Základy SQL a databáze PostgreSQL

153GIT2

Aleš Čepek

153GIT2

2021-09-29

Obsah I

Obsah II

Úvod

Literatura, odkazy a tutoriály

prezentace	http://geo102.fsv.cvut.cz/user/cepek/git2/git2.pdf
SQLtutor	http://sqltutor.fsv.cvut.cz/
GeoWikiCZ	http://geo.fsv.cvut.cz/wiki/index.php/155GIT2
PostgreSQL	http://www.postgresql.org/docs

Databáze a systém řízení báze dat

Databázový systém (databáze) je kolekce souvisejících dat, která jsou organizována tak, aby umožňovala snadný přístup, správu a údržbu. Jakákoli informace mohou být data, například jméno studenta. Databáze je prostor ve kterém jsou související informace ukládány a jsou na nich prováděny požadované operace.

Systém řízení báze dat (SŘBD) (DBMS - database management system) je software, který umožňuje vytváření, údržbu a správu databáze. SŘBD je nástrojem, která provádí veškeré operace s daty v databázi. ŠRBD také zajišťuje ochranu a bezpečnost databáze. Zajišťuje konzistenci dat v multiuživatelském prostředí. SŘBD je také rozhraním pro databázové aplikace.

Obecně známé SŘBD jsou například Oracle, DB2 (firmy IBM), SQL Server a Access (Micorsoft), PostgreSQL, SQLite, MariaDB, Firebird.

Booleova algebra

George Boole, *An Investigation of the Laws of Thought (1854)*

Booleova algebra pracuje se dvěma hodnotami 1 a 0, nebo se dvěma pravdivostními hodnotami pravda (true) a nepravda (false).

Tři základní operace: součin $\wedge$ (and), součet $\vee$ (or) a negace $\neg$ (not)

x	y	$x \wedge y$	$x \vee y$
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

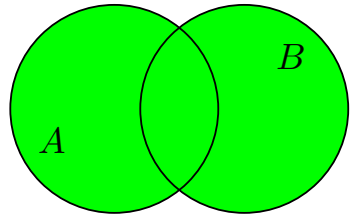
x	$\neg x$
1	0
0	1

De Morganovy zákony

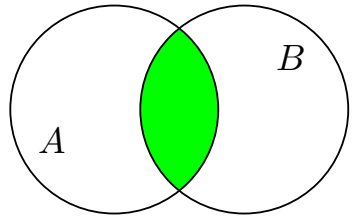
$$\begin{aligned} \neg x \wedge \neg y &= \neg(x \vee y) \\ \neg x \vee \neg y &= \neg(x \wedge y) \end{aligned}$$

Základní množinové operace

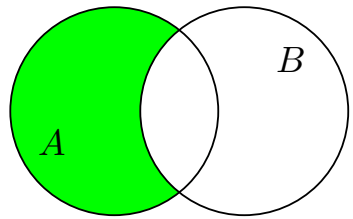
sjednocení $A \cup B = \{x : x \in A \vee x \in B\}$



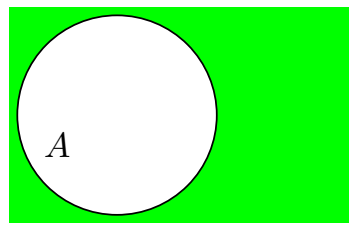
průnik $A \cap B = \{x : x \in A \wedge x \in B\}$



rozdíl $A - B = \{x : x \in A \wedge x \notin B\}$



doplněk $-A = \{x : x \notin A\}$



kartézský součin $A \times B = \{|x, y| : x \in A \wedge y \in B\}$

Jazyk SQL, první zmínka

Dotazy SQL

- veškerá data jsou ukládána v tabulkách
- výsledkem SQL dotazu je vždy množina, resp. tabulka
- SQL je neprocedurální jazyk
- dotazy SQL popisují **co** má mýt výsledkem dotazu (množina) a ne **jak** máme výsledek získat
- tabulku tvoří *řádky* (*n*-tice), jde o množinu a pořadí proto není určující
- všechny řádky v tabulce mají stejný počet sloupců, resp. atributů
- všechny hodnoty v daném sloupci mají stejný typ (doména)
- údaj na průsečíku daného řádku a daného sloupce je považován za dál nedělitelný (atomický)
- některé údaje mohou nabývat hodnoty `NULL`

Slavní skladatelé

skladatel	*narozen*	*zemrel*
J. S. Bach	1685	1750
W. A. Mozart	1756	1791
L. Beethoven	1770	1827
F. Chopin	1810	1849
R. Schumann	1810	1856
B. Bartók	1881	1945
E. Cavalieri		1602

- Tabulka `skladatele` má sedm řádků a tři sloupce (atributy).
- Sloupce tabulky mají jména (`skladatel`, `narozen` a `zemrel`). Jména sloupců uvádíme pro lepší čitelnost, nejsou ale explicitně součástí výsledku dotazu.
- V tabulce `skladatele` považujeme v našem příkladu hodnoty `skladatel` za dále nedělitelné.
- Emilio de Cavalieri nemá uveden rok narození (není znám), v tabulce je uvedena hodnota `NULL`.

Tři základní množinové operace s tabulkami

projekce *(projection)* v příkazu `SELECT` specifikujeme, které sloupce mají být uvedeny ve výsledku dotazu

```
SELECT skladatel, narozen
  FROM skladatele;
```

Výsledkem je projekce tabulky `skladatele` do tabulky o dvou sloupcích (jméno a narození) a šesti řádcích.

restrikce *(restriction)* definuje podmnožinu řádků, které mají být výsledkem dotazu

```
SELECT skladatel, narozen, zemrel
  FROM skladatele
 WHERE 1800 <= narozen AND narozen < 1900;
```

Výběr skladatelů narozených v devatenáctém století.

součin *(join)* SQL klauzule `JOIN` umožňuje spojovat dvě nebo více tabulek, obvykle podle shodných hodnot zvolených sloupců

SQLite

SQLite

SQLite je jedním z představitelů tvz. *light* databází.
Nevyžaduje instalaci, pro vytvoření databáze stačí zadat její jméno.

Kromě interaktivní terminálu `sqlite3` existují i grafická uživatelská rozhraní.

http://www.sqlite.org

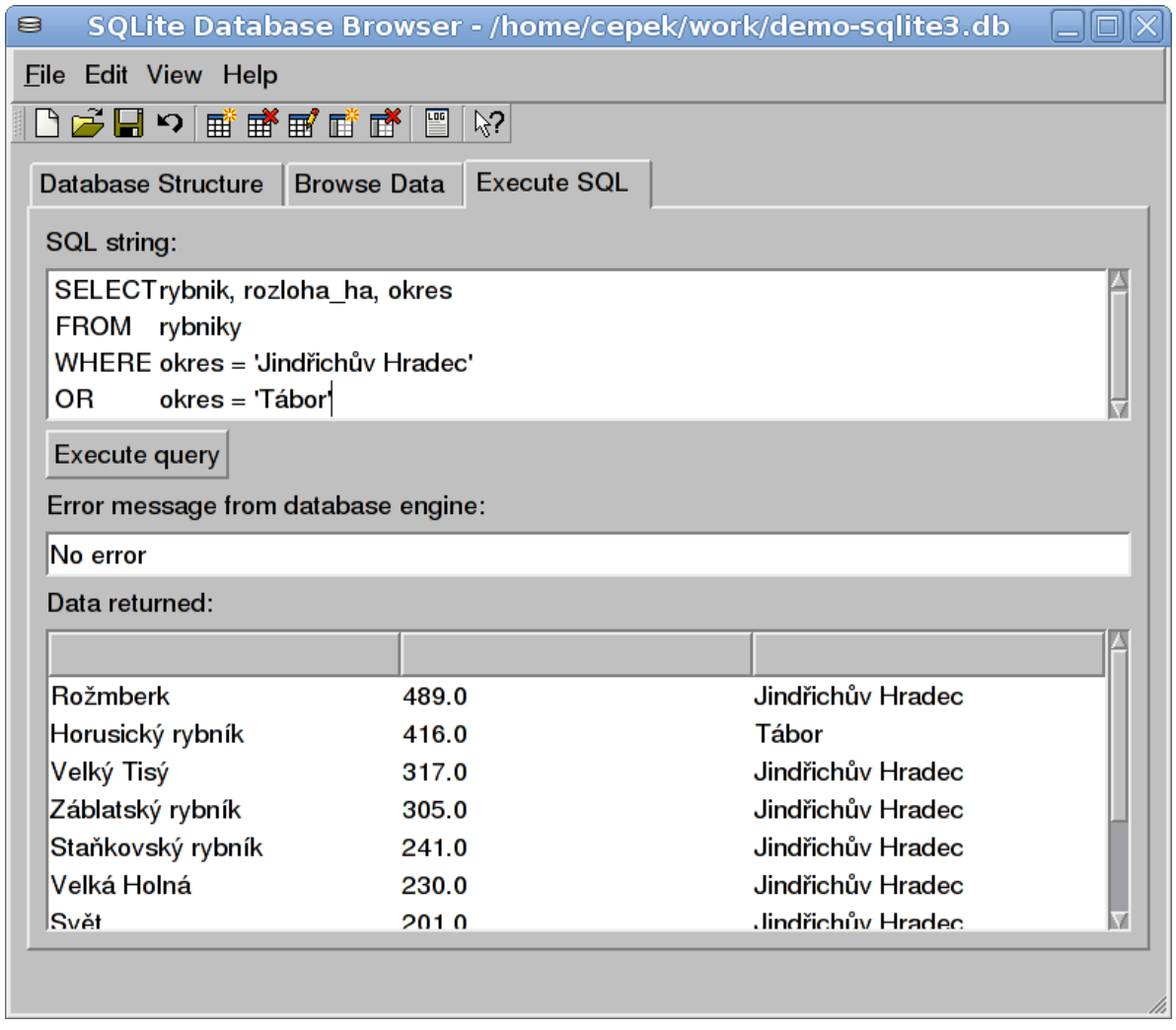
Databáze SQLite tvoří jeden soubor, který je přenositelný mezi různými operačními systémy. Je ideálním nástrojem pro učení jak vytvřet tabulky a měnit jejich obsah.

Základní poznámky a odkazy uvádí oborová wiki

http://geo.fsv.cvut.cz/wiki/index.php/SQLite

SQLite Database Browser

http://sqlitebrowser.sourceforge.net/



Otevření demo databáze v SQLite

```
$
$ sqlite3  demo-sqlite3.db
SQLite version 3.5.9
Enter ".help" for instructions
sqlite> .schema skladatele
CREATE TABLE skladatele (
   skladatel  VARCHAR(20),
   narozen    INTEGER,
   zemrel     INTEGER
);
sqlite> select * from skladatele;
J. S. Bach|1685|1750
W. A. Mozart|1756|1791
L. Beethoven|1770|1827
F. Chopin|1810|1849
R. Schumann|1810|1856
B. Bartók|1881|1945
E. Cavalieri||1602
sqlite> .quit
$
```

Databázový systém PostgreSQL

Databázový systém PostgreSQL

Základní informace a poznámky uvádí GeoWikiCZ

pgadmin multiplatformní grafický klient pro administraci PostgreSQL http://www.pgadmin.org/

psql klientská aplikace pro interaktivní přístup k databázovému systému PostgreSQL.

sql je jazyk, který můžeme zapisovat v textových souborech libovolným editorem (například gedit nebo kate)

emacs podporuje přímý přístup k databázi prostřednictvím psql

Stručná historie

Databázové modely

hierarchické modely byly založeny na modelování hierarchie mezi entitami

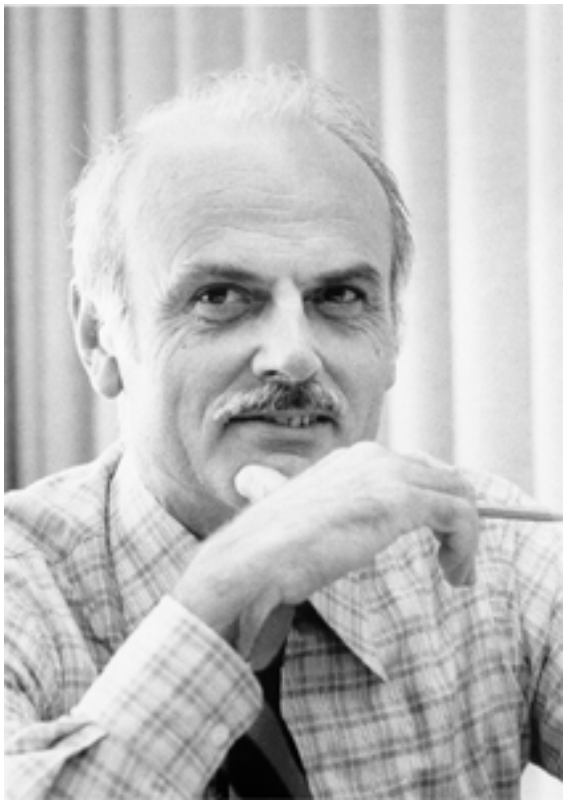
síťové modely představovaly zdokonalení hierarchických modelů. Modelování vztahů mezi entitami vycházelo z teorie grafů, orientované hrany modelovaly vztahy mezi entitami. Síťové databáze dominovaly v 80. letech.

S příchodem relačních modelů jsou hierarchické a síťové databáze překonanou vývojovou větví.

relační model je výsledkem výzkumu IBM z počátku 70. let. Relační databázovou teorii poprvé popsal Dr. E. F. Codd v článku “A Relational Model of Data for Large Shared Data Banks” publikovaném v *Communications of the ACM* (Association for Computer Machinery) v červnu 1970.

objektově relační SQL:1999 zavádí objektové rysy

Edgar Frank Codd



August 23, 1923 – April 18, 2003

http://en.wikipedia.org/wiki/Edgar_F._Codd

Relační databázový model

Relational Database Management System (RDBMS)

- základem je relační algebra a relační model dat
- v systému uživatelé vidí data jako soubor tabulek vzájemně propojených přes společné údaje
- tabulky mohou být vzájemně propojeny pomocí tzv. klíčů, které jednoznačně určují všechny řádky tabulky
- systém umožňuje provádět na tabulkách operace jako projekce, restrikce, join a další pomocí jednotného jazyka, nezávisle na fyzické implementaci databáze
- systém zaručuje udržení *referenční integrity*

E. F. Codd formuloval dvanáct pravidel relačních databází *(Twelve Principles of Relational Databases)*. V současnosti většina RDBMS splňuje všechny tyto požadavky.

12 pravidel E. F. Codda

12 pravidel E. F. Codda

1. Informace jsou logicky prezentovány jako tabulky.
2. Data musí být logicky dostupná přes tabulky, primární klíče a sloupce.
3. Hodnoty `NULL` musí být interpretovány jako *chybějící informace* a ne jako prázdné řetězce, mezery nebo nuly.
4. Metadata (informace o databázi) musí být ukládány v databázi stejně jako regulární data.
5. Jediný jazyk musí být schopen definovat data, pohledy, integritní omezení, autorizaci, transakce a manipulaci s daty (v praxi je tímto jazykem SQL)
6. Pohledy *(views)* musí zobrazovat aktualizace jejich bázových tabulek a naopak.

12 pravidel E. F. Codda

7. Jedna operace musí být k dispozici pro každou z následujících operací: výběr dat, vkládání dat, aktualizace dat a rušení dat.
8. Dávkové i uživatelské operace musí být logicky odděleny od fyzického uložení a přístupových metod.
9. Dávkové i uživatelské operace mohou změnit schéma databáze bez toho aniž by muselo být znovu vytvořeno právě tak jako aplikace, které jsou na něm postaveny.
10. Integritní omezení musí být přístupné a uloženy v metadatech, ne v aplikačních programech.
11. Jazyk pro manipulaci s daty v relačním systému nesmí být závislý na tom, jak jsou data fyzicky distribuována ani nesmí vyžadovat úpravy pokud jsou jsou fyzicky data centralizovaná nebo distribuovaná.
12. V systému musí jakékoli zpracování po řádcích podléhat stejným integritním pravidlům a omezením jako množinové operace.

Některé základní pojmy

Některé základní pojmy

V relačním modelu jsou data logicky zobrazována jako *dvourozměrné* tabulky. Dodržujeme konvenci, že jména tabulek volíme v množném čísle nebo jako pomnožná jména (`skladatele`, `studenti`, `vydaje` a pod.).

tabulky entity

řádky záznamy nebo *n*-tice (angl. tuples)

sloupce atributy

Průsečíkem jednoho záznamu a jednoho sloupce je (z pohledu databáze) jedna jediná *hodnota*.

stupeň je počet atributů dané tabulky (degree, arity)

kardinalita je počet řádků tabulky (cardinality)

Některé základní pojmy

klastry (clusters)

katalogy (catalogs) Každý databázový systém musí mít informace o schématech, která obsahuje. Pro každé schéma to zahrnuje minimálně následující metainformace:

- jména relací ve schématu
- jména sloupců každé relace a datový typ každého sloupce
- integritní omezení relace
- informace o indexech dané relace
- přístupová práva pro elementy relací

Této metadatabázi se často říká *system catalog* (Oracle používá označení *data dictionary*).

schémata (schemas) jednoznačně pojmenované množiny objektů a dat. Každý katalog musí obsahovat `INFORMATION_SCHEMA`, s metadaty o všech objektech daného katalogu

objekty (objects) tabulky, pohledy, moduly, úložné procedury

Pokud je objektem *tabulka* nebo *pohled* (view), může obsahovat jeden nebo více

sloupce (columns)

domény a uživatelské typy

pravidla a předpoklady (rules and assertions)

Historie SQL standardu

Existuje řada dialektů SQL, které se liší pro různé databázové systémy.

1986 první publikovaný ANSI standard SQL

1987 jazyk SQL přijat jako ISO standard

1989 revidovaná verze standardu rozšířená o integritní omezení

1992 též jako SQL-92 nebo SQL3

1999 SQL:1999

2003 SQL:2003

2006 SQL:2006

2008 SQL:2008

2011 SQL:2011

2016 SQL:2016

Různé systémy se liší v úrovni s jakou vyhovují standardu.

Významné relační databázové systémy

Oracle	Oracle
DB2	IBM
SQL server	Microsoft
PostgreSQL	http://www.postgresql.org/
SQLite	http://www.sqlite.org/
MariaDB	https://mariadb.org/
Firebird	http://www.firebirdsql.org/

SQL SELECT

SQL SELECT

```
SELECT projekce
  FROM tabulka
[ WHERE restrikce ] ;
```

Příklady:

```
SELECT * FROM skladatele;

SELECT skladatel, narozen
  FROM skladatele
 WHERE narozen > 1800;

SELECT skladatel, (zemrel-narozen)
  FROM skladatele;
```

Relační a logické operátory

operátor	*popis*
=	je rovno
<>	není rovno (nebo též !=)
<	je menší
>	je větší
<=	je menší nebo rovno
>=	je větší nebo rovno

OR logický operátor *nebo*

AND logický operátor *a současně*

NOT operátor negace

Relační a logické výrazy je možno v případě potřeby (nebo pro lepší čitelnost) uzávorkovat.

Největší české rybníky

rybnik	*okres*	*rozloha ha*	*hloubka_m*	*odtok*	*povodi*
Rožmberk	Jindřichův Hradec	489	6.2	Lužnice a Potěšilka	Lužnice
Horusický rybník	Tábor	416	6	Zlatá stoka	Lužnice
Bezdrev	České Budějovice	394	7	Netolický potok	Vltava
Dvořiště	České Budějovice	337	4.5	Zlatá stoka	Lužnice
Velký Tisý	Jindřichův Hradec	317	3.4	Zlatá stoka	Lužnice
Záblatský rybník	Jindřichův Hradec	305	3	Zlatá stoka	Lužnice
Nesyt	Břeclav	296	3	Včelínek	Dyje
Máchovo jezero	Česká Lípa	284	12	Robečský potok	Ploučnice
Žehuňský rybník	Nymburk	258	6	Cidlina	Labe
Dehtář	České Budějovice	246	6	Dehtářský potok	Vltava
Staňkovský rybník	Jindřichův Hradec	241	8.5	Koštěnický potok	Lužnice
Velká Holná	Jindřichův Hradec	230	3	Holenský potok	Nežárka
Velké Dářko	Žďár nad Sázavou	205	3	Sázava	Vltava
Svět	Jindřichův Hradec	201	3	Spolský potok	Lužnice
Kačležský rybník	Jindřichův Hradec	196	2	Koštěnický potok	Lužnice
Koclířov	Jindřichův Hradec	192		Zlatá stoka	Lužnice
Opatovický rybník	Jindřichův Hradec	161	3	Zlatá stoka	Lužnice
Bohdanečský rybník	Pardubice	160	2	Opatovický kanál	Labe

Příklad SQL dotazu

```
SELECT rybnik, okres, rozloha_ha,
       hloubka_m, odtok, povodi
  FROM rybniky;
```

Největší české rybníky - příklady

```
rybniky (rybnik, okres, rozloha_ha, hloubka_m, odtok, povodi
```

- Které rybníky mají plochu větší než 300 hektarů? Vypište vodní plochu a rybník.
- Vypište jména rybníků z okresu Jindřichův Hradec.
- Vypište jména rybníků z okresu Jindřichův Hradec, které nepatří do povodí Lužnice.
- Které rybníky mají hloubku větší než 4 metry?
- Které rybníky z okresu České Budějovice mají hloubku větší než 4 metry?
- Které rybníky mají hloubku větší než 4 metry a rozlohu větší než 400 hektarů?
- Které rybníky mají hloubku větší než 4 metry, rozlohu větší než 400 hektarů a nenacházejí se v okrese Tábor?
- Které rybníky mají odtok do Zlaté stoky, jakou mají plochu a hloubku?

Seznam sloupců příkazu SELECT

V seznamu sloupců příkazu `SELECT` lze použít kromě jmen sloupců i výrazy a funkce. Zvláštní význam má symbol `*`, který označuje všechny sloupce.

```
SELECT 'konstatní řetězec';

SELECT (zemrel - narozen)   -- kolika let se dožil
  FROM skladatele;

SELECT random();              -- náhodné číslo <0,1)
SELECT 50+10*random();        -- náhodné číslo <50,60)
```

V seznamu příkazu `SELECT` mohou být sloupce výsledné tabulky explicitně pojmenovány

```
SELECT skladatel AS jmeno,
       (zemrel-narozen) AS "věk skladatele"
  FROM skladatele;
```

Aritmetické operátory

operátor	*popis*
+	součet
−	rozdíl
*	násobení
/	dělení
%	zbytek po dělení
( )	závorky

Priority

1. závorky
2. násobení a dělení
3. sčítání a odčítání
4. `NOT`
5. `AND`
6. `OR`

Intervaly (BETWEEN a NOT BETWEEN)

```
SELECT * FROM skladatele
 WHERE narozen BETWEEN 1810 AND 1881;
```

skladatel	*narozen*	*zemrel*
F. Chopin	1810	1849
R. Schumann	1810	1856
B. Bartók	1881	1945

```
SELECT * FROM skladatele
 WHERE narozen NOT BETWEEN 1810 AND 1881;
```

skladatel	*narozen*	*zemrel*
J. S. Bach	1685	1750
W. A. Mozart	1756	1791
L. Beethoven	1770	1827

Poznámka: operátor `BETWEEN` definuje *uzavřený interval*, tj. zadané meze jsou součástí intervalu.

Seznamy (IN a NOT IN)

```
SELECT * FROM skladatele
 WHERE narozen IN (1810, 1770, 1685);
```

skladatel	*narozen*	*zemrel*
J. S. Bach	1685	1750
L. Beethoven	1770	1827
F. Chopin	1810	1849
R. Schumann	1810	1856

```
SELECT * FROM skladatele
 WHERE narozen NOT IN (1810, 1770, 1685);
```

skladatel	*narozen*	*zemrel*
W. A. Mozart	1756	1791
B. Bartók	1881	1945

Poznámka: Seznam nemusí být statický a jak si ukážeme později, může být generován příkazem `SELECT`.

Výběr hodnot IS NULL

```
SELECT rybnik, okres, hloubka_m, povodi
  FROM rybniky
 WHERE hloubka_m IS NULL
       OR povodi = 'Labe';
```

rybnik	*okres*	*hloubka m*	*povodi*
Žehuňský rybník	Nymburk	6	Labe
Koclířov	Jindřichův Hradec		Lužnice
Bohdanečský rybník	Pardubice	2	Labe

Poznámka: pro hodnoty `NULL` nelze používat relační operátory, jmenovitě operátory = a <> ani žádné jiné.

