů, nebo zda má pokus o smazání osoby vlastnící alespoň jednoho psa **selhat.**

Nastavení v phpMyAdmin pod odkazem **Zobrazit relace** na stránce struktury tabulky.

Tabulka musí být typu **InnoDB**.

```
ALTER TABLE psi
ADD FOREIGN KEY(majitel)
```

```
REFERENCES osoby(osoba_id)
ON UPDATE CASCADE     # změna id chovatele = změna i u všech jeho psů
                      # RESTRICT – zabránění změně id chovatele
ON DELETE RESTRICT;   # chovatel nemůže být smazán pokud má nějaké psy
                      # CASCADE – smazání chovatele = smazání i jeho psů
                      # SET NULL – nastaví majitele na NULL
```

Pohledy – Views

Statické dotazy, s nimiž lze pracovat, jako by to byly tabulky (**Pojmenované SELECTy**). V MySQL od verze 5.0.

```
CREATE VIEW vwPracovnici AS SELECT * FROM pracovnici;
SELECT * FROM vwPracovnici;
CREATE VIEW vwPrumernyVek AS SELECT AVG(vek) AS prumer FROM pracovnici;
CREATE VIEW vwDobNeurc AS SELECT * FROM pracovnici WHERE zam_do IS NULL;
CREATE VIEW vwLidi AS SELECT prijmeni FROM pracovnici ORDER BY prijmeni;
```

PhpMyAdmin zobrazí existující pohledy v seznamu tabulek.

Spojování tabulek

```
SELECT * FROM knihy, druhy;
```

Každý řádek s každým spojeny na jednom řádku – **kartézský součin**. Ke každé *knize* všechny *druhy*, které existují. My ale chceme *knihu* a její *druh* na jednom řádku. Vybereme tedy řádky kde *druh z* tabulky *knihy* se rovná *id* z tab. *druhy*.

```
SELECT * FROM knihy, druhy WHERE knihy.druh = druhy.id;
SELECT * FROM knihy INNER JOIN druhy ON knihy.druh = druhy.id [WHERE …
ORDER BY …];
```

K výpisu i *knih,* které nemají nastavený *druh* (vypíše **všechny** z levé tabulky – podle LEFT/RIGHT – a **k nim přiřadí** z té druhé podle výrazu):

```
… FROM knihy LEFT JOIN druhy ON knihy.druh = druhy.id;
… FROM druhy RIGHT JOIN knihy ON knihy.druh = druhy.id;
```

2× totéž.

Poddotazy – Subqueries

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
SELECT * FROM lidi WHERE mesto IN (SELECT mesto FROM mesta);
```

Uložené procedury – Stored procedures

Sada příkazů SQL, které jsou **uložené na serveru** a **zkompilované** pro rychlejší použití. V MySQL od verze 5.0.

```
CREATE PROCEDURE spVratRadky (od INT, do INT)
BEGIN
  SELECT * FROM software WHERE id BETWEEN od AND do;
END
CALL spVratRadky(10, 20)
```

Při opakovaném spuštění bude následující příkaz cca o 25-30% rychlejší než prostá sada INSERTů.

```
create procedure sp_vlozradek(id INT, nazev VARCHAR(50))
BEGIN
  INSERT INTO software (id, nazev) VALUES (id, nazev);
END
```

```
CREATE PROCEDURE sp_vlozneboaktualizuj (radek INT, novynazev
VARCHAR(50))
BEGIN
  IF EXISTS(SELECT * FROM software WHERE id = radek) THEN
    UPDATE software SET nazev = novynazev WHERE id = radek;
  ELSE
    INSERT INTO software (id, nazev) VALUES (radek, novynazev);
  END IF;
END
```

```
CREATE PROCEDURE spkalendar()
BEGIN
  DECLARE den DATE;
```

```
  SET den = CURDATE();
  CREATE TEMPORARY TABLE dny (datum DATE);
  WHILE den < (CURDATE() + INTERVAL 30 DAY) DO
    INSERT INTO dny (datum) VALUES (den);
    SET den = den + INTERVAL 1 DAY;
  END WHILE;
  SELECT datum FROM dny;
END
```

Nejprve je vytvořena prázdná dočasná tabulka a je zjištěno dnešní datum. Pak je ve smyčce WHILE přičteno postupně 30 dnů a výsledky jsou průběžně ukládány do dočasné tabulky. Nakonec je obsah této dočasné tabulky vrácen jako výsledek.

Trigger

Uložená procedura, která se spouští v souvislosti (před nebo po) s provedením nějakého akčního dotazu (UPDATE, DELETE,... , ne SELECT) na tabulce. Nic nevrací a nemá parametry.

```
CREATE TRIGGER trBackup
BEFORE DELETE
ON tabulka
FOR EACH ROW
BEGIN
  INSERT INTO tabulka_zaloha(id, jmeno, plat, cas_odstraneni, uzivatel)
  VALUES (old.id, old.jmeno, old.plat, NOW(), USER());
END;
```

Možné akce: BEFORE/AFTER INSERT/UPDATE/DELETE

Pro každou akci nejvýše jeden trigger, každý trigger pouze pro jednu akci.

Triggery mají přístup k měněným datům přes virtuální tabulky *old* a *new.*

```
CREATE TRIGGER trMaxPlat
BEFORE INSERT
ON tabulka
FOR EACH ROW
BEGIN
```

```
  IF new.plat>30000 THEN /*něco, co akci zruší*/;
  END IF;
END;
```

Bohužel pro „něco, co akci zruší“ neexistuje žádný příkaz, je třeba vyvolat nějakou chybu.

Uživatelsky definované funkce – UDF

```
CREATE FUNCTION mesicslovy (mesic TINYINT)
RETURNS VARCHAR (9)
BEGIN
RETURN CASE mesic
  WHEN 1 THEN 'ledna'
  WHEN 2 THEN 'února'
  ...
  WHEN 12 THEN 'prosince'
END;
END
```

Zdroje

MySQL 5.0 Reference Manual

Seriál MySQL na Linuxsoft.cz

PHP triky