Výběr hodnot IS NOT NULL

```
SELECT rybnik, okres, hloubka_m, povodi
  FROM rybniky
 WHERE hloubka_m IS NOT NULL
       AND povodi = 'Lužnice';
```

rybnik	*okres*	*hloubka m*	*povodi*
Rožmberk	Jindřichův Hradec	6.2	Lužnice
Horusický rybník	Tábor	6	Lužnice
Dvořiště	České Budějovice	4.5	Lužnice
Velký Tisý	Jindřichův Hradec	3.4	Lužnice
Záblatský rybník	Jindřichův Hradec	3	Lužnice
Staňkovský rybník	Jindřichův Hradec	8.5	Lužnice
Svět	Jindřichův Hradec	3	Lužnice
Kačležský rybník	Jindřichův Hradec	2	Lužnice
Opatovický rybník	Jindřichův Hradec	3	Lužnice

Výběr podobných řetězců (LIKE a NOT LIKE)

Operátor `LIKE` umožňuje definovat vzory (jednoduché regulární výrazy) pomocí žolíkových znaků `%` (procento) a _ (podtržítko):

- `%` libovolný řetězec (může být i prázdný)
- `_` jeden libovolný znak (právě jeden)

WHERE *výraz* [**NOT**] **LIKE** *vzor* [**ESCAPE** '*escape znak*']

```
SELECT rybnik AS "Velké rybníky"
  FROM rybniky
 WHERE rybnik LIKE 'Velk_ %';
```

Velké rybníky
Velký Tisý
Velká Holná
Velké Dářko

Výběr podobných řetězců (LIKE a NOT LIKE)

```
SELECT rybnik, okres, odtok
  FROM rybniky
 WHERE odtok LIKE '%potok';
```

rybnik	*okres*	*odtok*
Bezdrev	České Budějovice	Netolický potok
Máchovo jezero	Česká Lípa	Robečský potok
Dehtář	České Budějovice	Dehtářský potok
Staňkovský rybník	Jindřichův Hradec	Koštěnický potok
Velká Holná	Jindřichův Hradec	Holenský potok
Svět	Jindřichův Hradec	Spolský potok
Kačležský rybník	Jindřichův Hradec	Koštěnický potok

Výběr podobných řetězců (LIKE a NOT LIKE)

```
SELECT rybnik, okres, odtok
  FROM rybniky
 WHERE odtok NOT LIKE '%potok';
```

rybnik	*okres*	*odtok*
Rožmberk	Jindřichův Hradec	Lužnice a Potěšilka
Horusický rybník	Tábor	Zlatá stoka
Dvořiště	České Budějovice	Zlatá stoka
Velký Tisý	Jindřichův Hradec	Zlatá stoka
Záblatský rybník	Jindřichův Hradec	Zlatá stoka
Nesyt	Břeclav	Včelínek
Žehuňský rybník	Nymburk	Cidlina
Velké Dářko	Žďár nad Sázavou	Sázava
Koclířov	Jindřichův Hradec	Zlatá stoka
Opatovický rybník	Jindřichův Hradec	Zlatá stoka
Bohdanečský rybník	Pardubice	Opatovický kanál

Vytváření a úpravy tabulek

Komentáře

V naprosté většině případů nebudeme zadávat SQL příkazy interaktivně, ale budeme je zapisovat do textových souborů, které budeme předávat ke zpracování typicky prostřednictvím klientského programu `psql`.
Pro lepší čitelnost a pro dokumentační účely slouží komentáře, které v jazyce SQL jsou dvojího typu:

jednořádkové jsou uvozeny dvojicí znaků mínus

víceřádkové začínají dvojicí znaků `/*` a končí dvojicí znaků `*/`

```
/* víceřádkové komentáře jsou
   stejné jako v jazycích C a C++ */

SELECT * FROM rybniky;  -- jednořádkový komentář
```

V `SQL` kódu jsou komentáře interpretovány stejně jako *bílé znaky.*

CREATE TABLE

```
CREATE TABLE skladatele (
    skladatel   VARCHAR(20),
    narozen     INTEGER,
    zemrel      INTEGER
);
```

-- příklad příkazu `INSERT` pro vložení údajů do tabulky

```
INSERT INTO skladatele VALUES('J. S. Bach',    1685, 1750);
INSERT INTO skladatele VALUES('W. A. Mozart', 1756, 1791);
INSERT INTO skladatele VALUES('L. Beethoven', 1770, 1827);
INSERT INTO skladatele VALUES('F. Chopin',     1810, 1849);
INSERT INTO skladatele VALUES('R. Schumann',   1810, 1856);
INSERT INTO skladatele VALUES('B. Bartók',     1881, 1945);
```

Některé základní typy

varchar(n), character varying(n) textový řetězec se zadaným maximálním počtem znaků

numeric(m,n), decimal(m,n) číselný typ s rozsahem *m* a *n* desetinnými číslicemi

double precision, real, float reálné číselné typy

integer, int celočíselný typ

date datum: rok, měsíc a den

time čas: hodiny, minuty a vteřiny

timestamp datum a čas s vteřinami na 6 desetinných míst

serial nestandardní typ PostgreSQL

text obecný textový řetězec, obvykle většího rozsahu

Implicitní hodnoty

```
CREATE TABLE produkty (
   produkt_id int,
   nazev varchar(30),
   cena decimal(10,2),
   sleva decimal( 2,2) DEFAULT 10.5
);
```

- není-li při uložení záznamu uvedena hodnota atributu a nemá-li tento atribut deklarovánu implicitní hodnotu, použije se `NULL`
- implicitní hodnota je výraz, který se vyhodnocuje při vkládání údajů a ne při vytvoření tabulky

```
CREATE TABLE produkty (
   produkt_id int
          DEFAULT nextval('produkty_produkt_id_seq'),
     ...   -- v PostgreSQL můžeme použít typ serial
);
```

Podmínky a omezení — CHECK

Zajištění datové integrity

```
CREATE TABLE produkty (
   produkt_id int,
   nazev varchar(30),
   cena decimal(10,2) CHECK (cena > 0)
);
```

Pojmenovaná podmínka

```
CREATE TABLE produkty (
   produkt_id int,
   nazev varchar(30),
   cena decimal(10,2)
         CONSTRAINT nezaporna_cena CHECK (cena > 0)
);
```

Podmínky a omezení — CHECK

Podmínka může odkazovat na více sloupců a může být pojmenováná

```
CREATE TABLE produkty (
   produkt_id int,
   nazev varchar(30),
   cena decimal(10,2) CHECK (cena > 0),
   sleva decimal(10,2) CHECK (sleva > 0),
 [ CONSTRAINT platna_sleva ] CHECK (cena > sleva)
);
```

nebo např.

```
CREATE TABLE produkty (
   produkt_id int,
   nazev varchar(30),
   cena decimal(10,2) CHECK (cena > 0),
   sleva decimal(10,2),
   CHECK (sleva > 0 AND cena > sleva)
);
```

Podmínky a omezení — NOT NULL

```
CREATE TABLE produkty (
    produkt_id int NOT NULL,
    nazev varchar(30) NOT NULL,
    cena decimal(10,2),
    sleva decimal(10,2)
);
```

- pouze jako podmínka sloupce (a ne tabulky)
- stejný význam jako `CHECK (`*`sloupec`*` IS NOT NULL)`
- pořadí vícenásobných podmínek není významné
- alternativní podmínka NULL není součástí standardu SQL a neměla by být používána

Podmínky a omezení — UNIQUE

Ve sloupci s omezením UNIQUE se mohou vyskytovat hodnoty NULL, dvě hodnoty NULL jsou považovány za různé.

```
CREATE TABLE produkty (
    produkt_id int UNIQUE,
    nazev varchar(30),
    cena decimal(10,2),
    sleva decimal(10,2)
);
```

nebo

```
CREATE TABLE produkty (
    produkt_id int,
    nazev varchar(30),
    cena decimal(10,2),
    sleva decimal(10,2),
    UNIQUE (produkt_id)          -- obecně i seznam více sloupců
);
```

Podmínky a omezení — UNIQUE NOT NULL

```
CREATE TABLE produkty (
    produkt_id int UNIQUE NOT NULL,
    nazev varchar(30),
    cena decimal(10,2),
    sleva decimal(10,2)
);
```

nebo

```
CREATE TABLE produkty (
    produkt_id int PRIMARY KEY,
    nazev varchar(30),
    cena decimal(10,2),
    sleva decimal(10,2)
);
```

PRIMARY KEY (první zmínka)

- Každá databázová tabulka musí mít definován primární klíč, který je základním integritním omezením tabulky.
- Primární klíč je atribut nebo seznam atributů, které jednozančně určují daný řádek. V tabulce nelze vytvořit dva řádky se stejným primárním klíčem.
- Restrikce v klauzuli WHERE přes neklíčové atributy vede na sekvenční prohledávání.[1]

```
CREATE TABLE abc
(
   a  int PRIMARY KEY,
   b  int,
   c  int
);
```

```
CREATE TABLE xyz
(
   x  int,
   y  int,
   z  int,
   PRIMARY KEY(x, y)
);
```

[1] Pokud není definován index

Podmínky a omezení — PRIMARY KEY

```
CREATE TABLE produkty (
    produkt_id int,
    nazev varchar(30),
    cena decimal(10,2),
    sleva decimal(10,2),
    PRIMARY KEY (produkt_id)    -- obecně seznam sloupců
);
```

- primární klíč jednoznačně identifikuje jednotlivé záznamy dané tabulky
- tabulka může obsahovat jen jeden primární klíč
- počet sloupců s podmínkou `NOT NULL UNIQUE` není omezen

Primární klíč může být explicitně pojmenován

```
CREATE TABLE produkty (
   produkt_id int,
   nazev varchar(30),
   cena decimal(10,2),
   sleva decimal(10,2),
   CONSTRAINT jmeno_pklice PRIMARY KEY (produkt_id)
);
```

Podmínky a omezení — FOREIGN KEY

```
CREATE TABLE produkty (
    produkt_id int PRIMARY KEY,
    nazev varchar(30),
    cena decimal(10,2)
);
CREATE TABLE objednavky (
    objednavka_id int PRIMARY KEY,
    produkt_id int REFERENCES produkty (produkt_id),
    pocet int
);
```

nebo jen

```
CREATE TABLE objednavky (
    objednavka_id int PRIMARY KEY,
    produkt_id int REFERENCES produkty,   -- stejné jméno
    pocet int
);
```

Podmínky a omezení — FOREIGN KEY

```
CREATE TABLE tabulka (
    ...
    FOREIGN KEY ( seznam sloupců ) REFERENCES
        cizí tabulka ( seznam sloupců cizí tabulky )
    ...
);
```

- Tabulka může obsahovat více cizích klíčů
- Omezení definovaná cizími klíči znemožňují uložení záznamů, pro které neexistuje odkazovaný primární klíč.
- Operace aktualizace a rušení později.

Úpravy dat

Dočasné tabulky

```
CREATE [LOCAL|GLOBAL] TEMPORARY TABLE jméno ( ... );
```

nebo

```
CREATE [LOCAL|GLOBAL] TEMP TABLE jméno ( ... );
```

- dočasné tabulky jsou automaticky zrušeny po ukončení seance.
- dočasné tabulky jsou obvykle známkou špatného návrhu databáze, nepoužívejte je.
- místo dočasných tabulek používejte poddotazy, derivované tabulky nebo pohledy.

Vkládání dat

Zjednodušená syntax:

```
INSERT INTO {tabulka | pohled} [ ( seznam sloupců ) ]
    { DEFAULT VALUES |
      VALUES ( seznam hodnot ) |
      příkaz SELECT }
```

Implicitní hodnoty:

```
INSERT INTO produkty (produkt_id, nazev, cena, sleva)
            VALUES (1257, 'USB myš', 169.0, DEFAULT);
INSERT INTO produkty DEFAULT VALUES;
```

Možnost uvést pouze několik hodnot bez explicitního seznamu atributů s tím, že ostatním sloupcům se doplní implicitní hodnoty je nestandardní rozšíření Postgresu.

Příklady vkládání dat

vložení implicitních hodnot

```
INSERT INTO t1 DEFAULT VALUES;
```

ruzné varianty vkládání jednotlivých řádků

```
INSERT INTO t1 (num, txt) VALUES (DEFAULT, 'x');
INSERT INTO t1 (num, txt) VALUES (10, DEFAULT);
INSERT INTO t1 (num) VALUES (20);
```

vložení dat příkazem INSERT

```
INSERT INTO t1 (num, txt) SELECT a, b from t2;
```

Úpravy a odstraňování dat

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

```
DELETE FROM t1 WHERE num > 10;
DELETE FROM t1;                          -- odstraní všechny řádky
```

Poznámka: PostgreSQL a příkaz VACUUM

```
UPDATE tabulka
   SET sloupec = { skalární výraz | DEFAULT | NULL }
 WHERE podmínka
```

V klauzuli SET může být použito přiřazení jednou nebo vícekrát.

```
UPDATE t1 SET num = num+100;      -- aktualizace všech řádků
UPDATE t1 SET num = DEFAULT WHERE num IS NULL;
UPDATE t1 SET num = 0, TXT='zzz' WHERE num = -1;
```

Úpravy a odstraňování tabulek

Úpravy tabulek

```
ALTER TABLE tabulka
[ADD [COLUMN]] sloupec  datový_typ  atributy ]
| [ALTER [COLUMN] sloupec SET DEFAULT hodnota ]
| [ALTER [COLUMN] sloupec DROP DEFAULT ]
| [DROP [COLUMN] sloupec ]
| [ADD omezující_podmínka]
| [DROP CONSTRAINT jméno_podmínky {RESTRICT|CASCADE}]
```

Příklad:

```
ALTER TABLE t1 ADD CONSTRAINT t1_podminka
   FOREIGN KEY (num) REFERENCES t2 (a);
```

Odstraňování tabulek

DROP TABLE [IF EXISTS] *tabulka* [CASCADE | RESTRICT]

Pokud se pokusíme odstranit neexistující tabulku a nepoužijeme IF EXISTS, skončí operace chybou.

CASCADE automaticky odstraní i všechny objekty, které závisejí na tabulce (pohledy, indexy a pod.)

RESTRICT zabrání zrušení tabulky jestliže na ní závisí jakýkoli objekt. Implicitní hodnota.

Příklad:

```
DROP TABLE t1, t2;    -- odstranění dvou tabulek
```

PostgreSQL pg_dump, pg_restore a pg_dumpall

pg_dumpall extrahuje celý databázový cluster

pg_restore obnova databáze ze souboru

pg_dump extrahuje obsah databáze do souboru

-a --data-only pouze data a ne schéma (definice dat)

-c --clean výstup příkazů pro zrušení databázových objektů před jejich zrušením (drop)

-d --inserts dump dat s příkazy INSERT a ne COPY

-s --schema-only dump objektů schéma a ne data

-t *tabulka* --table=*tabulka* dump pouze zadané tabulky

-n *schéma* --schema=*schéma* dump pouze zadané tabulky

-x --no-priviledes --no-acl dump bez přístupových práv

-h *počítač*

-U *uživatel*

. . . a další přepínače

PostgreSQL klient psql

`psql` je textová klientská aplikace pro interaktivní a dávkový přístup k databázovému systému PostgreSQL.

```
psql [ volby... ] [ jméno_db [ uživatel ] ]
```

Po spuštění v interaktivním režimu se vypisuje základní nápověda

```
cepek @amonit:~$ psql sqltutor
Welcome to psql 8.3.6, the PostgreSQL interactive termin

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

sqltutor=>
```

Nápověda psql - příklad 1

```
sqltutor=> \h

Available help:
  ABORT                            DISCARD
  ALTER AGGREGATE                  DROP AGGREGATE
  ALTER CONVERSION                 DROP CAST
  ALTER DATABASE                   DROP CONVERSION
  ALTER DOMAIN                     DROP DATABASE
  ALTER FUNCTION                   DROP DOMAIN
  ALTER GROUP                      DROP FUNCTION
  ALTER INDEX                      DROP GROUP
  ALTER LANGUAGE                   DROP INDEX
  ALTER OPERATOR CLASS             DROP LANGUAGE
  ALTER OPERATOR                   DROP OPERATOR CLASS
  ALTER OPERATOR FAMILY            DROP OPERATOR
  ALTER ROLE                       DROP OPERATOR FAMILY
  ALTER SCHEMA                     DROP OWNED
  ALTER SEQUENCE                   DROP ROLE
  ALTER TABLE                      DROP RULE
  ALTER TABLESPACE                 DROP SCHEMA
  ALTER TRIGGER                    DROP SEQUENCE
```

Nápověda psql - příklad 2

```
sqltutor=> \h alter table

Command:     ALTER TABLE
Description: change the definition of a table
Syntax:
ALTER TABLE [ ONLY ] name [ * ]
    action [, ... ]
ALTER TABLE [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column TO new_column
ALTER TABLE name
    RENAME TO new_name
ALTER TABLE name
    SET SCHEMA new_schema

where action is one of:

    ADD [ COLUMN ] column type [ column_constraint [ ...
    DROP [ COLUMN ] column [ RESTRICT | CASCADE ]
    ALTER [ COLUMN ] column TYPE type [ USING expression
    ALTER [ COLUMN ] column SET DEFAULT expression
```

Příklad — přidávání a odebírání pojmenovaných klíčů

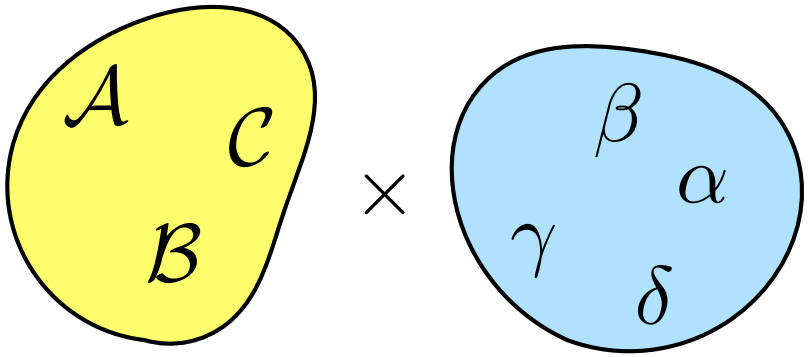
```
DROP TABLE taba CASCADE;
DROP TABLE tabb CASCADE;

CREATE TABLE taba (
    a int,
    b int,
    CONSTRAINT pkeya PRIMARY KEY (a)
);

CREATE TABLE tabb (
    y int,
    z int,
    CONSTRAINT fkeyb FOREIGN KEY (z) REFERENCES taba
);
```

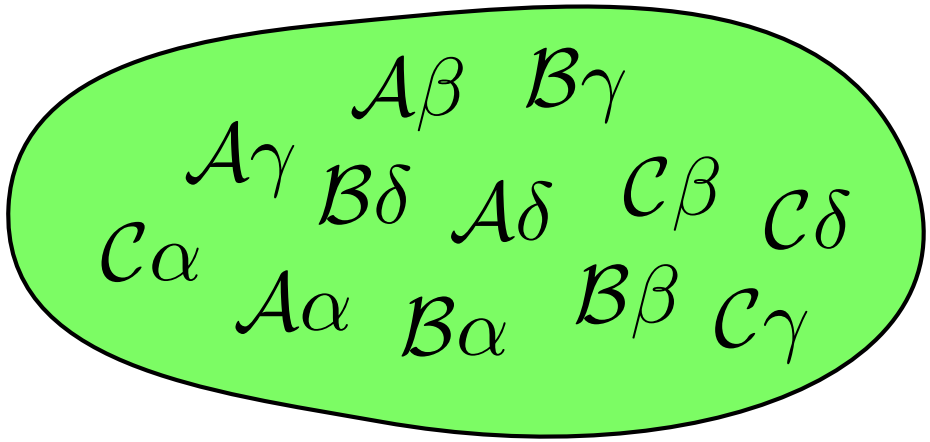
Příklad — přidávání a odebírání pojmenovaných klíčů

```
INSERT INTO taba VALUES (1, 10);
INSERT INTO taba VALUES (2, 20);
INSERT INTO tabb VALUES (11, 1);

ALTER TABLE tabb DROP CONSTRAINT fkeyb;
ALTER TABLE taba DROP CONSTRAINT pkeya;

INSERT INTO taba VALUES (4, 40);
INSERT INTO taba VALUES (5, 50);
INSERT INTO tabb VALUES (44, 4);

ALTER TABLE taba ADD CONSTRAINT pkeya PRIMARY KEY (a);
ALTER TABLE tabb ADD CONSTRAINT fkeyb FOREIGN KEY (z)
                                      REFERENCES taba;
INSERT INTO taba VALUES (7, 70);
INSERT INTO taba VALUES (8, 80);
INSERT INTO tabb VALUES (77, 7);

SELECT * FROM taba;
SELECT * FROM tabb;
```

Klauzule JOIN

Kartézský součin



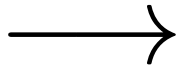


Kartézský součin

num	name
1	a
2	b
3	c

×

num	value
1	xxx
3	yyy
5	zzz



num	name	num	value
1	a	1	xxx
1	a	3	yyy
1	a	5	zzz
2	b	1	xxx
2	b	3	yyy
2	b	5	zzz
3	c	1	xxx
3	c	3	yyy
3	c	5	zzz

Spojování tabulek (ANSI JOIN)

Klauzule JOIN

```
SELECT seznam sloupců
  FROM tabulka 1
       JOIN
       tabulka 2
       ON podmínka pro spojení tabulek
[ WHERE restrikce ];
```

- lze spojovat dvě nebo více tabulek
- (obvykle) tabulky spojujeme porovnáním na rovnost hodnot ve vybraných sloupcích, tj. používáme relační operátor $=$
- *podmínka pro spojení tabulek* definuje výběr ze všech možných prvků kartézského součinu obou tabulek

Poznámka: existuje i tzv. *theta* syntax (nebo též FROM/WHERE syntax), kde se spojované tabulky v klauzuli `FROM` oddělují čárkou, výsledkem je kartézský součin a výběr z něj se uvádí v klauzuli `WHERE`. Tuto syntax nebudeme používat.

Příklad - pražské tramvajové linky

id	*zastavka*
224	U Elektry
201	Solidarita
142	Nuselská radnice
5	Baterie
226	U Kaštanu
178	Pražská tržnice
126	Nádraží Vysočany
125	Nádraží Vršovice
169	Podolská vodárna
85	Kublov
250	Vozovna Strašnice
184	Radlická
238	Vinice
249	Vozovna Pankrác

linka	*smer*	*poradi*	*zastavka id*
26	1	1	120
26	2	1	202
26	2	2	91
26	1	2	52
26	1	3	128
26	2	3	63
26	2	4	214
26	1	4	196
26	1	5	259
26	2	5	114
26	2	6	37
26	1	6	136
26	1	7	185
26	2	7	135
26	2	8	105

Příklad - pražské tramvajové linky

tabulka	*sloupce*
`zastavky`	`id, zastavka`
`linky`	`linka, smer, poradi, zastavka_id`

Které linky projíždějí stanicí Palmovka?

```
SELECT distinct linka
  FROM linky
       JOIN
       zastavky
       ON zastavka_id=id
 WHERE zastavka='Palmovka';
```

Příklad - pražské tramvajové linky

tabulka	*sloupce*
`zastavky`	`id, zastavka`
`linky`	`linka, smer, poradi, zastavka_id`

Vypište zastávky na lince 4.

```
SELECT zastavka
  FROM linky
       JOIN
       zastavky
       ON zastavka_id=id
 WHERE linka=4;
```

Příklad - pražské tramvajové linky

Zastávky na lince 4.

zastavky
Anděl
Anděl
Bertramka
Bertramka
Blatiny
Blatiny
Čechovo náměstí
Čechovo náměstí
Hlušičkova
Hlušičkova
Hotel Golf
Hotel Golf
I. P. Pavlova
I. P. Pavlova
Jana Masaryka
Jana Masaryka
Karlovo náměstí
Karlovo náměstí

Výsledkem předchozího dotazu je seznam zastávek, ve kterém se jména opakují, protože jsou zde uvedeny zastávky pro oba směry.
Pokud ve výsledném dotazu chceme odstranit duplicity, použijeme v příkazu `SELECT` klíčové slovo `DISTINCT`.

```
SELECT {ALL | DISTINCT} seznam
  FROM tabulka
 WHERE restrikce
```

- hodnota `ALL` je implicitní
- duplicity ve výsledku mohou být známkou chybně formulovaného dotazu, volbu `DISTINCT` je proto nutno používat jen v odůvodněných případech

Příklad - pražské tramvajové linky

tabulka	*sloupce*
`zastavky`	`id, zastavka`
`linky`	`linka, smer, poradi, zastavka_id`

Výpis zastávek na tramvajové lince číslo 4 s potlačením duplicit.

```
SELECT DISTINCT zastavka
 FROM linky
      JOIN
      zastavky
      ON zastavka_id=id
WHERE linka=4;
```

ANSI JOIN, opakování a příklady

Spojování a rozdělování tabulek

Otázky

- Proč potřebuje spojování tabulek?
- Proč rozdělujeme informace do různých tabulek?
- Jak spojujeme tabulky?
- Co jsou primární klíče?
- Co jsou cizí klíče?
- Co znamenají pojmy redundance, integrita, normalizace, anomalie, . . . ?

105 pracoviste

Tabulka	*Sloupce*
pracoviste	kod, popis
zamestnanci	id, jmeno, prijmeni, pracoviste_kod, vek
mzdy	id, vlozeno, zamestnanec_id, castka

Vypište jméno, příjmení a částku všech zaměstnanců, jejichž měsíční příjem je nižší než 30000

Radek	Hirjak	25000.00
Jan	Pytel	28000.00
Zdenek	Stehule	20000.00
Lucie	Kubikova	18000.00
Radek	Hirjak	25000.00
	...	

(13 rows)

102 pracoviste

Tabulka	*Sloupce*
pracoviste	kod, popis
zamestnanci	id, jmeno, prijmeni, pracoviste_kod, vek
mzdy	id, vlozeno, zamestnanec_id, castka

Vypište jméno, příjmení a pracoviště všech zaměstnanců

Pavel	Stehule	Informatika
Radek	Hirjak	Konstrukce
Jan	Pytel	Informatika
Zdenek	Stehule	Vyroba
Lucie	Kubikova	Sekretariat
	...	

(6 rows)

103 pracoviste

Tabulka	*Sloupce*
pracoviste	kod, popis
zamestnanci	id, jmeno, prijmeni, pracoviste_kod, vek
mzdy	id, vlozeno, zamestnanec_id, castka

Vypište jméno, příjmení, částku, datum a popis všech zaměstnanců

Pavel	Stehule	30000.00	2007-01-01	Informatika
Radek	Hirjak	25000.00	2007-01-01	Konstrukce
Jan	Pytel	28000.00	2007-01-01	Informatika
Zdenek	Stehule	20000.00	2007-01-01	Vyroba
Lucie	Kubikova	18000.00	2007-01-01	Sekretariat
		...		

(23 rows)

802 vodocty

Tabulka	*Sloupce*
toky	id, jmeno
stanice	id, nazev
vodocty	tok_id, stanice_id, cas, vodocet_cm
limity_cm	tok_id, stanice_id, bdelost, pohotovost, ohrozeni
cleneni	tok_id, stanice_id, kraj, pobocka, povodi

Vypište vodní toky z povodí Berounky.

Berounka
Červený potok
Klabava
Litavka
Mže
...

(10 rows)

304 tramvaje

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, smer, poradi, zastavka_id

Které linky projíždějí stanicí Palmovka?

1
3
8
10
12
...

(8 rows)

606 letadla

Tabulka	*Sloupce*
dopravni_letadla	id, vyrobce, letadlo, dolet_km, kapacita, v_provozu_d
letecke_spolecnosti	id, spolecnost, zeme, svetadil, aliance, zalozeno
letecke_flotily	spolecnost_id, letadlo_id, pocet_letadel

Která letadla a kolik mají společnosti 'Air Algerie', 'Atlas Blue' a 'Royal Air Maroc'? Uveďte společnost, výrobce, letadlo a počet letadel.

Air Algerie	Airbus	A321	2
Air Algerie	Airbus	A330	5
Air Algerie	ATR	ATR-72	6
Air Algerie	Boeing	737	16
Air Algerie	Boeing	767	3
	...		

(14 rows)

207 filmy

Tabulka	*Sloupce*
filmy	id, rok, titul
umelci	id, jmeno
obsazeni	film_id, umelec_id, poradi
rezie	film_id, umelec_id

Jaké je herecké obsazení filmu ’Dům u jezera’

Keanu Reeves
Sandra Bullock
Willeke van Ammelroy
Mike Bacarella
Lynn Collins

(5 rows)

303 tramvaje

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, smer, poradi, zastavka_id

Vypište jména zastávek na lince 4.

Anděl
Bertramka
Blatiny
Čechovo náměstí
Hlušičkova
...

(27 rows)

106 pracoviste

Tabulka	*Sloupce*
pracoviste	kod, popis
zamestnanci	id, jmeno, prijmeni, pracoviste_kod, vek
mzdy	id, vlozeno, zamestnanec_id, castka

Vypište jméno, příjmení, částku všech zaměstnanců, jejichž příjem je vyšší než 20000, jejich příjmeni je Stehule.

Pavel	Stehule	30000.00
Pavel	Stehule	30000.00
Pavel	Stehule	30500.00
Pavel	Stehule	31000.00

(4 rows)

216 filmy

Tabulka	*Sloupce*
filmy	id, rok, titul
umelci	id, jmeno
obsazeni	film_id, umelec_id, poradi
rezie	film_id, umelec_id

Vypište všechny filmy a jejich režisery pro rok 2003.

Pán prstenů: Návrat krále	Peter Jackson
Kill Bill	Quentin Tarantino
Úsměv Mony Lisy	Mike Newell

(3 rows)

217 filmy

Tabulka	*Sloupce*
filmy	id, rok, titul
umelci	id, jmeno
obsazeni	film_id, umelec_id, poradi
rezie	film_id, umelec_id

Vypište všechny filmy a herce za rok 2003.

Pán prstenů: Návrat krále	Elijah Wood
Pán prstenů: Návrat krále	Viggo Mortensen
Pán prstenů: Návrat krále	Ian McKellen
Pán prstenů: Návrat krále	Sean Astin
Pán prstenů: Návrat krále	Orlando Bloom
...	

(13 rows)

212 filmy

Tabulka	*Sloupce*
filmy	id, rok, titul
umelci	id, jmeno
obsazeni	film_id, umelec_id, poradi
rezie	film_id, umelec_id

Ve kterých filmech měl hlavní roli Johnny Depp?

Piráti z Karibiku: Na konci světa
Piráti z Karibiku - Truhla mrtvého muže
Karlík a továrna na čokoládu
Libertin
Bojovník

(5 rows)

210 filmy

Tabulka	*Sloupce*
filmy	id, rok, titul
umelci	id, jmeno
obsazeni	film_id, umelec_id, poradi
rezie	film_id, umelec_id

Vypište všechny filmy a herce hlavních rolí (tj. těch, kteří mají pořadí 1) z roku 2003

Pán prstenů: Návrat krále	Elijah Wood
Kill Bill	Uma Thurman
Úsměv Mony Lisy	Julia Roberts

(3 rows)

903 premyslovci

Tabulka	*Sloupce*
premyslovci	id, jmeno, narozeni, umrti, otec, matka, rod

Uveďte děti Karla IV. Lucemburského.

Anne Luxemburg
Elisabeth Luxemburg
John Luxemburg
Václav IV. Lucemburský
Zikmund Lucemburský
...

(6 rows)

902 premyslovci

Tabulka	*Sloupce*
premyslovci	id, jmeno, narozeni, umrti, otec, matka, rod

Kdo byli rodiče Elišky Rejčky?

Přemysl II. Velkopolský
Richenza

(2 rows)

804 vodocty

Tabulka	*Sloupce*
toky	id, jmeno
stanice	id, nazev
vodocty	tok_id, stanice_id, cas, vodocet_cm
limity_cm	tok_id, stanice_id, bdelost, pohotovost, ohrozeni
cleneni	tok_id, stanice_id, kraj, pobocka, povodi

Které vodočty na Blanici překročily některý z limitů SPA (stav povodňové aktivity).
Uveďte název stanice, tři stupně SPA (stav povodňové aktivity), hodnotu vodočtu a čas.

Blanický mlýn	120	160	180	155	2007-09-06 19:00:00
Blanický mlýn	120	160	180	204	2007-09-07 02:00:00
Blanický mlýn	120	160	180	198	2007-09-07 03:00:00
Blanický mlýn	120	160	180	189	2007-09-07 04:00:00
Blanický mlýn	120	160	180	179	2007-09-07 05:00:00
			...		

(21 rows)

906 premyslovci

Tabulka	*Sloupce*
premyslovci	id, jmeno, narozeni, umrti, otec, matka, rod

Uveďte vnuky a vnučky Boleslava I.

Boleslav III.
Oldřich
Jaromír
Boleslav I. Chrabrý
Vladivoj

(5 rows)

905 premyslovci

Tabulka	*Sloupce*
premyslovci	id, jmeno, narozeni, umrti, otec, matka, rod

Kdo byli prarodiče knížete Václava? Uveďte jméno a rod.

Bořivoj Ludmila ze Pšova	Přemyslovci

(2 rows)

908 premyslovci

Tabulka	*Sloupce*
premyslovci	id, jmeno, narozeni, umrti, otec, matka, rod

Vyhledejte sourozence Oldřicha Brněnského. Uveďte jméno a rod.

Lítold Znojemský	Přemyslovci

(1 row)

Pohledy – virtuální tabulky

Pohledy – virtuální tabulky

Zjednodušená syntax:

```
CREATE VIEW jméno_pohledu [( seznam atributů )]
AS
dotaz SELECT ;
```

Příklad:

```
CREATE VIEW herecke_obsazeni ( film_id, titul,
                               umelec_id, jmeno, poradi )
AS
SELECT film_id, titul, umelec_id, jmeno, poradi
  FROM filmy
       JOIN
       obsazeni ON filmy.id = film_id
       JOIN
       umelci ON umelci.id = umelec_id;
```

Pohledy – virtuální tabulky

pohledy jsou databázové objekty, se kterými uživatel zachází stejně/obdobně jako s bázovými tabulkami. Pohledy ovšem nevytváří vlastní data (zvláštní samostatnou kategorií jsou *materializované pohledy*, v PostgreSQL jsou implementovány od verze 9.3).

bezpečnost pohledy mohou být použity pro omezení přístupu k vybraným sloupcům anebo řádkům bázové tabulky (přístup k bázové tabulce je omezen, povolen je pouze přístup přes pohled).

zjednodušení pohledy mohou odstínit uživatele od složitých dotazů

přejmenování pohledy umožňují přejmenování tabulek a atributů s komplikovanými jmény.

smysluplnost nevytvářejte zbytečné pohledy, které nejsou potřeba.

Pohledy – virtuální tabulky

Pohledy mohou být použity pro aktualizaci *“svých tabulek”* (*updatable views*), pokud

- pohled neobsahuje operátory `UNION`, `INTERSECT` nebo `EXCEPT`
- příkaz pohledu `SELECT` neobsahuje klauzule `DISTINCT`, `GROUP BY` nebo `HAVING`
- všechny attributy bázové tabulky, které nejsou zobrazovány pohledem mají definovány implicitní hodnoty
- pokud je pohled definován jako `JOIN`, můžeme aktualizovat jen jednu tabulku

Pohledy – příklad *updatable view*

Příklad: vytvoříme pohled, smažeme v něm vybrané záznamy a vložíme do něj jeden záznam.

```
CREATE VIEW demo_view AS
SELECT * FROM demo
WHERE num%2 = 0;

DELETE FROM demo_view
WHERE num > 2;

INSERT INTO demo_view (num, txt)
VALUES (8,'h');
```

Výsledný stav tabulky `demo` je uveden vpravo dole.

select * from demo

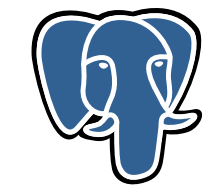
num	txt
1	a
2	b
3	c
4	d
5	e
6	f
7	g

select * from demo

num	txt
1	a
2	b
3	c
5	e
7	g
8	h

Materializované pohledy

Materializovaný pohled (materialized view) je „hybrid“ mezi tabulkou a klasickým pohledem. Povahou se podobá vyrovnávací paměti. Jde o fyzickou tabulku definovanou příkazem SELECT, což zvyšuje výkon databázového systému. Tento pohled přináší zrychlení dotazů, avšak duplikuje data v DB a zvyšuje režii UPDATE tabulky. [2]




```
CREATE MATERIALIZED VIEW table_name
    [ (column_name [, ...] ) ]
    [ WITH ( storage_parameter [= value] [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    AS query
    [ WITH [ NO ] DATA ]
```

[2] https://cs.wikipedia.org/wiki/Materializovaný_pohled, citováno 2019-10-05

Materializované pohledy – příklad

Pro tabulku `demo` definujeme materializovaný pohled `demo_view`, dále pak pomocný pohled `srovnani`, který zobrazí obě tabulky vedle sebe pomocí `FULL JOIN`.

```
 vnum | vtxt | dnum | dtxt
------+------+------+------
      |      |    1 | a
    2 | b    |    2 | b
      |      |    3 | c
    4 | d    |    4 | d
      |      |    5 | e
    6 | f    |    6 | f
      |      |    7 | g
```

```
CREATE MATERIALIZED VIEW demo_view
AS
SELECT * FROM demo WHERE  num%2 = 0;

CREATE VIEW srovnani
AS
SELECT demo_view.num AS vnum, demo_view.txt AS vtxt,
       demo.num AS dnum, demo.txt AS dtxt
FROM   demo_view FULL JOIN demo
       ON demo_view.num = demo.num;
```

Materializované pohledy – příklad (pokračování 1)

Počáteční stav

vnum	vtxt	dnum	dtxt
		1	a
2	b	2	b
		3	c
4	d	4	d
		5	e
6	f	6	f
		7	g

```
DELETE FROM demo WHERE num > 2;
INSERT INTO demo (num, txt) VALUES (8,'h');
```

Aktualizována tabulka `demo`, materializovaný pohled zůstává nezměněn.

vnum	vtxt	dnum	dtxt
		1	a
2	b	2	b
4	d		
6	f		
		8	h

Materializované pohledy – příklad (pokračování 2)

Počáteční stav

vnum	vtxt	dnum	dtxt
		1	a
2	b	2	b
4	d		
6	f		
		8	h

```
REFRESH MATERIALIZED VIEW demo_view;
```

Stav po aktualizaci materializovaného pohledu.

vnum	vtxt	dnum	dtxt
		1	a
2	b	2	b
8	h	8	h

Klauzule WITH / Common Table Expression (CTE)

Klauzule WITH

Klauzule `WITH` umožňuje definovat jednu nebo více dočasných pojmenovaných tabulek/relací. Anglicky se klauzuli `WITH` říká *common table expression (CTE)*.

Zjednodušená syntax:

```
WITH jméno_poddotazu_1 [( seznam atributů 1 )] AS (
    dotaz SELECT
    ),
    jméno_poddotazu_2 [( seznam atributů 2 )] AS (
    dotaz SELECT
)
SELECT ...
```

Pokud obsahuje klauzule `WITH` více poddotazů, oddělují se čárkou.

Klauzule WITH RECURSIVE

Varianta `WITH RECURSIVE` umožňuje rekurzivní definici poddotazů.

```
WITH RECURSIVE faktorial (n, f) AS (
      SELECT 0, 1
      UNION ALL
      SELECT n+1, (n+1)*f
      FROM faktorial
      WHERE n < 5
)
SELECT  n, f
FROM    faktorial;
```

```
 n |   f
---+-----
 0 |   1
 1 |   1
 2 |   2
 3 |   6
 4 |  24
 5 | 120
(6 rows)
```

Rekurzivní forma klauzule `WITH` obsahuje vždy *ukotvení (anchor)*, v našem případě `SELECT 0, 1`, následuje množinový operátor sjednocení (`UNION ALL`) a rekurzivní definice ostatních záznamů relace.

První select zajišťuje inicializaci, druhý se pak cyklicky opakuje.

A) Příklad WITH RECURSIVE

Mějme tabulku úředníků

```
CREATE TABLE urednici (
       id      integer,
       jmeno   text,
       vedouci integer
);
```

```
 id |   jmeno   | vedouci
----+-----------+---------
  1 | Marie     |    NULL
  2 | Jiří      |       1
  3 | Jan       |       1
  4 | Jana      |       6
  5 | Petr      |       4
  6 | Pavel     |       1
  7 | Jaroslav  |       5
  8 | František |      10
  9 | Věra      |      10
 10 | Eva       |       1
                .........
```

Vytvořte hierarchický strom podřízenosti vedoucí Marie. U každého úředníka uveďte úroveň jeho zařazení (0 vedoucí, 1 přímí podřízení, 2 podřízení podřízených atd.).

A) Příklad WITH RECURSIVE (pokračování)

```
WITH RECURSIVE hierarchie (id,jmeno,vedouci,uroven) AS (
      SELECT id, jmeno, vedouci,0
      FROM   urednici
      WHERE  jmeno='Marie'
      UNION ALL
      SELECT urednici.id, urednici.jmeno, urednici.vedouci,
             hierarchie.uroven+1
      FROM   urednici
      JOIN   hierarchie ON hierarchie.id = urednici.vedouci
)
SELECT * FROM hierarchie;
```

id	jmeno	vedouci	uroven
1	Marie		0
2	Jiří	1	1
3	Jan	1	1
6	Pavel	1	1
10	Eva	1	1
4	Jana	6	2
8	František	10	2
9	Věra	10	2
5	Petr	4	3
7	Jaroslav	5	4

B) Příklad WITH RECURSIVE

Najděte potomky Karla IV. Lucemburského.

```
 n  |  id  |       jmeno
----+------+-------------------
  0 | 1075 | Václav
  1 | 1111 | Vratislav I.
  1 | 1110 | Drahomíra
  2 |  163 | Bořivoj
  2 |  784 | Ludmila ze Pšova
  3 |  515 | Hostivít
  3 | 1032 | Slavibor
  4 |  968 | Neklan
  5 |  963 | Křesomysl
  6 |  974 | Vnislav
  7 |  975 | Vojen
  8 |  967 | Mnata
  9 |  969 | Nezamysl
 10 |  970 | Přemysl
 10 |  966 | Libuše
 11 |  964 | Krok
(16 rows)
```

B) Příklad WITH RECURSIVE (pokračování)

Najděte potomky Karla IV. Lucemburského.

```
 pokoleni |   id   |           jmeno
----------+--------+---------------------------
        0 |   652  | Karel IV. Lucemburský
        1 |   101  | Anne Luxemburg
        1 |   297  | Elisabeth Luxemburg
        1 |   628  | John Luxemburg
        1 |  1069  | Václav IV. Lucemburský
        1 |  1158  | Zikmund Lucemburský
        1 |   677  | Kateřina Luxemburg
        2 |   298  | Elisabeth Luxemburg
        2 |    88  | Alžběta Lucemburská
        3 |   959  | Ladislav Pohrobek
        3 |   102  | Anna Habsburg
        3 |   295  | Elisabeth Habsburg
(12 rows)
```

C) Příklad WITH RECURSIVE

Najděte předky svatého Václava.

```
WITH RECURSIVE predci (n, id, otec, matka, jmeno, rod) A
     SELECT 0, id, otec, matka, jmeno, rod
     FROM   premyslovci
     WHERE  jmeno='Václav'
     UNION ALL
     SELECT n+1, p.id, p.otec, p.matka, p.jmeno, p.rod
     FROM   premyslovci AS p
     JOIN   predci
            ON p.id IN (predci.otec, predci.matka)
     -- WHERE  n < 2 -- prarodice,
)
SELECT n, id, jmeno
FROM   predci;
```

C) Příklad WITH RECURSIVE (předci sv. Václava)

```
 n  |  id  |       jmeno
----+------+-------------------
  0 | 1075 | Václav
  1 | 1111 | Vratislav I.
  1 | 1110 | Drahomíra
  2 |  163 | Bořivoj
  2 |  784 | Ludmila ze Pšova
  3 |  515 | Hostivít
  3 | 1032 | Slavibor
  4 |  968 | Neklan
  5 |  963 | Křesomysl
  6 |  974 | Vnislav
  7 |  975 | Vojen
  8 |  967 | Mnata
  9 |  969 | Nezamysl
 10 |  970 | Přemysl
 10 |  966 | Libuše
 11 |  964 | Krok
(16 rows)
```

Agregační funkce, slučování dat a třídění

Agregační funkce

`COUNT(*)` počet řádků, které jsou výsledkem dotazu. Nepřipouští použití klíčového slova `DISTINCT`, počítají se i řádky s hodnotami `NULL`

`COUNT([DISTINCT]` *`výraz`*`)` počet `[`různých`]` hodnot

`SUM([DISTINCT]` *`výraz`*`)` součet `[`různých`]` hodnot

`AVG([DISTINCT]` *`výraz`*`)` průměr `[`různých`]` hodnot

`MAX(`*`výraz`*`)` maximum z `[`různých`]` hodnot

`MIN(`*`výraz`*`)` minimum z `[`různých`]` hodnot

S výjimkou funkce `COUNT(*)` ignorují všechny ostatní uvedené funkce hodnoty `NULL`.

Agregační funkce (příklady)

Funkce `COUNT(*)` vrací počet řádků, které by byly výsledkem dotazu `SELECT * FROM ...`

```
SELECT COUNT(*) AS počet
  FROM rybniky;
```

počet
18

V tabulce *rybníky* není známa hloubka rybníku Koclířov.

```
SELECT COUNT(rybnik)     AS pocet_rybniku,
       COUNT(hloubka_m) AS znama_hloubka
  FROM rybniky;
```

pocet_rybniku	*znama_hloubka*
18	17

Agregační funkce (příklady)

Zastávky na lince 4.

zastavky
Anděl
Anděl
Bertramka
Bertramka
Blatiny
Blatiny
Čechovo náměstí
Čechovo náměstí
Hlušičkova
Hlušičkova
Hotel Golf
Hotel Golf
I. P. Pavlova
I. P. Pavlova
Jana Masaryka
Jana Masaryka
Karlovo náměstí
Karlovo náměstí

```
SELECT COUNT(zastavka) AS count,
       COUNT(DISTINCT zastavka)
                     AS distinct
  FROM linky
       JOIN
       zastavky
       ON zastavka_id=id AND smer=1
 WHERE linka=4;
```

count	*distinct*
27	26

Agregační funkce (příklady)

r	*h*
489	6.2
416	6
394	7
337	4.5
317	3.4
305	3
296	3
284	12
258	6
246	6
241	8.5
230	3
205	3
201	3
196	2
192	
161	3
160	2

rozloha v hektarech a hloubka rybníků v metrech

```
SELECT rozloha_ha AS r, hloubka_m AS h
  FROM rybniky;
```

příklad použití agregačních funkcí

```
SELECT SUM(rozloha_ha) AS celkova_plocha,
       AVG(hloubka_m)  AS premerna_hloubka
  FROM rybniky;
```

celkova_plocha	*prumerna_hloubka*
4928	4.79999994390151

Poznámka: v explicitním pojmenování sloupců bychom mohli používat diakritiku a další znaky jako mezery a podobně, SQL ovšem není nástrojem pro tvorbu *tiskových sestav*, taková praxe se nedoporučuje.

Funkce CAST, výraz CASE a některé další funkce

Přetypování (CAST)

Na výstupu z předchozího příkladu je průměrná hloubka špatně čitelná

celkova_plocha	*prumerna_hloubka*
4928	4.79999994390151

Pro lepší čitelnost můžeme použít funkci přetypování `CAST`

`CAST` `(` *`výraz`* **`AS`** *`datový_typ`*`[`*`délka`*`]` `)`

Například

```
SELECT SUM(rozloha_ha) AS celkova_plocha,
       CAST(AVG(hloubka_m) AS DECIMAL(4,1)) AS prumerna\
  FROM rybniky;
```

celkova_plocha	*prumerna_hloubka*
4928	4.8

CASE

```
CASE vstupní hodnota    -- jednoduchý podmíněný výraz
   WHEN hodnota 1 THEN výsledek 1
   WHEN hodnota 2 THEN výsledek 2
     ...
   WHEN hodnota n THEN výsledek n
   [ ELSE výsledek pro ostatní případy ]
END
```

```
CASE                          -- obecný podmíněný výraz
  WHEN podmínka 1 THEN výsledek 1
  WHEN podmínka 2 THEN výsledek 2
    ...
  WHEN podmínka n THEN výsledek n
  [ ELSE výsledek pro ostatní případy ]
END
```

CASE příklad - výpočet studijního průměru

```
DROP TABLE IF EXISTS klasifikace CASCADE;

CREATE TABLE klasifikace (
       predmet varchar(12),
       znamka  char CHECK (znamka IN ('A', 'B', 'C', 'D', 'E', 'F')),
       jmeno   text NOT NULL
);

INSERT INTO klasifikace VALUES ('101MA1G','A','Zavadil');
INSERT INTO klasifikace VALUES ('152GP1', 'D','Zavadil');
INSERT INTO klasifikace VALUES ('101MA1G','A','Zavadil');
INSERT INTO klasifikace VALUES ('101MA1G','E','Strnad');
INSERT INTO klasifikace VALUES ('153GP1' ,'B','Rabas');
INSERT INTO klasifikace VALUES ('135GGO' ,'A','Zavadil');
INSERT INTO klasifikace VALUES ('153GGO',NULL,'Rabas');
INSERT INTO klasifikace VALUES ('101GP1' ,'C','Strnad');
INSERT INTO klasifikace VALUES ('101GGO' ,'F','Strnad');
INSERT INTO klasifikace VALUES ('151GDZ1','A','Strnad');
   ...
```

COALESCE

Funkce `COALESCE( par 1, par 2, ..., par n )` vrací hodnotu prvního parametru, který není `NULL`.

```
/* následující dotaz zobrazuje první známe telefonní číslo */

SELECT jmeno, prijmeni,
       COALESCE( do_prace,
                 domu,
                 mobil) AS "telefon"
       FROM telefony;
```

COALESCE (příklad)

```
SELECT * FROM telefony;
```

jmeno	*prijmeni*	*do prace*	*domu*	*mobil*
Rudolf	Cvach	233 533 233	543 333 234	753 344 228
Jan	Valenta	455 313 822	543 325 234	
Karel	Kulich		662 952 847	248 826 932
Petr	Zeman			888 436 962

```
SELECT jmeno, prijmeni,
       COALESCE(do_prace, domu, mobil) AS "telefon"
  FROM telefony;
```

jmeno	*prijmeni*	*telefon*
Rudolf	Cvach	233 533 233
Jan	Valenta	455 313 822
Karel	Kulich	662 952 847
Petr	Zeman	888 436 962

SPLIT_PART

Funkce `SPLIT_PART(`*`řetězec text, oddělovač text, pole int`*`)` vrací hodnotu *n*-tého pole vstupního řetězce

```
sqltutor=> select split_part('abc xyz', ' ', 1);
 split_part
------------
 abc
(1 row)

sqltutor=> select split_part('abc xyz', ' ', 2);
 split_part
------------
 xyz
(1 row)
```

EXTRACT

Funkce `EXTRACT(`*`pole FROM zdroj`*`)` vybírá dílčí složky z typů date/time. Prvním parametrem mohou být složky *day, decade, dow, doy, hour, minute, month, second, timezone, week, year* a další.

```
test=> select now();
                  now
-------------------------------
 2010-04-01 17:09:07.935512+02
(1 row)

test=> select extract(month from now());
 date_part
-----------
         4
(1 row)

test=> select extract(year from now());
 date_part
-----------
      2010
(1 row)
```

DATE

Fuknce `DATE(`*`timestamp`*`)` vrací hodnotu dne (datum) z typu `timestamp` (datum a čas ve vteřinách na 6 desetinných míst)ů

```
test=> select date(now()), now();
    date    |                now
------------+-------------------------------
 2011-03-17 | 2011-03-17 13:53:32.273402+01
(1 row)

test=>
```

Klauzule GROUP BY

Agregační funkce a klauzule GROUP BY

- Agregační funkce pracují vždy na jisté množině řádků a nemohou být proto kombinovány s jednotlivými řádkovými hodnotami.
- Všechny doposud uváděné příklady pracovaly s implicitní celou množinou řádků dané tabulky.
- Pro rozklad řádků tabulky na podmnožiny určené pro zpracování agregačními funkcemi slouží klauzule `GROUP BY`.
- Klauzule `GROUP BY` v příkazu `SELECT` následuje za klauzulemi `FROM` anebo `WHERE`
- Spolu s klauzulí `GROUP BY` můžeme použít klauzuli `HAVING`, která má v agregační klauzuli podobnou úlohu jako klauzule `WHERE` pro klauzuli `FROM`
- Sloupce podle kterých vytváříme rozklad, které jsou uvedeny v seznamu sloupců příkazu `SELECT`, musí být uvedeny v klauzuli `GROUP BY`

Klauzule GROUP BY v příkazu SELECT

```
    SELECT [ALL | DISTINCT] seznam sloupců
      FROM tabulka 1
          [ JOIN
            tabulka 2
            ON podmínka pro spojení tabulek ]
    [ WHERE restrikce ]
[ GROUP BY rozklad
   [ HAVING restrikce rozkladu ]];
```

Pořadí klauzulí `FROM`, `WHERE`, `GROUP BY` a `HAVING` v příkazu `SELECT` nelze měnit, některé z klauzulí ale mohou chybět.

Příkaz `SELECT` je vyhodnocen tak, že nejprve je provedena restrikce definovaná v klauzuli `WHERE` (pokud je přítomna), následuje vytvoření rozkladu definovaného v klauzuli `GROUP BY` a nakonec je na takto získaný rozklad uplatněna restrikce uvedená v klauzuli `HAVING`.

Příklady použití rozkladu GROUP BY

Počet rybníků v jednotlivých okresech a plocha největšího z nich.

```
SELECT okres, COUNT(*) AS pocet,
       MAX(rozloha_ha) AS max_plocha
  FROM rybniky
 GROUP BY okres;
```

okres	*pocet*	*max_plocha*
Jindřichův Hradec	9	489
Česká Lípa	1	284
Pardubice	1	160
Břeclav	1	296
Tábor	1	416
České Budějovice	3	394
Žďár nad Sázavou	1	205
Nymburk	1	258

Příklady použití rozkladu GROUP BY

Počet rybníků v jednotlivých okresech a plocha největšího z nich.
Neuvažujeme okresy, které mají v databázi jen jeden rybník.

```
SELECT okres, COUNT(*) AS počet,
       MAX(rozloha_ha) AS "max. plocha"
  FROM rybniky
 GROUP BY okres
HAVING COUNT(*) > 1;
```

okres	*počet*	*max. plocha*
Jindřichův Hradec	9	489
České Budějovice	3	394

Příklady použití rozkladu GROUP BY

Na kterých tramvajových linkách je čtyřicet a více zastávek?

```
SELECT linka, COUNT(zastavka_id) AS zastávky
  FROM linky
 WHERE smer=1
 GROUP BY linka
HAVING COUNT(zastavka_id) >= 40;
```

linka	*zastávky*
3	45
10	46
21	40

Příklady použití rozkladu GROUP BY

Kterými tramvajovými zastávkami projíždí více než sedm linek?

```
SELECT zastavka, COUNT(DISTINCT linka) AS "počet linek"
  FROM zastavky
       JOIN
       linky
       ON id = zastavka_id
 GROUP BY zastavka
HAVING COUNT(DISTINCT linka) > 7;
```

zastávka	*počet linek*
Anděl	8
Karlovo náměstí	9
Palackého náměstí	8
Palmovka	8

Klauzule ORDER BY, OFFSET a LIMIT

Setřídění výsledku dotazu (ORDER BY)

- klauzule `ORDER BY` umožňuje explicitně setřídit výsledek dotazu podle jednoho nebo více sloupců (lze uvádět aliasy)
- třídění se provádí v rámci jednotlivých sloupců (od prvního uvedeného do posledního).
- hodnoty `NULL` jsou vždy setříděny společně, standard ale neurčuje zda mají být setříděny jako první nebo jako poslední
- jednotlivé sloupce mohou být tříděny vzestupně (`ASC`) nebo sestupně (`DESC`)

```
SELECT rybnik, okres, povodi AS p
  FROM rybniky
 ORDER BY rybnik, okres DESC, p ASC;  -- p je alias
```

Omezení počtu řádků (OFFSET a LIMIT)

Spolu se tříděním se někdy používají klauzule `OFFSET` a `LIMIT`, které umožňují omezit počet řádků ve výsledku dotazu.

Obě klauzule se ale mohou používat i nezávisle na třídění. Mohou se používat společně, na jejich vzájemném pořadí pak nezáleží.

OFFSET udává počet *n* prvních řádků, které budou ve výsledku dotazu ignorovány (nebudou zobrazeny)

LIMIT udává počet řádků, které budou uvedeny ve výsledku dotazu (všechny ostatní budou ignorovány)

```
SELECT * FROM rybniky OFFSET 3;
SELECT * FROM rybniky LIMIT  4;
SELECT * FROM rybniky OFFSET 3 LIMIT 4;
```

Zjednodušená syntax příkazu SELECT

```
    SELECT [ALL | DISTINCT] seznam sloupců
      FROM tabulka 1
          [ JOIN
            tabulka 2
            ON podmínka pro spojení tabulek ]
    [ WHERE restrikce ]
[ GROUP BY rozklad
  [ HAVING restrikce rozkladu ]]
    [ UNION | INTERSECT | EXCEPT predikát ]
    [ ORDER BY třídění [ASC | DESC] ]
  [ OFFSET počet ]
    [ LIMIT počet ];
```

Klauzule `OFFSET` a `LIMIT` jsou nestandardním rozšířením SQL.

Příklad

Které linky mají na trase různé zastávky stejného jména? Vypište čísla linek a opakující se jména zastávek (pro každou linku a zastávku jen jednou). Seřaďte vzestupně podle čísel linek a omezte počet řádků ve výsledném výstupu na 10.

linka	*zastavka*
3	Palackého náměstí
3	Palmovka
3	Vltavská
4	Kotlářka
5	Vltavská
5	Výstaviště
6	Kotlářka
8	Palmovka
9	Nákladové nádraží Žižkov
11	Želivského

```
SELECT DISTINCT linka, zastavka
  FROM linky
       JOIN
       zastavky
       ON zastavka_id=id
 GROUP BY linka, smer, zastavka
HAVING COUNT(*) > 1
 ORDER BY linka, zastavka
 LIMIT 10;
```

Příklad

Jaký je nejvyšší počet linek projíždějících jednou zastávkou?

```
SELECT COUNT(DISTINCT linka) AS pocet
  FROM zastavky
       JOIN
       linky
       ON id = zastavka_id
 GROUP BY zastavka
 ORDER BY pocet DESC
LIMIT 1;
-- NESTANDARNI KONSTRUKCE, LEPE POMOCI DERIVOVANE TABULK
```

pocet
9

Agregační funkce, opakování a příklady

519 staty

Tabulka	*Sloupce*
staty	stat, region, populace, hdp

Kolik je států v jednotlivých regionech? Vypište vždy počet a region

13	South America
5	Micronesia
9	Middle Africa
17	Caribbean
5	Melanesia
	...

(20 rows)

410 unesco

Tabulka	*Sloupce*
unesco	pamatka, kategorie, zeme, region, zapis, doplneni

Kolik památek v seznamu světového dědictví je zapsáno v jednotlivých kategoriích. Pro každou kategorii uveďte počet.

kulturní	660
přírodní	166
smíšená	25

(3 rows)

412 unesco

Tabulka	*Sloupce*
unesco	pamatka, kategorie, zeme, region, zapis, doplneni

Které země mají v seznamu světového dědictví UNESCO zapsáno deset a více památek? Uveďte zemi a počet památek.

France	31
United Kingdom	27
Canada	14
Portugal	13
Czech Republic	12
...	

(20 rows)

218 filmy

Tabulka	*Sloupce*
filmy	id, rok, titul
umelci	id, jmeno
obsazeni	film_id, umelec_id, poradi
rezie	film_id, umelec_id

Které filmy měly více než jednoho režiséra? Vypište titul filmu a počet režisérů.

Lupiči paní domácí	2
Matrix	2
Invaze	2

(3 rows)

613 letadla

Tabulka	*Sloupce*
dopravni_letadla	id, vyrobce, letadlo, dolet_km, kapacita, v_provozu_d
letecke_spolecnosti	id, spolecnost, zeme, svetadil, aliance, zalozeno
letecke_flotily	spolecnost_id, letadlo_id, pocet_letadel

Kolik letadel je registrováno v databázi pro jednotlivé výrobce. Počítejte pouze letadla u kterých je známa kapacita. Výrobce, kteří nemají registrováno ani jedno letadlo se známou kapacitou neuvádějte.

Lockheed	1
Fokker	5
Embraer	4
Ilyushin	2
ATR	2
...	

(13 rows)

625 letadla

Tabulka	*Sloupce*
dopravni_letadla	id, vyrobce, letadlo, dolet_km, kapacita, v_provozu_
letecke_spolecnosti	id, spolecnost, zeme, svetadil, aliance, zalozeno
letecke_flotily	spolecnost_id, letadlo_id, pocet_letadel

Pro každého výrobce určete počet typů jeho letadel.

McDonnell Douglas	12
Boeing	10
BAe	9
Airbus	9
Embraer	6
...	

(22 rows)

158 nobelova_cena

Tabulka	*Sloupce*
laureati	rok, obor, laureat

Kdo získal Nobelovu cenu více než jedenkrát?

Office of the United Nations High Commissioner for Refugees
International Committee of the Red Cross
Linus Pauling
John Bardeen
Frederick Sanger
...

(6 rows)

521 staty

Tabulka	*Sloupce*
staty	stat, region, populace, hdp

Vypište regiony s celkovým počtem obyvatel větším než 100 milionů.

South America
Middle Africa
Eastern Europe
Northern Africa
Eastern Africa
...

(13 rows)

309 tramvaje

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, smer, poradi, zastavka_id

Na kterých tramvajových linkách je čtyřicet a více zastávek? Uveďte číslo linky, smer a počet zastávek. Poznámka: počet zastávek se na jedné lince může lišit pro oba směry.

10	2	49
22	2	41
3	1	45
10	1	46
3	2	46
	...	

(6 rows)

310 tramvaje

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, smer, poradi, zastavka_id

Jaký je nejvyšší počet linek projíždějících jednou zastávkou?

9

(1 row)

609 letadla

Tabulka	*Sloupce*
dopravni_letadla	id, vyrobce, letadlo, dolet_km, kapacita, v_provozu_
letecke_spolecnosti	id, spolecnost, zeme, svetadil, aliance, zalozeno
letecke_flotily	spolecnost_id, letadlo_id, pocet_letadel

Kolik je (registrováno v databázi) leteckých společností a jakou mají celkovou přepravní kapacitu? Uveďte po jednotlivých světadílech.

Afrika	12	24029
Asie	21	185801
Austrálie	30	82040
Evropa	114	352952
Jižní Amerika	11	27679
...		

(6 rows)

108 pracoviste

Tabulka	*Sloupce*
pracoviste	kod, popis
zamestnanci	id, jmeno, prijmeni, pracoviste_kod, vek
mzdy	id, vlozeno, zamestnanec_id, castka

Vypište, kolik zaměstnanců je na každém pracovišti

2	Sekretariat
0	Provoz
1	Vyroba
1	Konstrukce
2	Informatika

(5 rows)

413 unesco

Tabulka	*Sloupce*
unesco	pamatka, kategorie, zeme, region, zapis, doplneni

Které země Latinské Ameriky ('Latin America') mají v seznamu světového dědictví UNESCO zapsáno alespoň pět památek? Uveďte vždy zemi a počet.

Mexico	27
Brazil	17
Peru	10
Cuba	8
Argentina	8
...	

(9 rows)

615 letadla

Tabulka	*Sloupce*
dopravni_letadla	id, vyrobce, letadlo, dolet_km, kapacita, v_provozu_d
letecke_spolecnosti	id, spolecnost, zeme, svetadil, aliance, zalozeno
letecke_flotily	spolecnost_id, letadlo_id, pocet_letadel

Která aliance má největší celkovou přepravní kapacitu?

SkyTeam	164445

(1 row)

414 unesco

Tabulka	*Sloupce*
unesco	pamatka, kategorie, zeme, region, zapis, doplneni

Která země má v seznamu světového dědictví UNESCO zapsáno nejvíce památek a kolik?

Italy	41

(1 row)

214 filmy

Tabulka	*Sloupce*
filmy	id, rok, titul
umelci	id, jmeno
obsazeni	film_id, umelec_id, poradi
rezie	film_id, umelec_id

Vypište filmy z roku 2006 (tj. počtu herců uvedených v databázi). Uveďte počet herců a titul filmu.

2	Elizabeth: The Golden Age
5	Rituál
5	Piráti z Karibiku - Truhla mrtvého muže
5	Šifra mistra Leonarda
5	Goyovy přízraky
	...

(15 rows)

305 tramvaje

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, smer, poradi, zastavka_id

Kterými zastávkami projíždí více než 6 linek. Vypište jména zastávek a počet linek, které jimi projíždí. Poznámka: na některých linkách jsou dvě zastávky stejného jména, linku musíte ale započítat jen jednou.

Moráň	7
Anděl	8
Palackého náměstí	8
Palmovka	8
Karlovo náměstí	9

(5 rows)

512 staty

Tabulka	*Sloupce*
staty	stat, region, populace, hdp

Najděte země z regionů, u kterých je celková populace regionu menší než 25 milionů. Vypište zemi, region a počet obyvatel

Solomon Islands	Melanesia	478000
Fiji	Melanesia	848000
French Polynesia	Polynesia	257000
Kiribati	Micronesia	84000
Nauru	Micronesia	10000
	...	

(14 rows)

607 letadla

Tabulka	*Sloupce*
dopravni_letadla	id, vyrobce, letadlo, dolet_km, kapacita, v_provozu_d
letecke_spolecnosti	id, spolecnost, zeme, svetadil, aliance, zalozeno
letecke_flotily	spolecnost_id, letadlo_id, pocet_letadel

Které letecké společnosti mají sto a více letadel. Uveďte společnost, zemi a celkový počet letadel dané společnosti.

Air France	Francie	392
Lufthansa	Německo	243
Ryanair	Irsko	137
Qantas	Austrálie	123
Korean Air	Jižní Korea	123
	...	

(7 rows)

213 filmy

Tabulka	*Sloupce*
filmy	id, rok, titul
umelci	id, jmeno
obsazeni	film_id, umelec_id, poradi
rezie	film_id, umelec_id

Kteří herci hráli alespoň pětkrát v hlavní roli?

Bruce Willis
Tom Hanks
Mel Gibson
Johnny Depp

(4 rows)

306 tramvaje

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, smer, poradi, zastavka_id

Kterou zastávkou, resp. kterymi zastávkami, projíždí nejvíc linek?

Karlovo náměstí

(1 row)

159 nobelova_cena

Tabulka	*Sloupce*
laureati	rok, obor, laureat

Kdo získal Nobelovu cenu více než jedenkrát? Uveďte rok, obor a jméno laureáta.

1981	Mír	Office of the United Nations High Commissioner for Refu
1980	Chemie	Frederick Sanger
1972	Fyzika	John Bardeen
1963	Mír	International Committee of the Red Cross
1962	Mír	Linus Pauling
		...

(13 rows)

415 unesco

Tabulka	*Sloupce*
unesco	pamatka, kategorie, zeme, region, zapis, doplneni

Které památky ze seznamu světového dědictví UNESCO překračují hranice jedné země? Tj. které památky se týkají více zemí a kolika?

Uvs Nuur Basin	2
Primeval Beech Forests of the Carpathians	2
Mount Nimba Strict Nature Reserve	2
Belfries of Belgium and France	2
Jesuit Missions of the Guaranis	2
...	

(20 rows)

520 staty

Tabulka	*Sloupce*
staty	stat, region, populace, hdp

Pro každý region vypište jeho označení a počet státu s počtem obyvatel větším než 10 milionů. Poznámka: uveďte i regiony s nulovým počtem států.

South America	7
Micronesia	0
Middle Africa	3
Caribbean	1
Melanesia	0
...	

(20 rows)

612 letadla

Tabulka	*Sloupce*
dopravni_letadla	id, vyrobce, letadlo, dolet_km, kapacita, v_provozu_d
letecke_spolecnosti	id, spolecnost, zeme, svetadil, aliance, zalozeno
letecke_flotily	spolecnost_id, letadlo_id, pocet_letadel

U kterých společností není známa kapacita ani jednoho letadla?
Uveďte společnost, zemi a světadíl.

Jeju Air	Jižní Korea	Asie
Berjaya Air	Malajsie	Asie
Eastern Australia Airlines	Austrálie	Austrálie
Maroomba Airlines	Austrálie	Austrálie
Queensland Regional Airlines	Austrálie	Austrálie
...		

(37 rows)

611 letadla

Tabulka	*Sloupce*
dopravni_letadla	id, vyrobce, letadlo, dolet_km, kapacita, v_provozu_d
letecke_spolecnosti	id, spolecnost, zeme, svetadil, aliance, zalozeno
letecke_flotily	spolecnost_id, letadlo_id, pocet_letadel

Které jsou dvě největší letecké společnosti (tj. které mají největší celkovou kapacitu všech letadel)? Poznámka: kapacita není známa u všech letadel. Uveďte společnost a celkovou kapacitu.

Air France	83425
Lufthansa	62016

(2 rows)

513 staty

Tabulka	*Sloupce*
staty	stat, region, populace, hdp

Najděte v každém regionu zemi s nejvyšším počtem obyvatel.
Uveďte stát, region a populaci.

Brazil	South America	186405000
China	Eastern Asia	1315844000
Cuba	Caribbean	11269000
Ethiopia	Eastern Africa	77431000
French Polynesia	Polynesia	257000
	...	

(20 rows)

211 filmy

Tabulka	*Sloupce*
filmy	id, rok, titul
umelci	id, jmeno
obsazeni	film_id, umelec_id, poradi
rezie	film_id, umelec_id

Ve kterém roce natočil Tom Hanks nejvíc filmů? Vypište rok a počet filmů.

2004	3

(1 row)

621 letadla

Tabulka	*Sloupce*
dopravni_letadla	id, vyrobce, letadlo, dolet_km, kapacita, v_provozu_
letecke_spolecnosti	id, spolecnost, zeme, svetadil, aliance, zalozeno
letecke_flotily	spolecnost_id, letadlo_id, pocet_letadel

Uveďte všechny německé letecké společnosti a kolik mají Airbusů.

Air Berlin	2
WDL Aviation	0
Blue Wings	1
LTU International	3
Contact Air	0
...	

(20 rows)

311 tramvaje

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, smer, poradi, zastavka_id

Které zastávky na tramvajových linkách jsou konečné (tj. zastávky, které mají pořadí 1 nebo maximální pořadí na dané lince)? Uveďte číslo linky a konečnou zastávku. Poznámka: na některých linkách se konečné liší pro oba směry.

1	Petřiny
1	Spojovací
2	Červený Vrch
2	Petřiny
3	Lehovec
	...

(53 rows)

803 vodocty

Tabulka	*Sloupce*
toky	id, jmeno
stanice	id, nazev
vodocty	tok_id, stanice_id, cas, vodocet_cm
limity_cm	tok_id, stanice_id, bdelost, pohotovost, ohrozeni
cleneni	tok_id, stanice_id, kraj, pobocka, povodi

Jaké jsou denní průměry vodočtů na Dyji?
Uveďte název stanice, střední čas měření (průměr maximmálního a minimálního času v daném dni) a průměrnou denní hodnotu vodočtu na stanici (zaokrouhlete na celé centimetry).

Podhradí nad Dyjí	2007-09-05 17:00:00	37
Podhradí nad Dyjí	2007-09-06 14:00:00	68
Podhradí nad Dyjí	2007-09-07 11:30:00	140
Podhradí nad Dyjí	2007-09-08 11:30:00	153
Podhradí nad Dyjí	2007-09-09 14:30:00	102
	...	

(31 rows)

909 premyslovci

Tabulka	*Sloupce*
premyslovci	id, jmeno, narozeni, umrti, otec, matka, rod

Kolik dětí měli Přemyslovci? Uveďtě vždy jméno a počet dětí (u bezdětných uvádějte nulu).

Hostivít	1
Vratislav II.	5
Jindřich	1
Konrád III. Ota	0
Mnata	1
...	

(62 rows)

160 nobelova_cena

Tabulka	*Sloupce*
laureati	rok, obor, laureat

Kdo získal Nobelovu cenu ve více oborech? Uveďte rok, obor a jméno laureáta.

1962	Mír	Linus Pauling
1954	Chemie	Linus Pauling
1911	Chemie	Marie Curie
1903	Fyzika	Marie Curie

(4 rows)

806 vodocty

Tabulka	*Sloupce*
toky	id, jmeno
stanice	id, nazev
vodocty	tok_id, stanice_id, cas, vodocet_cm
limity_cm	tok_id, stanice_id, bdelost, pohotovost, ohrozeni
cleneni	tok_id, stanice_id, kraj, pobocka, povodi

Jaký je počet překročení limitů SPA za den (stav povodňové aktivity). Uveďte i nulové hodnoty, tj. dny ve kterých nebyly limity SPA překročeny.

2007-09-05	0
2007-09-06	18
2007-09-07	271
2007-09-08	112
2007-09-09	14
...	

(6 rows)

Poddotazy

Poddotazy

Příkaz `SELECT` se může vyskytovat v jiných příkazech `SELECT` nebo v příkazech `INSERT`, `UPDATE` a `DELETE`.

skalární poddotazy vrací jedinou hodnotu, vyskytují se v klauzulích WHERE, HAVING nebo i seznamu sloupců příkazu SELECT

tabulkové poddotazy které vrací více řádků, typicky s jedním sloupcem ve spojení s operátory IN, ANY, SOME EXISTS nebo ALL (poddotaz může obsahovat ale i více sloupců — *řádkové výrazy /row constructs*)

derivované tabulky (fiktivní tabulky) v klauzuli FROM, musí být pojmenovány AS *jméno*

nekorelované poddotazy nezávislé na vnějším dotazu

korelované poddotazy závislé na vnějším dotazu

Různé typy poddotazů v příkazu SELECT

```
SELECT sloupec 1, sloupec 2, ... (skalární poddotaz)
  FROM tabulka 1
         ...
       (derivovaná tabulka) AS jméno derivované tab.
         ...
 WHERE výraz = (skalární poddotaz)
    OR výraz IN (tabulkový poddotaz)
```

- poddotazy jsou vždy uzavřeny v závorkách
- v klauzuli `WHERE` operátory =, <, >, <=, >= a <> spolu s výrazy `[NOT] ANY`, `ALL`, `SOME` mohou být použity s tabulkovými poddotazy
- u korelovaných poddotazech můžeme v klauzuli `WHERE` použít operátor `[NOT] EXISTS`
- derivovaná tabulka musí být vždy pojmenována

Nad Primaskou - nekorelovaný poddotaz

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, poradi, zastavka_id

linka
7
19
26

Které linky projíždějí stanicí *Nad Primaskou*?

```
SELECT linka
  FROM linky
 WHERE zastavka_id IN
          (SELECT id
             FROM zastavky
            WHERE zastavka='Nad Primaskou');
```

Nad Primaskou - korelovaný poddotaz 1

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, poradi, zastavka_id

linka
7
19
26

Které linky projíždějí stanicí *Nad Primaskou*?

```
SELECT linka
  FROM linky
 WHERE 'Nad Primaskou' IN (SELECT zastavka
                             FROM zastavky
                            WHERE id = zastavka_id);
```

Nad Primaskou - korelovaný poddotaz 2

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, poradi, zastavka_id

linka
7
19
26

Které linky projíždějí stanicí *Nad Primaskou*?

```
SELECT linka
  FROM linky
 WHERE EXISTS (SELECT zastavka
                 FROM zastavky
                WHERE id = zastavka_id
                      AND zastavka='Nad Primaskou');
```

Nad Primaskou - JOIN 1

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, poradi, zastavka_id

linka
7
19
26

Které linky projíždějí stanicí *Nad Primaskou*?

```
SELECT linka
  FROM linky
       JOIN
       zastavky
       ON zastavka_id=id
 WHERE zastavka='Nad Primaskou';
```

Nad Primaskou - JOIN 2

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, poradi, zastavka_id

linka
7
19
26

Které linky projíždějí stanicí *Nad Primaskou*?

```
SELECT linka
  FROM linky
       JOIN
       zastavky
       ON zastavka_id=id
          AND zastavka='Nad Primaskou';
```

Poznámka: V tomto případě spojujeme do jedné podmínky definici spojení a restrikce, což není vždy možné (spojení LEFT JOIN a další).

Konečné tramvají - korelovaný poddotaz

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, poradi, zastavka_id

Které zastávky na tramvajových linkách jsou konečné (tj. zastávky, které mají pořadí 1 nebo maximální pořadí na dané lince)?

```
SELECT DISTINCT zastavka
 FROM zastavky
      JOIN
      linky AS A
   ON zastavka_id = id
WHERE poradi=1 OR poradi =
        (select max(poradi)
           from linky AS B
          where B.linka = A.linka);
```

Konečné tramvají - nekorelovaný poddotaz

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, poradi, zastavka_id

Které zastávky na tramvajových linkách jsou konečné (tj. zastávky, které mají pořadí 1 nebo maximální pořadí na dané lince)?

```
SELECT DISTINCT zastavka
  FROM zastavky
       JOIN
       linky AS A
       ON zastavka_id = id
       JOIN                    /* derivovaná tabulka */
       (SELECT linka, MAX(poradi) AS posledni
          FROM linky
         GROUP BY linka) AS B
       ON A.linka = B.linka
 WHERE poradi = 1 OR poradi = posledni;
```

Množinové operace UNION, INTERSECT a EXCEPT

```
dotaz1 UNION [ALL] dotaz2
dotaz1 INTERSECT [ALL] dotaz2
dotaz1 EXCEPT [ALL] dotaz2
```

UNION spojí výsledek dvou dotazů, implicitně přitom odstraňuje duplicity (podobně jako SELECT DISTINCT). Protože odstranění duplicit obvykle znamená třídění, alternativou může být použití `UNION ALL`, kdy výsledkem je spojení všech řádků obou dotazů. Obdobně i u dalších dvou operací.

INTERSECT vrací všechny řádky, které jsou přítomny v obou dotazech, implicitně jsou odstraněny duplicity (pokud není použita volba `ALL`).

EXCEPT vrací všechny řádky prvního dotazu, které nejsou obsaženy ve druhém dotazu, implicitně s odstraněním duplucit (pokud není použita varianta `EXCEPT ALL`).

Příklady různých typů spojení JOIN

CROSS JOIN

num	*name*
1	a
2	b
3	c

num	*value*
1	xxx
3	yyy
5	zzz

```
SELECT * FROM t1 CROSS JOIN t2;
```

num	*name*	*num*	*value*
1	a	1	xxx
1	a	3	yyy
1	a	5	zzz
2	b	1	xxx
2	b	3	yyy
2	b	5	zzz
3	c	1	xxx
3	c	3	yyy
3	c	5	zzz

INNER JOIN

num	*name*
1	a
2	b
3	c

num	*value*
1	xxx
3	yyy
5	zzz

```
SELECT * FROM t1 INNER JOIN t2 ON t1.num = t2.num;
```

num	*name*	*num*	*value*
1	a	1	xxx
3	c	3	yyy

INNER JOIN USING

num	*name*
1	a
2	b
3	c

num	*value*
1	xxx
3	yyy
5	zzz

```
SELECT * FROM t1 INNER JOIN t2 USING (num);
```

num	*name*	*value*
1	a	xxx
3	c	yyy

NATURAL INNER JOIN

num	*name*
1	a
2	b
3	c

num	*value*
1	xxx
3	yyy
5	zzz

```
SELECT * FROM t1 NATURAL INNER JOIN t2;
```

num	*name*	*value*
1	a	xxx
3	c	yyy

LEFT JOIN

num	*name*
1	a
2	b
3	c

num	*value*
1	xxx
3	yyy
5	zzz

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.num = t2.num;
```

num	*name*	*num*	*value*
1	a	1	xxx
2	b		
3	c	3	yyy

RIGHT JOIN

num	*name*
1	a
2	b
3	c

num	*value*
1	xxx
3	yyy
5	zzz

```
SELECT * FROM t1 RIGHT JOIN t2 ON t1.num = t2.num;
```

num	*name*	*num*	*value*
1	a	1	xxx
3	c	3	yyy
		5	zzz

FULL JOIN

num	*name*
1	a
2	b
3	c

num	*value*
1	xxx
3	yyy
5	zzz

```
SELECT * FROM t1 FULL JOIN t2 ON t1.num = t2.num;
```

num	*name*	*num*	*value*
1	a	1	xxx
2	b		
3	c	3	yyy
		5	zzz

Příklady

Přemyslovci (LEFT JOIN)

Tabulka	*Sloupce*
premyslovci	id, jmeno, narozeni, umrti, otec, matka, rod

Kolik dětí měli Přemyslovci?

Uveďte vždy jméno a počet dětí,
u bezdětných uvádějte nulu.

Poznámka: jméno není jednoznačným identifikátorem.

Richza Přemyslovna	0
Václav II.	4
Přemysl Otakar I.	3
Jindřich Vladislav	0
Přemysl	1
Vratislav I.	2
Spytihněv	0
Konrád II. Znojemský	2
Anna Přemyslovna	2
Vojen	2
...	

Přemyslovci (komplikovaně)

```
SELECT jmeno, coalesce(deti, 0)
  FROM premyslovci
       LEFT JOIN
       (
          SELECT otec AS rodic, count(id) AS deti
            FROM premyslovci
           GROUP BY otec
          UNION ALL
          SELECT matka AS rodic, count(id) AS deti
            FROM premyslovci
           GROUP BY matka
       ) AS rodice -- derivovaná tabulka musí být pojmenována
       ON id=rodic
 WHERE rod='Přemyslovci';
```

Přemyslovci (jednoduše)

Tabulka	*Sloupce*
premyslovci	id, jmeno, narozeni, umrti, otec, matka, rod

Kolik dětí měli Přemyslovci? Uveďte vždy jméno a počet dětí (u bezdětných uvádějte nulu).

```
SELECT rodice.jmeno, count(deti.jmeno)
  FROM premyslovci as rodice
       LEFT JOIN
       premyslovci as deti
       ON rodice.id IN (deti.otec, deti.matka)
 WHERE rodice.rod = 'Přemyslovci'
 GROUP BY rodice.id, rodice.jmeno;
```

Tramvajové linky (JOIN a LIMIT)

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, poradi, zastavka_id

Jaký je nejvyšší počet linek projíždějí jednou zastávkou?

```
SELECT COUNT(DISTINCT linka) AS pocet
 FROM zastavky
      JOIN
      linky
      ON id = zastavka_id
GROUP BY zastavka
ORDER BY pocet DESC
LIMIT 1;
```

Tramvajové linky (derivovaná tabulka)

Tabulka	*Sloupce*
zastavky	id, zastavka
linky	linka, poradi, zastavka_id

Jaký je nejvyšší počet linek projíždějí jednou zastávkou?

```
SELECT MAX(pocet)
  FROM (SELECT COUNT(distinct linka) AS pocet
          FROM linky
         GROUP BY zastavka_id) AS dt;
```

Státy (korelovaný a nekorelovaný poddotaz)

Tabulka	*Sloupce*
staty	stat, region, populace, hdp

Najděte v každém regionu zemi s nejvyšším počtem obyvatel.
Uveďte stát, region a populaci.

```
SELECT stat, region, populace
  FROM staty A
 WHERE populace = (SELECT MAX(populace) FROM staty B
                     WHERE A.region=B.region);

SELECT stat, region, populace
  FROM staty
 WHERE ROW(region, populace)   -- řádkový výraz (row construct)
            IN (SELECT region, MAX(populace)
                  FROM staty
                 GROUP BY region);
```

Návrh databáze

Návrh databáze

- každou základní entitu reprezentuje jedna bázová tabulka
- vlastnosti entit jsou reprezentovány atributy (sloupci tabulek)
- každá tabulka musí mít **primární klíč** (přirozený nebo umělý)
- relace 1:N (každá tabulka z "N" musí mít **cizí klíč** odkazující na primární klíč v tabulce "1")
- relace M:N musí být vyjádřeny pomocí asociačních tabulek

$\Longleftrightarrow$

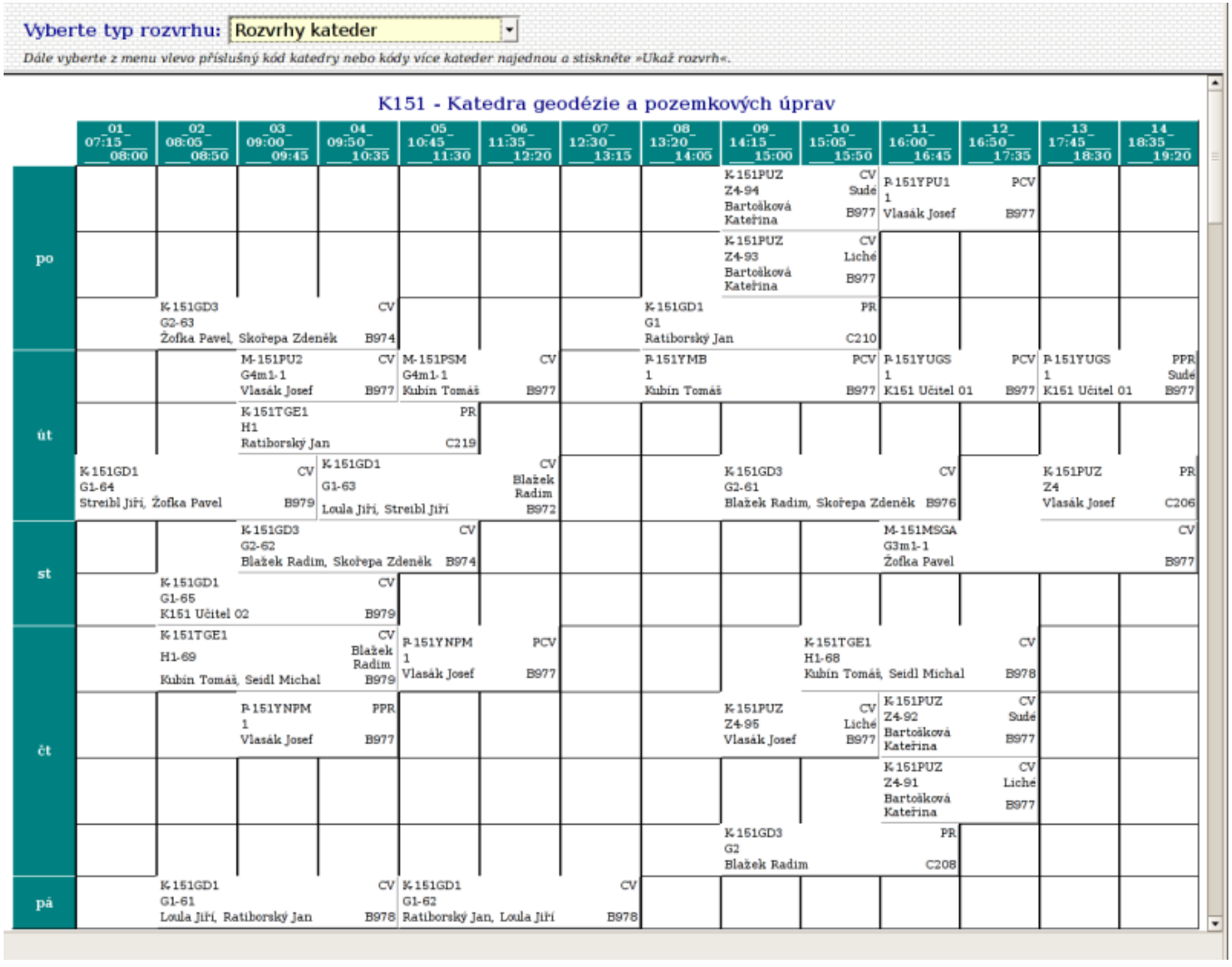
- iterační proces
- znalost modelovaného prostředí, tok informací, aktualizace dat
- datové struktury, modelování E-R relací
- normalizace tabulek
- ověřování návrhu na testovacích datech
- vyhodnocení testů, přehodnocení jednotlivých kroků návrhu, pokud je potřeba

Návrh databáze

Relační model činí návrh databáze intuitivní a snadný.

Pokud nejde o zcela triviální databázi, pak návrh databáze mohou s úspěchem realizovat pouze specializovaní odborníci.

Návrh databáze “rozvrh předmětů”



Vyberte typ rozvrhu: Rozvrhy kateder

Dále vyberte z menu vlevo příslušný kód katedry nebo kódy více kateder najednou a stiskněte »Ukaž rozvrh«.

K151 - Katedra geodézie a pozemkových úprav

	01 07:15 08:00	02 08:05 08:50	03 09:00 09:45	04 09:50 10:35	05 10:45 11:30	06 11:35 12:20	07 12:30 13:15	08 13:20 14:05	09 14:15 15:00	10 15:05 15:50	11 16:00 16:45	12 16:50 17:35	13 17:45 18:30	14 18:35 19:20
po									K151PUZ Z4-94 Bartošková Kateřina	CV Sudé B977	P151YPU1 1 Vlasák Josef	PCV B977		
									K151PUZ Z4-93 Bartošková Kateřina	CV Liché B977				
		K151GD3 G2-63 Žofka Pavel, Skořepa Zdeněk	CV B974					K151GD1 G1 Ratiborský Jan		PR C210				
út			M-151PU2 G4m1-1 Vlasák Josef	CV B977	M-151PSM G4m1-1 Kubín Tomáš	CV B977		P151YMB 1 Kubín Tomáš		PCV B977	P151YUGS 1 K151 Učitel 01	PCV B977	P151YUGS 1 K151 Učitel 01	PPR Sudé B977
			K151TGE1 H1 Ratiborský Jan	PR C219										
	K151GD1 G1-64 Streibl Jiří, Žofka Pavel	CV B979	K151GD1 G1-63 Loula Jiří, Streibl Jiří			CV Blažek Radim B972			K151GD3 G2-61 Blažek Radim, Skořepa Zdeněk	CV B976			K151PUZ Z4 Vlasák Josef	PR C206
st			K151GD3 G2-62 Blažek Radim, Skořepa Zdeněk	CV B974							M-151MSGA G3m1-1 Žofka Pavel			CV B977
		K151GD1 G1-65 K151 Učitel 02	CV B979											
čt		K151TGE1 H1-69 Kubín Tomáš, Seidl Michal	CV Blažek Radim B979		P151YNPM 1 Vlasák Josef	PCV B977				K151TGE1 H1-68 Kubín Tomáš, Seidl Michal		CV B978		
			P151YNPM 1 Vlasák Josef	PPR B977					K151PUZ Z4-95 Vlasák Josef	CV Liché B977	K151PUZ Z4-92 Bartošková Kateřina	CV Sudé B977		
											K151PUZ Z4-91 Bartošková Kateřina	CV Liché B977		
									K151GD3 G2 Blažek Radim			PR C208		
pá		K151GD1 G1-61 Loula Jiří, Ratiborský Jan	CV B978		K151GD1 G1-62 Ratiborský Jan, Loula Jiří	CV B978								

Entity a atributy

Sestavíme seznam entit (objektů), které figurují v rozvrhu a jejich základní atributy (vlastnosti)

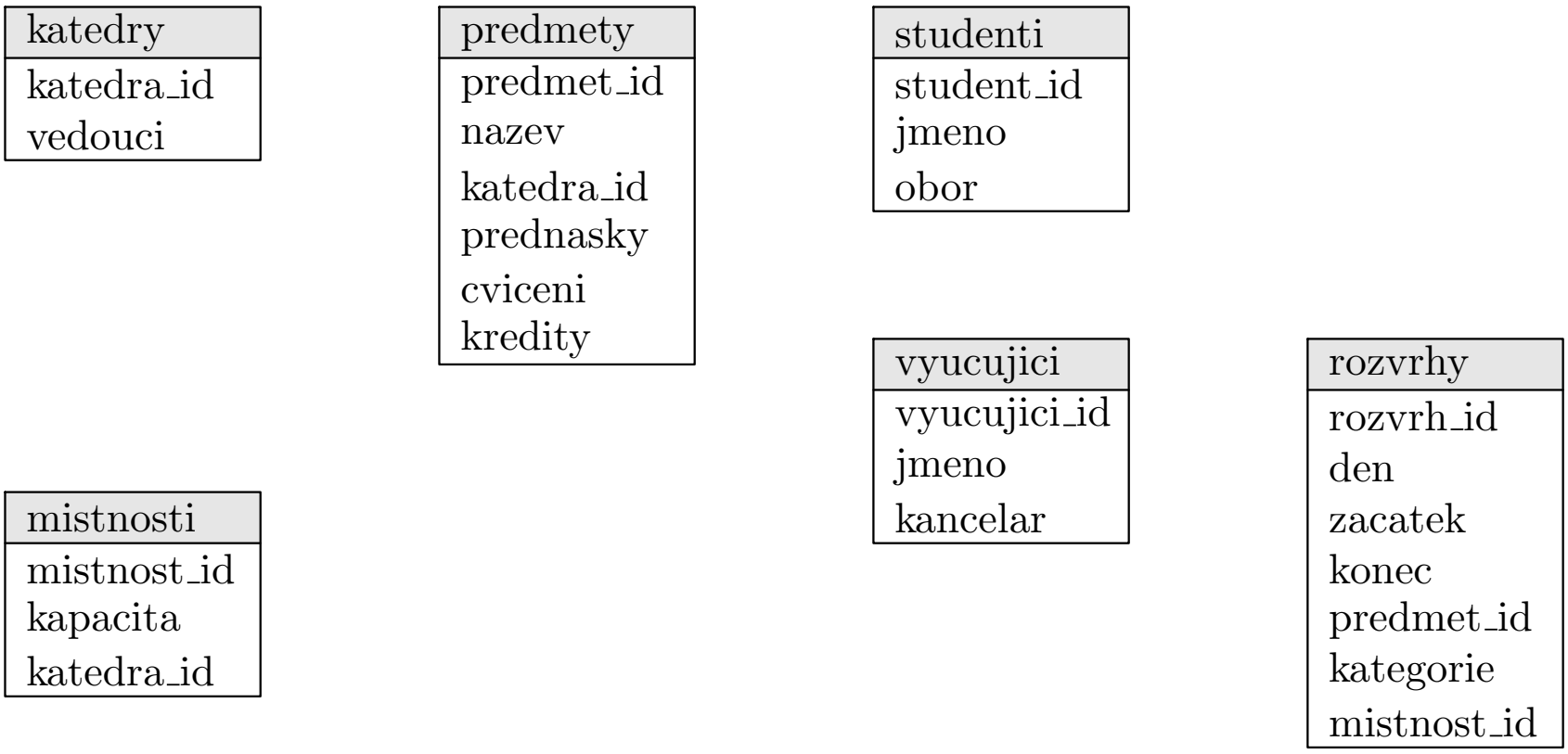
katedra	**číslo katedry**, název, vedoucí, telefon, . . .
předmět	**kód předmětu**, název, přednášky, cvičení, kredity, návaznost
místnost	**číslo místnosti**, kapacita, telefon
vyučující	**jméno**, místnost, email
student	**jméno**, obor, email, . . .

Primární klíče

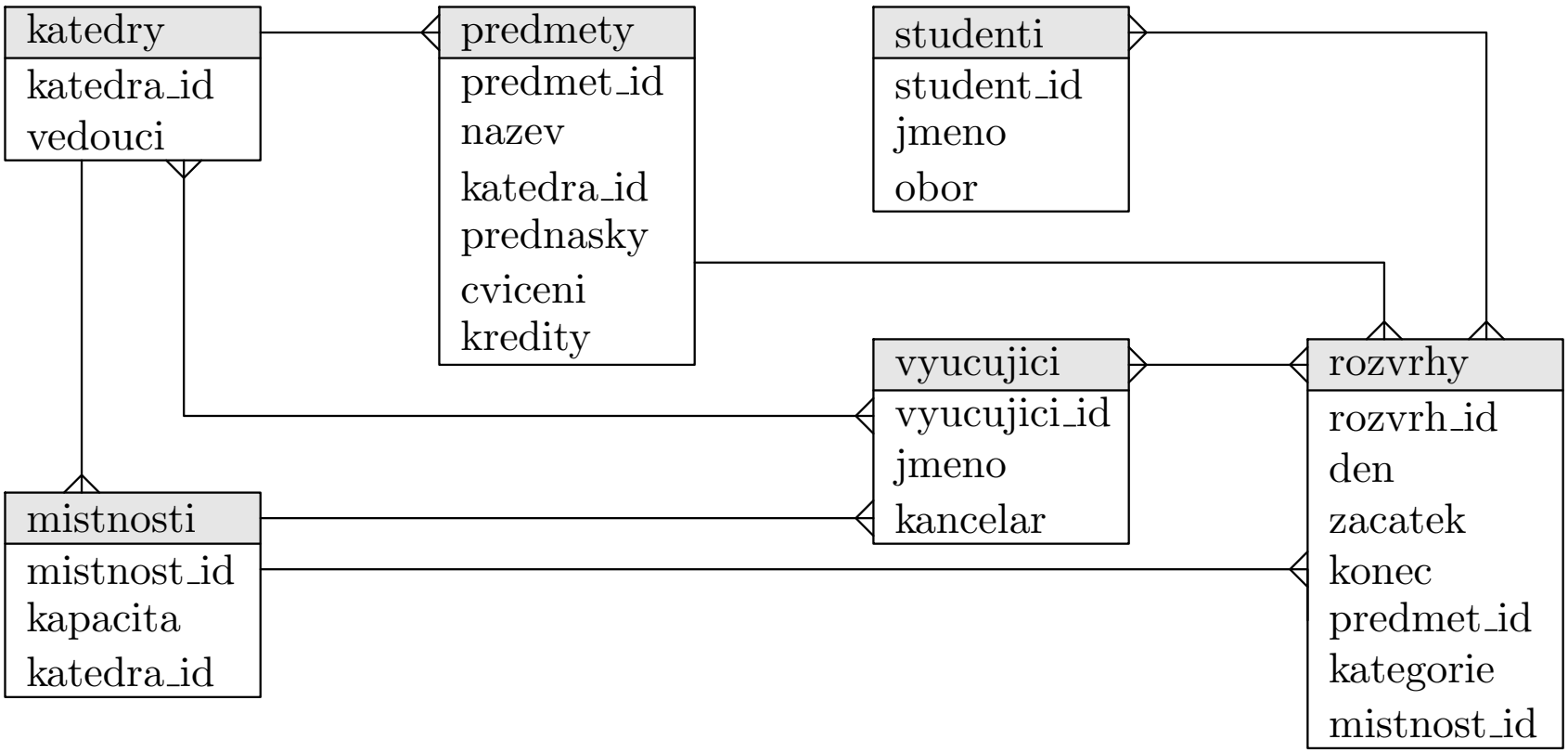
Číslo katedry, kód předmětu a číslo místnosti jsou přirozezenými kandidáty na **primární klíče**. Jméno (vyučujícího, studenta) je naproti tomu nevhodný atribut pro primární klíč, protože nemusí být jednoznačné. Pro entity bez přirozeného primárního klíče zavedeme umělé primární klíče.

ER modelování

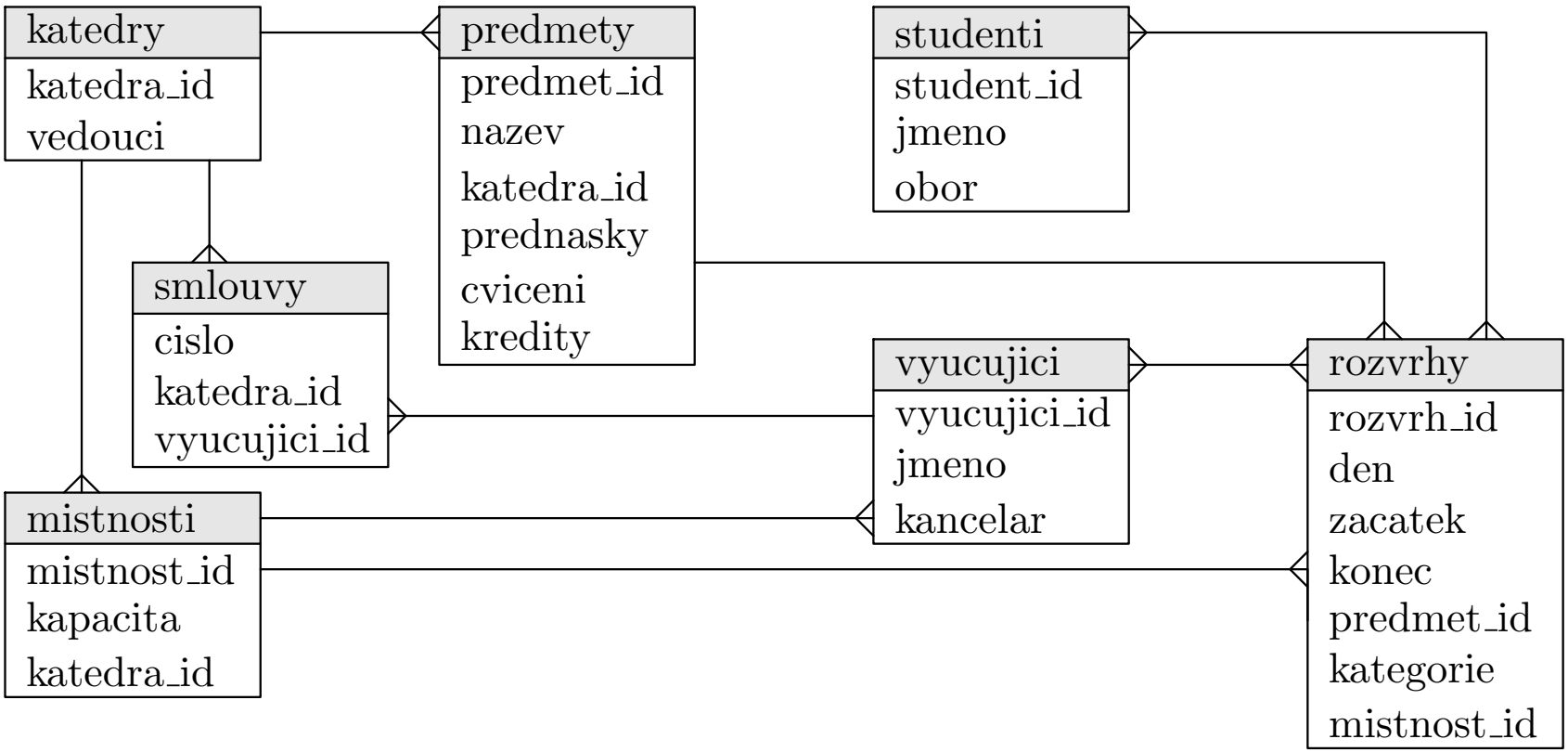
Entity a atributy



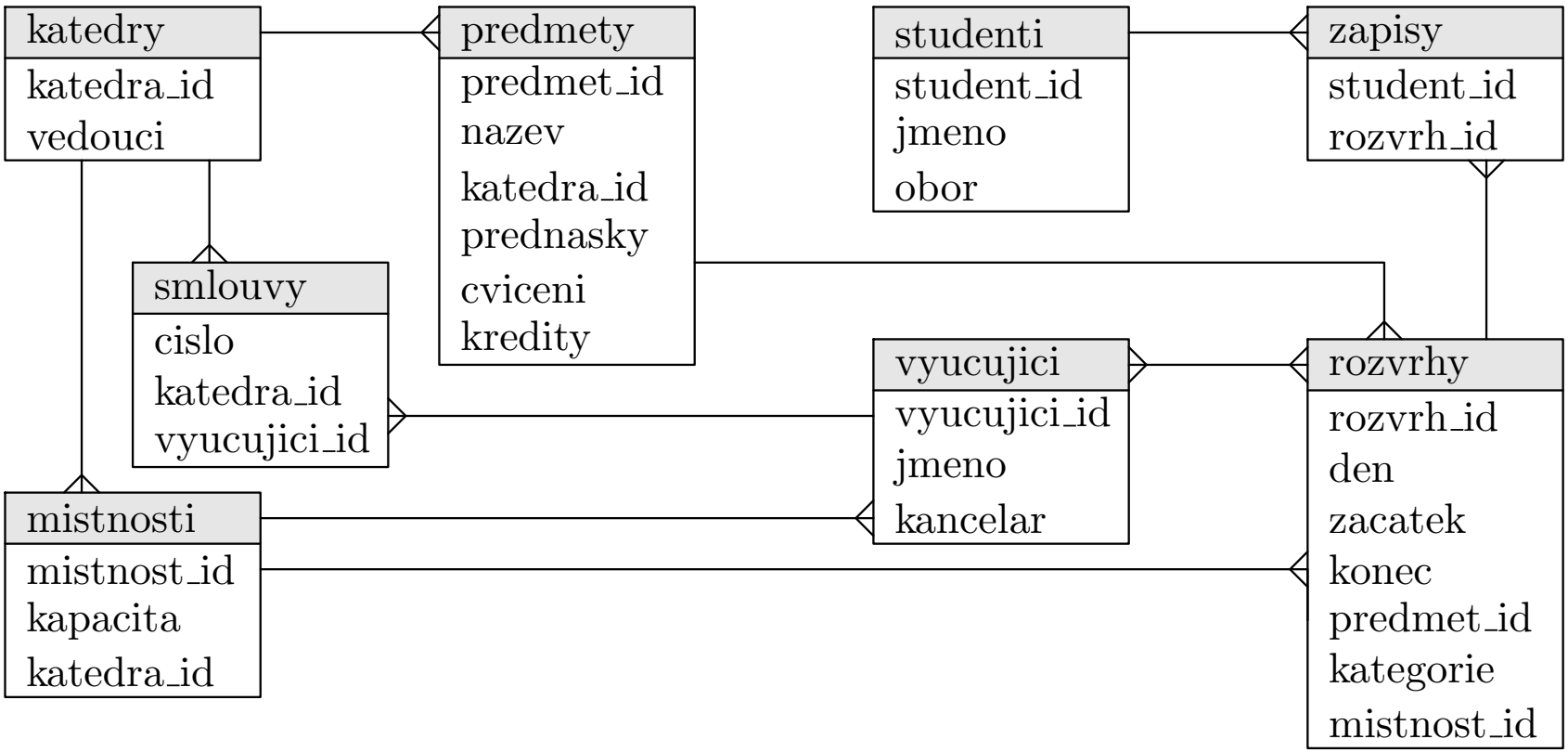
Entity a relace



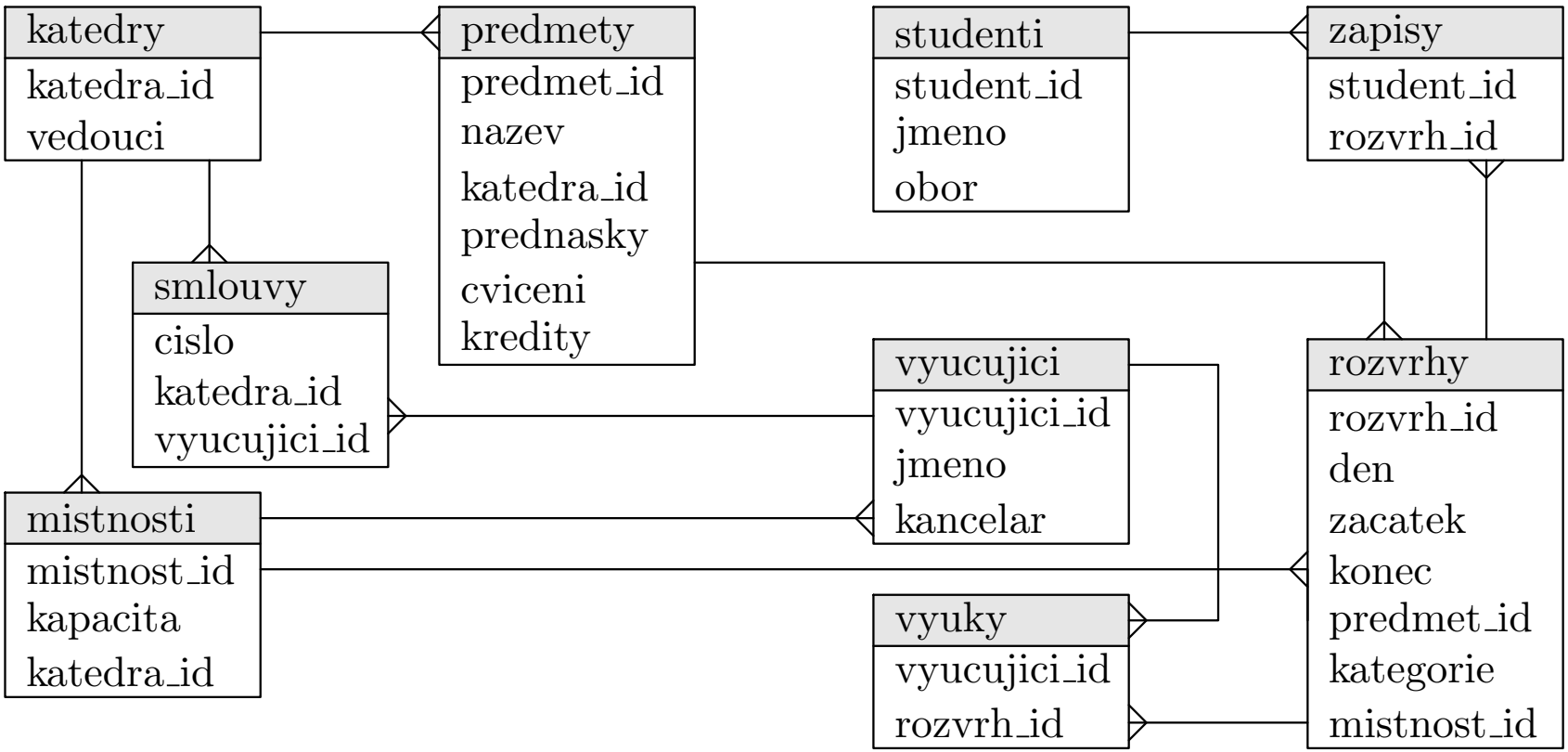
Katedry – vyučující $\Longrightarrow$ smlouvy



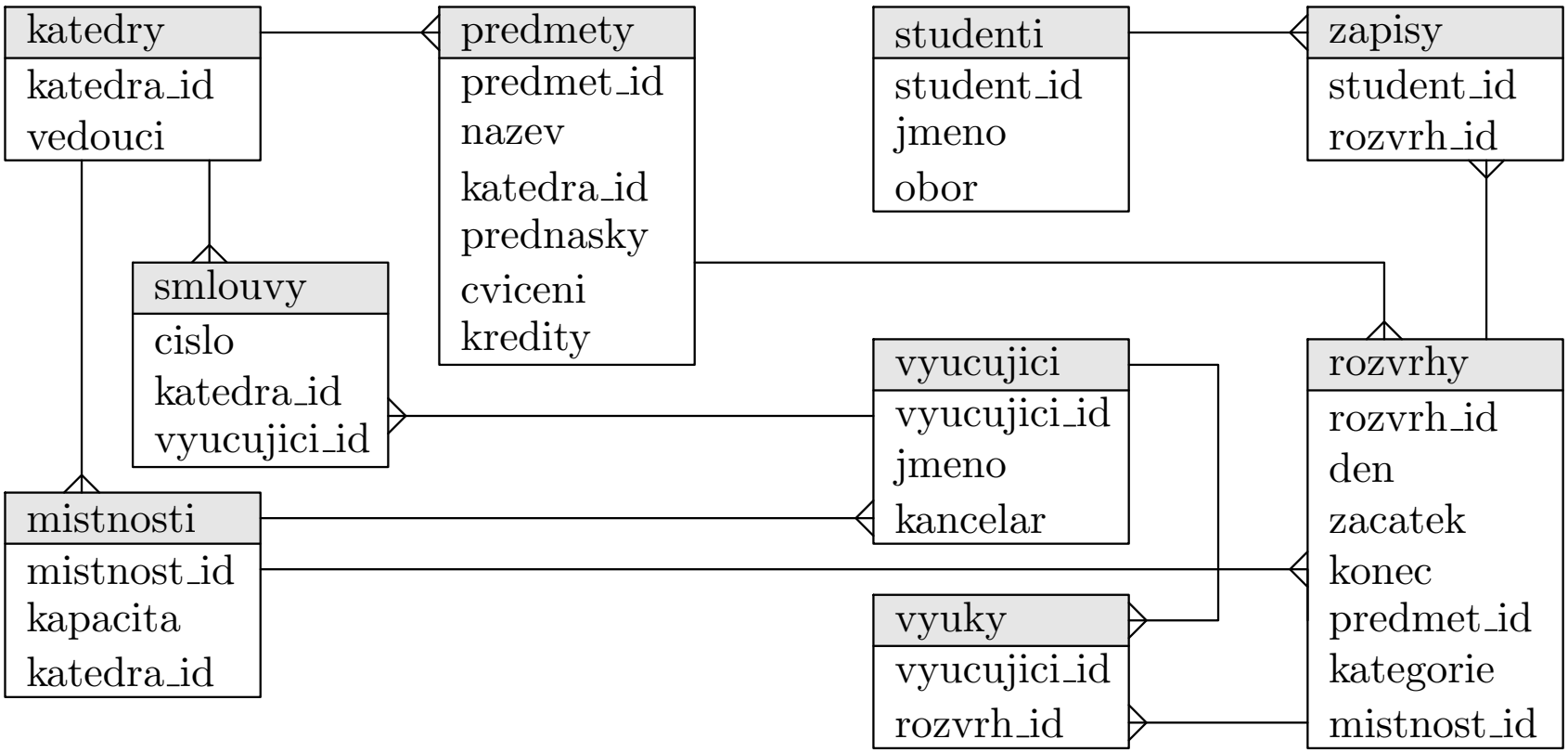
Studenti – rozvrhy $\Longrightarrow$ zápisy



Vyučující – rozvrhy $\Longrightarrow$ výuky



Entity a relace



Normalizace

Normalizace

- Normalizace je proces organizace tabulek databáze, tak aby správně modelovala realitu a zamezovala anomaliím spojeným s
  1. vkládáním, `INSERT`
  2. úpravami `UPDATE`
  3. a odstraňováním dat `DELETE`

  a eliminovala redundantní data (tj. aby se stejná data neukládala ve více sloupcích).
- Proces normalizace je postaven na hierarchii normálních forem
- Normální formy se běžně označují 1NF, 2NF, 3NF, BCNF, 4NF a 5NF.
- Nejčastěji se při normalizaci usiluje o dosažení 3NF.

Anomálie - zápis volitelných předmětů

Příklad podle Joe Celko's Data & Databases: Concepts in Practice

student	*předmět*	*vyučující*
Janata	Zpracování obrazových dat	Halounová
Jiránek	Zpracování obrazových dat	Halounová
Svobodová	Projekt digitální mapy	Kolář

INSERT Pokud chceme zapsat studenta na předmět Zpracování obrazových dat, musíme vědět, že vyučující je doc. Halounová nebo nemůžeme nový zázanam vložit.

UPDATE Pokud Jan Jiránek usoudí, že předmět Zpracování obrazových dat je příliš obtížný a rozhodne se změnit jej na Tenzorový počet, vytvoří řádek s neplatnými údaji ('Jiránek', 'Tenzorový počet', 'Halounová')

DELETE Pokud se doc. Kolář rozhodne odejít na jiné pracoviště, zrušením jeho záznamu zmizí informace, že Eliška Svobodová si zapsala předmět Projekt digitální mapy.

Kandidátní klíče

- Superklíč je množina atributů, které jednoznačně identifikují záznam. Triviálním superklíčem je množina všech atributů.
- Kandidátní klíč je minimální mmnožina atributů superklíče (neobsahuje nesouvisející informace), která identifikuje záznam.
- Libovolný kandidátní klíč můžeme zvolit za primární klíč.

Příklad: V tabulce

zápisy
student_id
student_jmeno
predmet_kod
predmet_nazev

jsou kandidátními klíči následující kombinace sloupců:

(student_id, predmet_kod)
(student_id, predmet_nazev)
(student_jmeno, predmet_kod)
(student_jmeno, predmet_nazev)

Bezztrátová dekompozice

Bezztrátová dekompozice je rozklad tabulky na více tabulek tak, aby bylo možno jejich spojením rekonstruovat původní tabulku.

Funkční závislosti

Funkční závislost $A \to B$ znamená, že atributy A jednoznačně určují hodnoty atributů B.

Armstrongovy axiomy
X, Y, Z jsou podmnožiny množiny atributů, XY označuje $X \cup Y$

axiomy

reflexivita: je-li $X \supseteq Y$, pak $X \to Y$

rozšíření: je-li $X \to Y$, pak $XZ \to YZ$ pro každou Z

transitivita: je-li $X \to Y$ a $Y \to Z$, pak $X \to Z$

pravidla

spojení: je-li $X \to Y$ a $X \to Z$, pak $X \to YZ$

dekompozice: je-li $X \to YZ$, pak $X \to Y$ a $X \to Z$

pseudotransitivita: je-li $A \to B$ a $CB \to D$, pak $AC \to D$

První normální forma (1NF)

Tabulka je v první normální formě, pokud neobsahuje žádné opakující se skupiny a průsečíkem každého sloupce a řádku je atomická (tj. z pohledu databáze dále nedělitelná hodnota) a pokud neobsahuje duplicitní záznamy.

Tabulka není v 1NF

prezident	*zvolen*
Havel	1993, 1998
Klaus	2003, 2008
Zeman	2013, 2018

Tabulka převedena do 1NF

prezident	*zvolen*
Havel	1993
Havel	1998
Klaus	2003
Klaus	2008
Zeman	2013
Zeman	2018

Druhá normální forma (2NF)

Tabulka je ve druhé normální formě, pokud

- je v první normální formě a
- každý neklíčový sloupec je závislý na každém celém kandidátním klíči.

Tabulka je vždy ve druhé normální formě, pokud primární klíč tvoří jediný sloupec nebo pokud tabulka neobsahuje žádné neklíčové sloupce (všechny sloupce tabulky tvoří primární klíč).

Následující tabulka s kandidátním klíčem (student, predmet) není ve 2NF, protože atribut *název předmětu* je dán kódem předmětu, tj. je závislý pouze na části klíče.

student	*predmet*	*znamka*	*nazev*
safraj11	101MA1G	A	Matematika 1G
vanovpe3	101MA1G	C	Matematika 1G
svobom35	152TCV2	E	Teorie pravděpodobnosti

Druhá normální forma (2NF)

Tabulku s kandidátním klíčem (student, predmet)

student	*predmet*	*znamka*	*nazev*
safraj11	101MA1G	A	Matematika 1G
vanovpe3	101MA1G	C	Matematika 1G
svobom35	152TCV2	E	Teorie pravděpodobnosti

převedeme do 2NF odstraněním redundatních dat, která převedeme do samostatné tabulky.

student	*predmet*	znamka
safraj11	101MA1G	A
vanovpe3	101MA1G	C
svobom35	152TCV2	E

predmet	*nazev*
101MA1G	Matematika 1G
152TCV2	Teorie pravděpodobnosti

Třetí normální forma (3NF)

Tabulka je ve třetí normální formě, pokud

- je ve druhé normální formě a
- každý neklíčový sloupec je netransitivně závislý na každém kandidátním klíči (žádný neklíčový sloupec není závislý na jiném neklíčovém sloupci).

Následující tabulka je ve 2NF, protože ale neexistuje přímá závislost mezi předmětem a kanceláří, není ve 3NF.

2NF

vyuka
predmet
vyucujici
kancelar

závislosti:

předmět $\rightarrow$ vyučující

vyučující $\rightarrow$ kancelář

3NF

vyuka
predmet
vyucujici

vyucujici
vyucujici
kancelar

- První normální forma (1NF) je triviální
- Druhá normální forma (2NF) je zastaralá
- Obvykle proto usilujeme o dosažení třetí normální formy (nebo vyšších)

Boyce-Coddova normální forma (BCNF)

Definice BCFN

Relace R je v BCNF tehdy a jen tehdy, když pro každou netriviální závislost X $\rightarrow$ Y, kde X a Y jsou množiny atributů a zároveň Y není podmnožinou X, platí, že X je nadmnožinou nějakého klíče, nebo X je klíčem relace R.

Jinak řečeno relace R je v BCNF tehdy a jen tehdy, když každý determinant funkční závislosti v relaci R je zároveň kandidátním klíčem relace R.[3]

[3] citováno z http://cs.wikipedia.org/wiki/Boyce-Coddova_normální_forma

BCNF vs. 3NF, Carlo Zaniolo 1982

Relace R je v BCNF tehdy a jen tehdy, když

Pro každou netriviální závislost A→B, platí, že A je superklíčem v R

Relace R je v 3NF tehdy a jen tehdy, když

Pro každou netriviální závislost A→B, platí, že A je superklíčem v R
nebo B je podmnožinou kandidátního klíče

1. Jestliže R je v BCNF $\longrightarrow$ je také v 3NF
2. Jestliže R je v 3NF $\longrightarrow$ může ale nemusí být v BCNF
3. Jestliže R není v 3NF $\longrightarrow$ není ani v BCNF

Normální formy - shrnutí

1NF Tabulka je v první normální formě, pokud průsečíkem každého sloupce a řádku je atomická hodnota (neobsahuje opakující se skupiny).

Každý atribut obsahuje pouze atomické hodnoty.

2NF Každý neklíčový sloupec je závislý na každém celém kandidátním klíči (žádný atribut nesmí být závislý jen na části PK). Tabulka je vždy ve druhé normální formě, pokud primární klíč tvoří jediný sloupec nebo pokud tabulka neobsahuje žádné neklíčové sloupce (všechny sloupce tabulky tvoří primární klíč).

Každý neklíčový atribut je plně závislý na primárním klíči.

3NF Každý neklíčový sloupec je netransitivně závislý na každém kandidátním klíči (žádný neklíčový sloupec není závislý na jiném neklíčovém sloupci).

Všechny neklíčové atributy musí být vzájemně nezávislé.

BCNF Relace R je v BCNF tehdy a jen tehdy, když každý determinant funkční závislosti v relaci R je zároveň kandidátním klíčem relace R.

Atributy, které jsou součástí primárního klíče, musí být vzájemně nezávislé.

1NF příklady

jmeno	prijmeni	adresa	mesto	PSC	telefon
Jan	Novák	Kraví hora 23	Brno	616 00	345 543 235, 753 552 565

Tabulka není v 1NF protože nemá primární klíč (jsou možné duplicitní záznamy), atribut telefon neobsahuje atomickou hodnotu (telefonni čísla oddělená čárkou).

Id	jmeno	prijmeni	adresa	mesto	PSC	telefon1	telefon
7	Jan	Novák	Kraví hora 23	Brno	616 00	345 543 235	753 55

Tabulka sice má primární klíč (Id), ale přesto není v 1NF. Zavedení dvou či více atributů nahrazujících seznam telefonních čísel sice vede na atomické hodnoty, je to ale *podvod* – kolik atributů pro telefonní čísla je nutné definovat, 2, 3, . . . ?

Id	Username	Port	IP_address	Domain_name
2159	cepek	pts/4	147.68.239.109	b869-01.fsv.cvut.cz

Tabulka je minimálně v 1NF, má primární klíč (Id) a data jsou atomická (neobsahují opakující se skupiny).

Příklad: převod z 2NF do 3NF

ISBN	titul	vydavatelstvi	adresa
80-247-0999-6	SQL	Grada	U Průhonu 22, Praha

Tabulka má jednoduchý primární klíč a je proto minimálně v 1NF. Mezi neklíčovými atributy ale existuje funkční závislost

$$\text{vydavatelství} \longrightarrow \text{adresa}$$

a je proto jen ve 2NF.

Do 3NF převedeme rozdělením na dvě tabulky

ISBN	titul	vydavatelstvi
80-247-0999-6	SQL	Grada

vydavatelstvi	adresa
Grada	U Průhonu 22, Praha

Závislosti: $\text{ISBN} \longrightarrow \text{titul}$

$\text{ISBN} \longrightarrow \text{vydavatelstvi}$

$\text{vydavatelství} \longrightarrow \text{adresa}$

BCNF poznámky

- pokud je relace ve 3NF a má jednoduchý klíč (jediný atribut), je zároveň i v BCNF
- relace, která je ve 3NF není v BCNF pokud platí tyto skutečnosti[4]:
  - v relaci existuje více kandidátních klíčů,
  - všechny kandidátní klíče jsou složené ze dvou nebo více atributů,
  - existuje takový atribut, který je společný pro všechny kandidátní klíče.

[4] zdroj http://cs.wikipedia.org/wiki/Boyce-Coddova_normální_forma

BCNF příklad ClientInterview[5]

clientNo	interviewDate	interviewTime	staffNo	roomNo
CR76	13-May-02	10.30	SG5	G101
CR56	13-May-02	12.00	SG5	G101
CR74	13-May-02	12.00	SG37	G102
CR56	1-Jul-02	10.30	SG5	G102

Závislosti (kandidátní klíč, není kandidátním klíčem):

- clientNo, interviewDate $\longrightarrow$ interviewTime, staffNo, roomNo
- staffNo, interviewDate, interviewTime $\longrightarrow$ clientNo
- roomNo, interviewDate, interviewNo $\longrightarrow$ clientNo, staffNo
- staffNo, interviewDate $\longrightarrow$ roomNo

clientNo	interviewDate	interviewTime	staffNo
CR76	13-May-02	10.30	SG5
CR56	13-May-02	12.00	SG5
CR74	13-May-02	12.00	SG37
CR56	1-Jul-02	10.30	SG5

staffNo	interviewDate	roomNo
SG5	13-May-02	G101
SG37	13-May-02	G102
SG5	1-Jul-02	G102

[5] zdroj neznámý

Koncerty — BCNF vs 3NF

datum	*město*	*koncert*	*místo_konání*
2013-12-20	Brno	Lucie Šoralová	Cafe Gracian
2013-12-06	Moravské Budějovice	Vladimír Mišík	Pogo Bar
2013-12-04	Praha	Support Lesbians	Palác Akropolis
2013-12-08	Praha	Vladimír Mišík	Palác Akropolis
2013-11-30	Moravské Budějovice	Skyline	Pogo Bar
2013-12-13	Brno	Vladimír Mišík	Cafe Gracian
2013-12-11	Brno	Support Lesbians	Semilaso
2013-12-14	Klatovy	Vladimír Mišík	Music Club U Košile

Závislosti

datum, *město*, *koncert* $\rightarrow$ *místo_konání*
místo_konání $\rightarrow$ *město*

3NF

město je podmnožinout kandidátního klíče {*datum*, *město*, *koncert*}

BCNF

město není superklíčem (nadmnožinou klíče)

Koncerty — převod do BCNF

datum	*koncert*	*místo_konání*
2013-12-20	Lucie Šoralová	Cafe Gracian
2013-12-06	Vladimír Mišík	Pogo Bar
2013-12-04	Support Lesbians	Palác Akropolis
2013-12-08	Vladimír Mišík	Palác Akropolis
2013-11-30	Skyline	Pogo Bar
2013-12-13	Vladimír Mišík	Cafe Gracian
2013-12-11	Support Lesbians	Semilaso
2013-12-14	Vladimír Mišík	Music Club U Košile

místo_konání	*město*
Cafe Gracian	Brno
Pogo Bar	Moravské Budějovice
Palác Akropolis	Praha
Semilaso	Brno
Music Club U Košile	Klatovy

Příklad – normalizace databáze maturantů

Nekorektní tabulka

maturanti
maturant_id
maturant_jmeno
skola_kod
skola_nazev
skola_adresa
predmety_id
predmety_nazev
predmety_znamky

- Tabulka maturantů je nekorektní, protože obsahuje opakující se skupiny.
- Zavedení vícenásobných atributů pro opakující se skupiny není řešení, ale *"podvod"* (kolik jich má být?).
- Tabulku převedeme do 1NF tak, že ji rozdělíme do samostatných tabulek podle souvisejících atributů a definujeme pro ně primární klíče (bezztrátová dekompozice).

100	Novák	800	Gymnázium	Plzeň	10, 11, 12, 13	ČJ, A, Matematika, Zeměpis	1, 2, 4, 2
101	Votočková	800	Gymnázium	Plzeň	10, 14, 15	ČJ, Sociologie, Politologie	4, 1, 1
300	Zuna	850	Obch. akad.	Praha	10, 11, 16	ČJ, A, Účetnictví	1, 1, 1
310	Palivec	850	Obch. akad.	Praha	10, 16, 17, 18	ČJ, Účetnictví, Excel, Word	1, 1, 1, 1
400	Kozel	900	Zahradnická	Louny	19, 20	Trávníky, Zalévání	2, 3
500	Prusák	900	Gymnázium	Praha	10, 21, 22	ČJ, Ruština, Ekonomie	1, 1, 1

1NF: Odstraňte opakující se skupiny

maturanti
maturant_id
maturant_jmeno
skola_kod
skola_nazev
skola_adresa

predmety
maturant_id
predmet_id
predmet_nazev
predmet_znamka

- Hodnoty sloupce predmet_nazev jsou závislé pouze na části klíče predmet_id a v tabulce *predmety* se zbytečně opakují (mnoho studentů má shodné maturitní předměty).
- Pokud atribut závisí pouze na části klíče, převedeme jej do samostatné tabulky (bezztrátová dekompozice). Tabulku takto převedeme do 2NF.

2NF: Odstraňte redundantní data

maturanti
maturant_id
maturant_jmeno
skola_kod
skola_nazev
skola_adresa

maturitni_predmety
maturant_id
predmet_id
predmet_znamka

predmety
predmet_id
predmet_nazev

- Název a adresa jsou v tabulce *maturanti* závislé na kódu školy.
- Pokud atributy nepřispívají k popisu klíče, převedeme je do samostatné tabulky bezztrátovou dekompozicí. Tabulku tak převedeme do 3NF.

3NF: Odstraňte sloupce, které nejsou závislé na klíči

maturanti
maturant_id
maturant_jmeno
skola_kod

maturitni_predmety
maturant_id
predmet_id
predmet_znamka

predmety
predmet_id
predmet_nazev

skoly
skola_kod
skola_nazev
skola_adresa

- Všechny tabulky jsou nyní ve třetí normální formě (3NF).

Indexy

Indexy

- Indexy jsou databázovými objekty, které vytvářejí pomocné datové struktury pro rychlý a efektivní přístup k záznamům tabulek. Bez indexů by tabulkové dotazy degenerovaly na sekvenční prohledávání všech záznamů tabulky.
  Dotazy, ve které by v klauzuli `WHERE` byly neindexované atibuty, by byly stejně neefektivní jako hledání v nesetříděném telefonním seznamu.
- Pokud je v tabulce definován primimární klíč, je zároveň vždy automaticky vytvořen i index.
- V tabulce je implicitně vytvořen index i pro každou deklaraci UNIQUE (pro jeden či více atributů).
- Indexy ale můžeme vytvářet i nad neklíčovými atributy, které přitom mohou obsahovat duplicitní hodnoty a hodnoty `NULL`.
- Nad tabulkou může být definováno více indexů. V mnoha případech indexy zabírají větší objem diskového prostoru než vlastní data. Správa indexů vždy představuje nezanedbatelnou režii, je třeba proto definovat je uváženě.

Indexy v databázovém systému PostgreSQL

- Indexy jsou nejčastěji implementovány jako vyvážené B-stromy, jsou specifické pro daný databázový systém a v následujícím textu se budeme zabývat indexy v databázovém systému PostgreSQL.
- Existuje celá řada typů indexů, typ můžeme explicitně volit v definici indexu (B-tree, Hash, GiST, GIN).
- Zjednodušená syntax definice indexu

```
CREATE INDEX jméno indexu ON tabulka (seznam atributů
```

- Stejně tak jako jiné databázové objekty můžeme index odstranit příkazem

```
DROP INDEX jméno indexu;
```

- V příkladech práce s indexy budeme pro generování dat používat pomocné funkce postgresu

generate_series(d,h,k) generuje posloupnost čísel od dolní meze d do horní meze h s volitelným krokem k.

random() náhodné číslo z intervali $< 0, 1)$

cast() přetypování

Příklad - tabulka s 10 miliony záznamů

```
CREATE TEMPORARY TABLE demo (key int, val int);
INSERT INTO demo SELECT *,cast(random()*1000 AS int)FROM generate_series(1,10000
SELECT count(*) FROM demo WHERE val=123;
 count
-------
 10040
Time: 3199.271 ms

CREATE INDEX valind ON demo (val);
Time: 18076.845 ms

SELECT count(*) FROM demo WHERE val=123;
 count
-------
 10040
Time: 40.505 ms

SELECT count(*) FROM demo WHERE val=987;
 count
-------
 10132
Time: 38.674 ms

SELECT count(*) FROM demo WHERE val=358;
 count
-------
  9918
Time: 37.083 ms
```

Statistiky

- Po zadáné SQL dotazu se dotaz nejprve analyzuje parserem, který výsledek předá planneru. Planner se pokusí sestavit nejlepší exekuční plán (query plan), podle kterého se dotaz provede (předá jej exekutorovi). Pri sestavení optimálního exekučního plánu potřebuje znát rozložení dat v tabulkách (statistiky). Tyto informace je nutno pravidelně aktualizovat příkazem ANALYZE.
- V případě, že planner nemá aktuální statistiky nebo statistiky neexistují, nemá planner možnost sestavit optimální plán.
- Pro velké tabulky příkaz ANALYZE analyzuje pouze náhodný vzorek dat (statistiky se mohou i výrazně lišit).
- Po velké změně dat v tabulkách je nutné vždy spouštět příkaz ANALYZE.
- Statistiky uchovává příkaz ANALYZE v systémové tabulce `pg_statistics`.

Příkaz EXPLAIN

Pro každý zadaný SQL dotaz vytváří PostgreSQL *exekuční plán (query plan)*. Exekuční plán je stromová struktura, kde v nejnižší úrovni jsou uzly, které *skenují (scan)* přímo řádky tabulky. Existuje několik typů skenování: sekvenční, indexové a bitmapové (sequential scans, index scans, and bitmap index scans). Další uzly řeší operace join, agregace, třídění a další. Zjednodušená syntax příkazu:

```
EXPLAIN [ANALYZE] sql dotaz;
```

Příklad pro předchozí tabulku bez indexů

```
EXPLAIN SELECT count(*) FROM demo WHERE val=123;
                            QUERY PLAN
------------------------------------------------------------------
 Aggregate  (cost=162729.76..162729.77 rows=1 width=0)
   ->  Seq Scan on demo  (cost=0.00..162611.40 rows=47345 width=0)
         Filter: (val = 123)
(3 rows)
```

Sekvenční sken čte řádky tabulky postupně od prvního, dokud není celý dotaz zpracován (tj. nemusí číst celou tabulky, například při použití klauzule limit).

Ocenění (cost) je ve zvolených jednotkách, typicky čas na sekvenční načtení jedné diskové stránky (uvádí se dvě hodnoty, *čas* do načtení prvního záznamu a celkový *čas* pro všechny řádky).

EXPLAIN pokračování

Po přidání indexu pro atribut `val` tabulky `demo` z předchozího příkladu vypadá exekuční plán následovně.

```
EXPLAIN SELECT count(*) FROM demo WHERE val=123;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=47930.40..47930.41 rows=1 width=0)
   ->  Bitmap Heap Scan on demo  (cost=828.63..47805.40 rows=50000 width=0)
         Recheck Cond: (val = 123)
         ->  Bitmap Index Scan on valind  (cost=0.00..816.13 rows=50000 width=0)
               Index Cond: (val = 123)
(5 rows)
```

Bitmap heap scan znamená, že PostgreSQL našel jistou malou podmnožinu řádků (např. s využitím indexu) a je připraven načíst pouze tyto řádky. To pochopitelně představuje náročné vyhledávání, které je rychlejší pouze pokud se jedná o malou podmnožinu řádků.

EXPLAIN pokračování 2

Přidejme do klauzule `WHERE` druhou podmínku.

```
EXPLAIN SELECT count(*) FROM demo WHERE val=123 OR val=627;
                                      QUERY PLAN
-------------------------------------------------------------------------------------
 Aggregate  (cost=47679.51..47679.52 rows=1 width=0)
   ->  Bitmap Heap Scan on demo  (cost=1682.13..47430.13 rows=99750 width=0)
         Recheck Cond: ((val = 123) OR (val = 627))
         ->  BitmapOr  (cost=1682.13..1682.13 rows=100000 width=0)
               ->  Bitmap Index Scan on valind  (cost=0.00..816.13 rows=50000 wi
                     Index Cond: (val = 123)
               ->  Bitmap Index Scan on valind  (cost=0.00..816.13 rows=50000 wi
                     Index Cond: (val = 627)
(8 rows)
```

Exekuční plán tvoří následující kroky:

1. Vytvoření bitové mapy řádků `val=123` (Bitmap Index Scan)
2. Vytvoření bitové mapy řádků `val=627` (Bitmap Index Scan)
3. Logický součet obou bitových map (BitmapOR)
4. Vyhledání požadovaných řádků v tabulce (Bitmap Heap Scan) a ověření podmínky `val=123 OR val=627` (Recheck Cond).

EXPLAIN ANALYZE

Příkaz EXPLAIN s volbou ANALYZE kromě exekučního plánu také provádí zadaný SQL příkaz a vypisuje skutečné časy potřebné pro provedení exekučního plánu

```
EXPLAIN ANALYZE SELECT count(*) FROM demo WHERE val=123;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=47930.40..47930.41 rows=1 width=0)
                              (actual time=59.970..59.970 rows=1 loops=1)
   ->  Bitmap Heap Scan on demo  (cost=828.63..47805.40 rows=50000 width=0)
                              (actual time=12.007..57.882 rows=10011 loops=1)
         Recheck Cond: (val = 123)
         ->  Bitmap Index Scan on valind (cost=0.00..816.13 rows=50000 width=0)
                              (actual time=6.292..6.292 rows=10011 loops=1)
               Index Cond: (val = 123)
 Total runtime: 60.074 ms
(6 rows)
```

Všimněte si, jak se změní exekuční plán při změně podmínky v klauzuli WHERE

```
EXPLAIN ANALYZE SELECT count(*) FROM demo WHERE val<>123;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=194123.00..194123.01 rows=1 width=0)
                              (actual time=4246.309..4246.309 rows=1 loops=1)
   ->  Seq Scan on demo  (cost=0.00..169248.00 rows=9950000 width=0)
                              (actual time=0.025..2664.457 rows=9989941 loops=1)
         Filter: (val <> 123)
 Total runtime: 4246.335 ms
(4 rows)
```

CREATE INDEX syntax

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ name ] ON tab
            [ USING method ]
    (  column | ( expression )  [ COLLATE collation ]
            [ opclass ] [ ASC | DESC ]
            [ NULLS  FIRST | LAST  ] [, ...] )
    [ WITH ( storage_parameter = value [, ... ] ) ]
    [ TABLESPACE tablespace ]
    [ WHERE predicate ]
```

- Metody jsou B-tree, hash, GiST a GIN (metoda hash se nedoporučuje, protože testy neprokazují, že by poskytovala lepší výsledky než vyvážené B-stromy)
- Klauzule WHERE se používá u částečných indexů (partial index)
- TABLESPACE umožňuje administrátorovi databáze definovat umístění databázových objektů v operačním systému

Dědičnost

Dědičnost

- PostgreSQL implementuje dědičnost tabulek a řadí se tak do kategorie objektově-relačních databázových systémů (ORDBMS).
- Dědičnost (inheritance) je spolu se zapouzdřením (encapsulation) a polymorfismem (polymorphism) jedním ze základních kamenů objektově orientovaného programování.
- V případě databázového systém PostgreSQL dědičnost znamená, že můžeme vytvářet tabulky, které dědí vlastnosti od *rodičovských* tabulek, ke kterým přidávají další atributy.

Vezměme si jako jednoduchý příklad tabulku s obecnými informacemi o studentech.

```
CREATE TABLE studenti (
    id      int PRIMARY KEY,
    jmeno   varchar(40),
    email   varchar(40),
    prijeti date
);
```

Zároveň bychom ale chtěli ukládat i specifické informace o bakalářích, magistrech a studentech doktorského studia.

Dědičnost - pokračování příkladu tabulky studentů

Tabulku bakalari můžeme definovat jako potomka rodičovské třídy studenti, bude obsahovat všechny obecné atributy studentů (obecné informace) a atributy, které jsou specifické pouze pro bakaláře (střední škola a rok maturity)

```
CREATE TABLE bakalari (
    str_skola  text,
    maturita   int   -- rok
) INHERITS (studenti);
ALTER TABLE bakalari ADD PRIMARY KEY (id);
```

jak se můžeme přesvědčit v programu `psql`

```
test=> \d bakalari
            Table "public.bakalari"
  Column   |          Type          | Modifiers
-----------+------------------------+-----------
 id        | integer                | not null
 jmeno     | character varying(40)  |
 email     | character varying(40)  |
 prijeti   | date                   |
 str_skola | text                   |
 maturita  | integer                |
Indexes:
    "bakalari_pkey" PRIMARY KEY, btree (id)
Inherits: studenti

test=>
```

Dědičnost - pokračování příkladu tabulky studentů

Obdobně můžeme odvodit tabulky magistri a doktorandi

```
CREATE TABLE magistri (
       obor  char CHECK (obor in ('G', 'H'))
) INHERITS (studenti);
ALTER TABLE magistri ADD PRIMARY KEY (id);

CREATE TABLE doktorandi (
       skolitel  varchar(40)
) INHERITS (studenti);
ALTER TABLE doktorandi ADD PRIMARY KEY (id);
```

Odvozené tabulky dědí všechny atributy a všechna omezení CHECK a NOT NULL. Nedědí ale (verze PostgreSQL 9.1) omezení UNIQUE, primární a cizí klíče, které proto musí být explicitně definovány v dceřinných tabulkách.

PostgreSQL umožňuje dědit tabulky od více rodičů. V takovém případě musí mít případné společné rodičovské atributy stejná jména a stejné typy.

Dědičnost - pokračování příkladu tabulky studentů

Informace pro jednotlivé katgorie studentů ukládáme do příslušných dceřinných tabulek (není možné předávat přes rodičovské tabulky informace do odvozených, mechanismu *rules* zde neuvažujeme).

```
INSERT INTO bakalari VALUES(1, 'Hurda', 'hurda@fsv.cvut.cz', '2012-06-23',
                           'SPSZ', 2012);
INSERT INTO bakalari VALUES(2, 'Zavadil', 'zavadil@fsv.cvut.cz', '2012-06-25',
                           'SPSS', 2012);
INSERT INTO bakalari VALUES(3, 'Balcar', 'balcar@fsv.cvut.cz',
                           '2012-09-25', 'SPSS', 2007);

INSERT INTO magistri VALUES(4, 'Drda', 'drda@fsv.cvut.cz', '2011-01-12',
                           'G');
INSERT INTO magistri VALUES(5, 'Nová', 'nova@fsv.cvut.cz', '2011-01-12',
                           'H');

INSERT INTO doktorandi VALUES(6, 'Ing. Fanfulová', 'fanfulova@fsv.cvut.cz',
                             '2011-01-12', 'prof. Rambousek');
```

Příkazem INSERT bychom mohli ukládat údaje i do rodičovské tabulky `studenti`, v daném příkladu by to ale po věcné stránce nemělo smysl.

Dědičnost - pokračování příkladu tabulky studentů

```
SELECT * FROM studenti;
 id |      jmeno      |          email          |  prijeti
----+-----------------+-------------------------+------------
  1 | Hurda           | hurda@fsv.cvut.cz       | 2012-06-23
  2 | Zavadil         | zavadil@fsv.cvut.cz     | 2012-06-25
  3 | Balcar          | balcar@fsv.cvut.cz      | 2012-09-25
  4 | Drda            | drda@fsv.cvut.cz        | 2011-01-12
  5 | Nová            | nova@fsv.cvut.cz        | 2011-01-12
  6 | Ing. Fanfulová  | fanfulova@fsv.cvut.cz   | 2011-01-12
```

```
SELECT * FROM bakalari;
 id |  jmeno   |        email         |  prijeti   | str_skola | maturita
----+----------+----------------------+------------+-----------+----------
  1 | Hurda    | hurda@fsv.cvut.cz    | 2012-06-23 | SPSZ      |     2012
  2 | Zavadil  | zavadil@fsv.cvut.cz  | 2012-06-25 | SPSS      |     2012
  3 | Balcar   | balcar@fsv.cvut.cz   | 2012-09-25 | SPSS      |     2007
```

```
SELECT * FROM magistri;
 id | jmeno |       email         |  prijeti   | obor
----+-------+---------------------+------------+------
  4 | Drda  | drda@fsv.cvut.cz    | 2011-01-12 | G
  5 | Nová  | nova@fsv.cvut.cz    | 2011-01-12 | H
```

```
SELECT * FROM doktorandi;
 id |      jmeno      |          email          |  prijeti   |    skolitel
----+-----------------+-------------------------+------------+-----------------
  6 | Ing. Fanfulová  | fanfulova@fsv.cvut.cz   | 2011-01-12 | prof. Rambousek
```

Dědičnost

- Uvedený příklad bychom mohli realizovat i pomocí standardního SQL. Dědičnost ale PostgreSQL využívá i v dalších rozšířeních, jmenovitě v implementaci *partitioningu (partitioning).*
- Pokud potřebujeme explicitní informaci o tom, které tabulky se týká konkrétní řádek, můžeme použít implicitní atribut tableoid.
- Odvozenou tabulku můžeme vytvořit i pomocí `ALTER TABLE`, podrobnosti viz dokumentace http://www.postgresql.org/
- Rodičovská tabulka nemůže být odstraněna (drop), pokud existují od ní odvozené tabulky. Pokud chceme odstranit rodičovskou třídou a všechny její potomky, musíme použít v příkazu `DROP TABLE` parametr `CASCADE`.

SQL atributy typu pole

SQL atributy typu pole

```
CREATE TABLE pole (
    id int,
    seznam int[]          -- porusuje 1NF
);

INSERT INTO pole VALUES (10, '{}');     -- řetězec!
INSERT INTO pole VALUES (20, '{1}');
INSERT INTO pole VALUES (30, '{2, 3}');
INSERT INTO pole VALUES (40, '{4, 5, 6"}');
INSERT INTO pole VALUES (50, '{7, 8, 9, 10"}');

SELECT * FROM pole;
SELECT id, seznam[1] FROM pole;
SELECT id, seznam[2:3] FROM pole;    -- slicing
SELECT id FROM pole WHERE seznam[1] = 2;
```

SQL atributy textové typu pole

```
CREATE TABLE pole1 (
    cislo int,
    popis text[]
);

INSERT INTO pole1 VALUES (1, '{}');
INSERT INTO pole1 VALUES (2, '{"a", "b"}'); --uvozovky!
INSERT INTO pole1 VALUES (3, '{"aa", "bb", "cc"}');
INSERT INTO pole1 VALUES (4, '{"aaa", "bbb",
                               "ccc", "ddd"}');

SELECT * FROM pole1;
SELECT cislo, popis[1] FROM pole1;
SELECT cislo, popis[1:2] FROM pole1;    -- slicing
SELECT cislo FROM pole1 WHERE popis[1] = 'aaa';
```

Vícerozměrná pole

Ukládání po řádcích (tj. *pole polí*)

```
CREATE TABLE pole3 (
  x int[3][4]
);

INSERT INTO pole3 VALUES ('{{11,12,13,14},
                           {21,22,23,24},
                           {31,32,33,34}}');

SELECT * FROM pole3;

SELECT x[1][1], x[1][2], x[1][3], x[1][4] FROM pole3;
SELECT x[2][1], x[2][2], x[2][3], x[2][4] FROM pole3;
SELECT x[3][1], x[3][2], x[3][3], x[3][4] FROM pole3;
```

Vícerozměrná pole

```
CREATE TABLE pole2 (
   poradi int,
   zaznam text[][]   -- meze nemusíme uvádět
);                   -- pokud je uvedeme, nejsou kontrolovány

INSERT INTO pole2 VALUES (10,'{"x11"}');
INSERT INTO pole2 VALUES (20,'{{"y11", "y12"},
                               {"y21", "y22"}}');
INSERT INTO pole2 VALUES (30,'{{"z11", "z12", "z13"},
                               {"z21", "z22", "z23"},
                               {"z31", "z32", "z33"}}');

SELECT poradi, zaznam[1][2] FROM pole2;
SELECT poradi, zaznam[2][1] FROM pole2;
SELECT poradi, array_dims(zaznam) FROM pole2;
SELECT poradi, array_upper(zaznam, 1) FROM pole2;
```

Aktualizace a spojování polí

```
UPDATE pole2 SET poradi = poradi + 1000
                 WHERE zaznam[1][1] = 'z11';
SELECT * FROM pole2;


SELECT ARRAY[1, 2];
SELECT ARRAY[3, 4];

SELECT ARRAY[1, 2] || ARRAY[3, 4];
SELECT ARRAY[5, 6] || ARRAY[[1, 2], [3, 4]];
SELECT array_dims(ARRAY[5, 6]
                  || ARRAY[[1, 2], [3, 4]]);
```

Spojování a prohledávání polí

funkce pro jednorozměrná pole (přidání jedné hodnoty)

```
SELECT array_prepend( 1, ARRAY[2, 3]);
SELECT array_append( ARRAY[10, 20], 30);
```

funkce pro vícerozměrná pole (musí byt "kompatibilni")

```
SELECT array_cat(ARRAY[1,2], ARRAY[3, 4]);
SELECT array_cat(ARRAY[[1, 2], [3, 4]], ARRAY[10, 20]);
```

prohledávání polí

```
SELECT * FROM pole2;
SELECT * FROM pole2 WHERE 'y21' = ANY (zaznam);
SELECT * FROM pole2 WHERE 'x11' = ALL (zaznam);
```

Přidělování a odebírání práv

Přehled různých práv

```
SELECT       INSERT     UPDATE    DELETE
REFERENCES   TRIGGER    CREATE    CONNECT
TEMPORARY    EXECUTE    USAGE
```

Speciální práva `DROP` `GRANT` `REVOKE` náležejí vždy vlastníkovi a nemohou být předána ani odebrána.

`ALL`	označuje všechna práva.
`PUBLIC`	označuje všechny uživatele v systému.

Příklad:

```
REVOKE ALL ON zapisy FROM PUBLIC;
GRANT UPDATE ON zapisy TO jana;
```

Přidělování práv

```
GRANT { právo k objektu [,...] | role [,...]}
[ON databázový objekt]
[TO příjemce práv [,...] ]
[WITH HIERARCHY OPTIONS] [WITH GRANT OPTIONS]
                         [WITH ADMIN OPTIONS]
[FROM { CURRENT_USER | CURRENT_ROLE } ]
```

- Vlastníkem objektu se stává ten, kdo jej vytvořil.
- Implicitně může vlastník s objektem libovolně nakládat.
- *Superuser* má neomezená práva ke všem objektům.
- Pro změnu vlastníka slouží příkazy ALTER TABLE a další příkazy ALTER.

Přidělování práv

Příkazem GRANT lze přidělovat jedno nebo více práv současně.

ALL PRIVILEGES zkratka pro všechna práva, která má udílející k dispozici. Všeobecné udílení práv se obvykle nedoporučuje.

EXECUTE uděluje právo spouštět SQL rutiny.

SELECT | INSERT | UPDATE | DELETE uděluje specifikovaná práva pro obekty jako jsou tabulky a pod.

REFERENCES práva pro použití sloupců v podmínkách a omezeních, mimo cizí klíče.

TRIGGER právo vytvářet triggery na uvedených tabulkách a sloupcích.

UNDER právo vytvářet podtypy

USAGE právo pro vytváření domén, uživatelských typů, znakových sad a pod.

Přidělování práv

Administrátor databáze může vytvářet *role* (příkazem `CREATE ROLE`), které představují určitou sadu společných práv, která může být přidělena více uživatelům anebo dalším rolím.

Autorizační identifikace:

- uživatelé
- role

Přidělování práv

ON *databázový objekt* přidělení práv pro uvedený objekt:

[TABLE] *jméno objektu*
DOMAIN *jméno objektu*
COLLATION *jméno objektu*
CHARACTER SET *jméno objektu*
TRANSLATION *jméno objektu*
SPECIFIC ROUTINE *jméno objektu*

Přidělování práv

TO *příjemce práv* přidělují se práva uvedenému uživateli nebo roli

WITH HIERARCHY OPTION příjemce práv má získat práva nejen pro uvedené tabulky, ale také pro subtables.

WITH GRANT OPTIONS příjemce práv může dále udílet práva dalším uživatelům

WITH ADMIN OPTION právo přidělovat role.

FROM {CURRENT_USER | CURRENT_ROLE} Nepovinná klauzule umožňující specifikovat, kdo uděluje uvedená práva, zda CURRENT_USER nebo CURRENT_ROLE, implicitně běžný uživatel.

Odebírání práv

```
REVOKE zvláštní_volby { právo [,...] | role [,...] }
ON databázový objekt
FROM nositel práv [,...]
[GRANTED BY { CURRENT_USER | CURRENT_ROLE }  ]
[{ CASCADE | RESTRICT } ]
```

Zvláštní volby viz příkaz `GRANT`

`GRANT OPTION FOR`	možnost udílet práva
`HIERARCHY OPTION FOR`	příkaz SELECT i pro subtables
`ADMIN OPTION FOR`	možnost udílet role

Úložné procedury v PostgreSQL

Uživatelem definované funkce

V databázovém systému PostgreSQL jsou k dispozici uživatelům k dispozici čtyři typy uživatelských funkcí:

- interní funkce
- C – funkce
- funkce napsané v SQL
- funkce napsané v některém procedurálním jazyce (PLpgSQL, PL/Tcl, Python, . . .)

PostgreSQL následuje model uživatelských funkcí zavedený v Oraclu (jazyk PL/SQL byl implementován v roce 1988, jde o procedurální jazyk na bázi jazyka ADA rozšířený o SQL).

Query Language (SQL) Functions

Příklad 1:

```
CREATE FUNCTION cele_jmeno (TEXT, TEXT)
RETURNS TEXT
AS '                                    -- apostrof
   SELECT $1 || '' '' || $2;
' LANGUAGE SQL;
--
SELECT cele_jmeno('Jan', 'Novak');
```

nebo

```
CREATE FUNCTION cele_jmeno (TEXT, TEXT)
RETURNS TEXT
AS $$                                   -- dva dolary
   SELECT $1 || ' ' || $2;
$$ LANGUAGE SQL;
```

Query Language (SQL) Functions

Příklad 2:

```
CREATE TABLE soubory (      -- podklady pro SZZ ve formátu LaTeX
    soubor      VARCHAR(30) PRIMARY KEY,
    jmeno       TEXT,
    kategorie   VARCHAR(1),
    ...
);

-- parametrem je explicitně zadané jméno oponenta nebo jméno souboru;
-- v DB jsou registrováni pouze členové SZZ, oponenti mohou být i externí

CREATE FUNCTION jmeno(text)
RETURNS text
AS $$
    SELECT coalesce((SELECT jmeno
                       FROM soubory
                      WHERE soubor=$1), $1);
$$ LANGUAGE SQL;
```

Příklad — databáze filmů

```
CREATE TABLE filmy (
        id      INT PRIMARY KEY,
        rok     INT,
        titul   TEXT
);

CREATE TABLE umelci (
        id      INT PRIMARY KEY,
        jmeno   TEXT
);

CREATE TABLE obsazeni (
        film_id    INT REFERENCES filmy (id),
        umelec_id  INT REFERENCES umelci(id),
        poradi     INT
);
```

Příklad — databáze filmů

Mějme vstupní data v textovém souboru s následující jednoduchou strukturou

```
Tom Hanks                | Forrest Gump | 1994
Robin Wright Penn        | Forrest Gump | 1994
Gary Sinise              | Forrest Gump | 1994
Mykelti Williamson       | Forrest Gump | 1994
Sally Field              | Forrest Gump | 1994
Michael Connor Humphreys | Forrest Gump | 1994
Hanna Hall               | Forrest Gump | 1994
Haley Joel Osment        | Forrest Gump | 1994
Keanu Reeves             | Matrix       | 1999
Carrie-Anne Moss         | Matrix       | 1999
Hugo Weaving             | Matrix       | 1999
Gloria Foster            | Matrix       | 1999
Joe Pantoliano           | Matrix       | 1999
Marlon Brando            | Kmotr        | 1972
Al Pacino                | Kmotr        | 1972
```

...

Příklad — databáze filmů

Jednoduchou upravou převedeme vstupní textový soubor na posloupnost volání SQL uživatelské funkce, která bude plnit tabulky databáze filmů

```
SELECT uloz_hf1('Tom Hanks                ', Forrest Gump', 199
SELECT uloz_hf1('Robin Wright Penn        ', Forrest Gump', 199
SELECT uloz_hf1('Gary Sinise              ', Forrest Gump', 199
SELECT uloz_hf1('Mykelti Williamson       ', Forrest Gump', 199
SELECT uloz_hf1('Sally Field              ', Forrest Gump', 199
SELECT uloz_hf1('Michael Connor Humphreys', Forrest Gump', 199
SELECT uloz_hf1('Hanna Hall               ', Forrest Gump', 199
SELECT uloz_hf1('Haley Joel Osment        ', Forrest Gump', 199
SELECT uloz_hf1('Keanu Reeves             ', Matrix       ', 199
SELECT uloz_hf1('Carrie-Anne Moss         ', Matrix       ', 199
SELECT uloz_hf1('Hugo Weaving             ', Matrix       ', 199
SELECT uloz_hf1('Gloria Foster            ', Matrix       ', 199
SELECT uloz_hf1('Joe Pantoliano           ', Matrix       ', 199
SELECT uloz_hf1('Marlon Brando            ', Kmotr        ', 197
SELECT uloz_hf1('Al Pacino                ', Kmotr        ', 197
```

...

Příklad — databáze filmů

Funkce `uloz_hf1` pro ukládaní do databáze filmů by mohla vypadat například takto

```
CREATE OR REPLACE FUNCTION uloz_hf1(text, text, int)
RETURNS text
AS $$
   -- předpokládáme, že jména herců a filmů jsou v db jednozna
   INSERT INTO umelci
          SELECT count(*)+1, $1
            FROM umelci
           WHERE NOT EXISTS (SELECT *
                               FROM umelci
                              WHERE jmeno=$1);

    /* dále obdobně uložení filmů a obsazení */
       ...

   SELECT $2;
$$ LANGUAGE SQL;
```

SQL uživatelem definované funkce

- SQL funkce provádí posloupnost SQL příkazů uvedených v těle funkce a jako výsledek vrací první řádek posledního dotazu (pokud nespecifikujem požadavek na předání celé tabulky)
- pokud má funkce vracet celou množinu, musíme její návratový typ specifikovat jako `SETOF` *typ*
- příkazy v těle funkce se oddělují znakem středník, středník za posledním příkazem je nepovinný, pokud funkce není typu void, musí posledním příkazem být `SELECT`
- v těle funkce nelze uvádět příkazy `BEGIN`, `COMMIT`, `ROLLBACK` a `SAVEPOINT`
- tělo funkce musí být zapsáno v řetězci, obvykle se ale místo apostrofů požívají zdvojené znaky dolar.
- na argumenty se v těle funkce funkce odvoláváme pozičně, tj. $1 pro první argument, $2 pro druhý, atd. Předávat lze pouze hodnoty a ne jména.

Předefinování funkce

```
CREATE OR REPLACE FUNCTION pocet_rybniku()
RETURNS BIGINT AS $$
   SELECT COUNT(*) FROM rybniky AS "pocet";
$$ LANGUAGE SQL;
```

nebo

```
DROP FUNCTION pocet_rybniku();
CREATE FUNCTION pocet_rybniku()
RETURNS BIGINT AS $$
   SELECT COUNT(*) FROM rybniky AS "pocet";
$$ LANGUAGE SQL;

SELECT pocet_rybniku();
 pocet_rybniku
---------------
            18
(1 row)
```

SQL uživatelem definované funkce a řádkové typy

Parametrem funkce může být řádkový (kompozitní) typ. Funkci volané v příkazu `SELECT` je předáván vždy jeden celý řádek dané tabulky.

Přemyslovci						
id	jmeno	narozeni	umrti	otec	matka	rod

```
CREATE OR REPLACE FUNCTION pocet_deti(premyslovci)
RETURNS BIGINT AS $$    -- fce COUNT vraci typ BIGINT
   SELECT COUNT(*)
     FROM premyslovci   -- toto není parametr funkce!
    WHERE $1.id IN (otec, matka);
$$ LANGUAGE SQL;

SELECT jmeno, pocet_deti(premyslovci.*)
  FROM premyslovci
 WHERE jmeno IN ('Krok', 'Teta', 'Kazi', 'Libuše');
```

SQL uživatelem definované funkce a řádkové typy

Přemyslovci						
id	jmeno	narozeni	umrti	otec	matka	rod

```
SELECT jmeno, pocet_deti(premyslovci.*)
  FROM premyslovci
 WHERE jmeno IN ('Krok', 'Teta', 'Kazi', 'Libuše');

 jmeno   | pocet_deti
---------+------------
 Krok    |          3
 Kazi    |          0
 Teta    |          0
 Libuše  |          1
(4 rows)
```

SQL uživatelem definované funkce a řádkové typy

Přemyslovci						
id	jmeno	narozeni	umrti	otec	matka	rod

```
SELECT jmeno, pocet_deti(premyslovci.*)
  FROM premyslovci
 WHERE jmeno IN ('Krok', 'Teta', 'Kazi', 'Libuše');
```

Na řádek tabulky se lze odvolat pouze uvedením jména tabulky

```
SELECT jmeno, pocet_deti(premyslovci)
  FROM premyslovci
 WHERE jmeno IN ('Krok', 'Teta', 'Kazi', 'Libuše');
```

Tato možnost je ale považována za nevhodnou, protože může být matoucí (předávaným argumentem je jeden řádek a ne celá tabulka). Zápis *jméno tabulky.** explicitně zdůrazňuje, že se jedná o jediný řádek a že parametrem není celá tabulka.

Navratová hodnota kompozitního (řádkového) typu

Přemyslovci						
id	jmeno	narozeni	umrti	otec	matka	rod

```
CREATE OR REPLACE FUNCTION otec(int)
RETURNS premyslovci AS $$
 SELECT id, jmeno, narozeni, umrti, otec, matka, rod
   FROM premyslovci
  WHERE id=(SELECT otec
              FROM premyslovci
             WHERE id=$1);
$$ LANGUAGE SQL;
```

Navratová hodnota kompozitního (řádkového) typu

Přemyslovci						
id	jmeno	narozeni	umrti	otec	matka	rod

```
SELECT otec(962);          -- id 962 ('Teta')
       otec
-----------------
 (964,Krok,,,,,)
(1 row)

SELECT * FROM otec(962);
 id  | jmeno | narozeni | umrti | otec | matka | rod
-----+-------+----------+-------+------+-------+-----
 964 | Krok  |          |       |      |       |
(1 row)
```

Vyběr jediného atributu

Přemyslovci						
id	jmeno	narozeni	umrti	otec	matka	rod

```
SELECT (otec(962)).jmeno;        -- závorky jsou nutné!
  jmeno
-------
 Krok
(1 row)


SELECT jmeno(otec(962));
  jmeno
-------
 Krok
(1 row)
```

Vstupní a výstupní parametry

```
DROP FUNCTION otec(IN pid int, OUT id int, OUT jmeno text);
CREATE FUNCTION otec(IN pid int, OUT id int, OUT jmeno text)
AS $$
 SELECT id, jmeno
   FROM premyslovci
  WHERE id=(SELECT otec
              FROM premyslovci
             WHERE id=$1);
$$ LANGUAGE SQL;
```

Výstupní parametry vytvářejí implicitní výstupní kompozitní typ.

```
SELECT otec(962);
```

```
  otec
------------
 (964,Krok)
```

```
SELECT * FROM otec(962);
```

```
 id  | jmeno
-----+-------
 964 | Krok
```

Návratová hodnota typu množina

Přemyslovci						
id	jmeno	narozeni	umrti	otec	matka	rod

```
CREATE OR REPLACE FUNCTION sourozenci(varchar(50))
RETURNS SETOF premyslovci AS
$$
SELECT DISTINCT C.*
  FROM premyslovci A
       JOIN
       premyslovci B
       ON B.id IN (A.otec, A.matka) AND A.jmeno=$1
       JOIN
       premyslovci C
       ON B.id IN (C.otec, C.matka)
$$ LANGUAGE SQL;
```

Návratová hodnota typu množina

Přemyslovci						
id	jmeno	narozeni	umrti	otec	matka	rod

```
SELECT * FROM sourozenci('Václav I.');
```

```
  id  |       jmeno        | narozeni | umrti | otec | mat
------+--------------------+----------+-------+------+----
   96 | Anežka Česká       |     1211 |  1282 |  957 |   6
  109 | Anna Přemyslovna   |     1204 |  1265 |  957 |   6
 1071 | Václav I.          |     1205 |  1253 |  957 |   6
(3 rows)
```

PL/pgSQL

PL/pgSQL

- PL/pgSQL je procedurální jazyk pro databázový systém PostgreSQL
- umožňuje psát funkce a triggery
- většinu z toho co lze napsat v C lze naprogamovati i v PL/pgSQL (s výjimkou vstupů a výstupů a pod.)
- inspirován jazykem PL/SQL firmy Oracle
- je přenositelný, dá se snadno naučit
- PL/pgSQL může používat všechny datové typy, operátory a funkce SQL

Další procedurlní jazyky: PL/Tcl, PL/Perl, PL/Python a pochopitelně C (a všechny s ním kompatibilní jazyky)

PL/pgSQL

- každý procedurální jazyk musí být nainstalován
- procedurální jazyk nainstalovaný v databazi `template1` je automaticky k dispozici i pro všechny následně vytvořené databáze

```
CREATE LANGUAGE jméno_jazyka
```

Příklad

```
su -
su - postgres
psql template1
CREATE LANGUAGE plpqsql;
```

PL/pgSQL

```
CREATE OR REPLACE FUNCTION priklad(integer)
RETURNS integer AS $$
    ...
$$ LANGUAGE plpgsql;
```

Pokud potřebujeme v těle funkce používat zdvojené dolary, můžeme použít *dollar-quoted* řetězcové literály.

```
CREATE FUNCTION ukazka(text) RETURNS text AS $abc$
BEGIN
   return '$$ ' || $1 || ' $$';  -- $$ v těle funkce
END
$abc$ LANGUAGE plpgsql;
--
SELECT ukazka('ahoj');
```

Bloky v jazyce PL/pgSQL

```
[ <<návěští>>]
[ DECLARE
    deklarace... ]
BEGIN
    příkazy
END [návěští] ;
```

Příklad

```
CREATE FUNCTION ukazka2(text) RETURNS text AS $$
DECLARE
   tmp TEXT := ' *** ';
BEGIN
   RAISE NOTICE 'Hodnota proměnné ''tmp'' je ''%''!', tmp;
   RETURN tmp || $1 || tmp;
END ;
$$ LANGUAGE plpgsql;
```

Deklarace v jazyce PL/pgSQL

```
jméno [ CONSTANT ] typ [ NOT NULL ]
                       [ { DEFAULT | := } výraz ];
```

PL/pgSQL - jména parametrů funkcí

```
CREATE FUNCTION ukazka3(castka REAL)
RETURNS REAL AS $$
BEGIN
   RETURN 0.19*castka;
END ;
$$ LANGUAGE plpgsql;
```

Jména parametrů $1 zůstávají i nadále v platnosti.

Alternativně lze jméno parametru definovat jako *alias*

jméno ALIAS FOR $*n*

PL/pgSQL - výstupní parametry funkcí

- alternativa k předávání příkazem `return`
- příkaz `return` lze stále použít, je ale redundantní

```
CREATE FUNCTION ukazka4(castka REAL, OUT dph REAL) AS
$$
BEGIN
   dph := 0.19*castka;
END ;
$$ LANGUAGE plpgsql;

SELECT ukazka4(100);

 ukazka4
---------
      19
(1 row)
```

PL/pgSQL

Copying Types Datový typ pro proměnnou nebo sloupec tabulky. Skutečný typ na který se odkazujeme nemusíme předem znát.

```
proměnná%TYPE;
```

Row Types Kompozitní proměnné

```
jméno jméno_tabulky%ROWTYPE;
jméno jméno_kompozitního_typu;
```

Record Types Nemají předem danou strukturu.

```
jméno RECORD;
```

RENAME Přejmenování, především u triggerů (NEW a OLD)

```
RENAME původní_jméno TO nové_jméno;
```

PL/pgSQL základní příkazy

Přiřazení

```
idenifikátor := výraz;
```

SELECT INTO

```
SELECT INTO kam select_výrazy FROM ...;
```

Výrazy a dotazy bez ukládání výsledné hodnoty

```
PERFORM dotaz;
```

Prázdný příkaz

```
NULL;
```

Dynamické příkazy

```
EXECUTE příkazový_řetězec [ INTO kam ];
```

Návratová hodnota

```
RETURN výraz;
RETURN NEXT výraz;
```

PL/pgSQL řídící struktury

IF-THEN

```
IF logický výraz THEN
   příkazy
END IF
```

nebo

```
IF logický výraz THEN
   příkazy
ELSE
   příkazy
END IF
```

PL/pgSQL řídící struktury

Jednoduchý cyklus

```
[ <<návěští>> ]
LOOP
    příkazy
END LOOP [ návěští ];
```

Exit

```
EXIT [ návěští ] [ WHEN výraz ];
```

Continue

```
CONTINUE [ návěští ] [ WHEN výraz ];
```

PL/pgSQL řídící struktury

WHILE

```
[ <<návěští>> ]
WHILE výraz LOOP
    příkazy
END LOOP [ návěští ];
```

FOR

```
[ <<návěští>> ]
FOR jméno IN [ REVERSE] výraz..výraz LOOP
    příkazy
END LOOP [ návěští ];
```

PL/pgSQL řídící struktury

Průchod výsledky dotazu

```
[ <<návěští>> ]
FOR záznam_nebo_řádek IN dotaz LOOP
    příkazy
END LOOP [ návěští ];
```

nebo

```
[ <<návěští>> ]
FOR záznam_nebo_řádek IN EXECUTE řetězec LOOP
    příkazy
END LOOP [ návěští ];
```

kde příkazový řetězec se dynamicky vyhodnocuje při každém spuštění cyklu FOR.

PL/pgSQL výjimky

```
[ <<návěští>>]
[ DECLARE
    deklarace... ]
BEGIN
    příkazy
EXCEPTION
   WHEN podmínka [ OR podmínka ... ] THEN
      příkazy ovladače
 [ WHEN podmínka [ OR podmínka ... ] THEN
      příkazy ovladače
   ... ]
END;
```

PL/pgSQL kurzory