```
SELECT CASE WHEN sloupec = 0 THEN 'NE'
            WHEN sloupec = 1 THEN 'ANO' END
   FROM tabulka;
```

V případě, že se nenajde hledaná konstanta a nebo žádný výraz není pravdivý, tak je výsledkem hodnota za klíčovým slovem ELSE – nebo NULL, pokud chybí ELSE.

Agregační funkce s definovaným pořadím

Výsledek nových agregačních funkcí – string_agg, array_agg závisí na pořadí ve kterém se zpracovala agregovaná data. Proto je možné přímo v agregační funkci určit v jakém pořadí bude agregační funkce načítat hodnoty. **Klauzule ORDER BY musí být za posledním argumentem agregační funkce**.

Vrátí seznam zaměstnanců v každém oddělení řazený podle příjmení:

```
SELECT sekce_id, string_agg(prijmeni, ',' ORDER BY prijmeni)
 FROM zamestnanci
 GROUP BY sekce_id
```

Agregační funkce nad uspořádanou množinou

Tato speciální syntax se používá pouze pro funkce, jejichž výpočet vyžaduje seřazená data (např. výpočet percentilů). Následující dotaz zobrazí medián (50% percentil) mzdy zaměstnanců.

```
SELECT percentile_cont(0.5[16]) WITHIN GROUP (ORDER BY mzda)
   FROM zamestnanci
```

Poddotazy

Příkaz SELECT může obsahovat vnořené příkazy SELECT. Vnořený příkaz SELECT se nazývá **poddotaz** a vkládá se do oblých závorek. Poddotazy se mohou použít i u dalších SQL příkazů.

Poddotaz ve WHERE

Používá se pro filtrování – následující dotaz zobrazí obce z okresu Benešov:

```
SELECT nazev
   FROM obce o
  WHERE o.okres_id = (SELECT id
                        FROM okresy
                       WHERE kod = 'BN')
```

Korelované poddotazy

Poddotaz se může odkazovat na výsledek, který produkuje vnější dotaz.

Pro každého zaměstnance zobrazí seznam jeho dětí:

```
SELECT jmeno, prijmeni,
       (SELECT string_agg(jmeno, ',')
          FROM deti d
         WHERE d.zamestnanec_id = z.id)
   FROM zamestnanci z
```

Zobrazí zaměstnance, kteří mají děti:

```
SELECT jmeno, prijmeni
   FROM zamestanci z
  WHERE EXISTS(SELECT id
                 FROM deti d
                WHERE d.zamestnanec_id = z.id)
```

10 Rozšíření vůči ANSI/SQL umožňuje zadat více parametrů jako pole – výsledkem je opět pole.

Zobrazí z každého oddělení dva nejstarší zaměstnance (více násobné použití tabulky)

```
SELECT jmeno, prijmeni
    FROM zamestnanci z1
   WHERE vek IN (SELECT vek
                   FROM zamestnanci z2
                  WHERE z2.sekce_id = z1.sekce_id
                  ORDER BY vek DESC
                  LIMIT 2)
```

Spojení relací[11] JOIN

Příkaz JOIN spojuje relace (tabulky) vedle sebe a to na základě stejných hodnot v jednom nebo více atributech (sloupcích). Každé spojení specifikuje dvě relace (spojkou je klíčové slovo JOIN) a podmínku, která určuje, jak se tyto relace budou spojovat (zapsanou za klíčovým slovem ON).

Vnitřní spojení relací – INNER JOIN

Nejčastější varianta – do výsledku se zahrnou pouze řádky, které se podařilo dohledat v obou relacích (stejné hodnota/hodnoty) se nalezly v obou tabulkách.

Zobrazí jméno dítěte a jméno rodiče (zaměstnance) – v případě, že má zaměstnanec více dětí, tak jeho jméno bude uvedeno opakovaně:

```
SELECT d.jmeno, d.prijmeni, z.jmeno, z.prijmeni
   FROM deti d
        JOIN zamestnanci z
        ON d.zamestnanec_id = z.id
```

Vnější spojení relací – OUTER JOIN

Jedná se o rozšíření vnitřního spojení – kromě řádků, které se spárovaly se do výsledku zařadí i nespárované řádky z tabulky nalevo od slova JOIN (LEFT JOIN) nebo napravo od slova JOIN (RIGHT JOIN). Chybějící hodnoty se nahradí hodnotou **NULL**.

Často se používá dohromady s testem na hodnotu NULL – operátorem **IS NULL**[12]. Tím se vyberou nespárované řádky – např. pro zobrazení zaměstnanců, kteří nemají děti, lze použít dotaz:

```
SELECT z.jmeno, z.prijmeni
   FROM zamestnanci z
        LEFT JOIN deti d
        ON z.id = d.zamestnanec_id
  WHERE d.id IS NULL.
```

Použití derivované tabulky

Poddotaz se může objevit i v klauzuli FROM – pak jej označujeme jako derivovaná tabulka[13]. I derivovanou tabulku lze spojovat s běžnými tabulkami (obojí je relací).

Následující příklad zobrazí seznam nejstarších zaměstnanců z každého oddělení:

```
SELECT z.jmeno, z.prijmeni
   FROM zamestnanci z
        JOIN (SELECT sekce_id, MAX(vek) AS vek
                FROM zamestnanci
               GROUP BY sekce_id) s
         ON z.sekce_id = s.sekce_id
        AND z.vek = s.vek
```

11 Tabulka je relací. Výsledek SQL dotazu je relací. Tudíž příkaz SELECT může se aplikovat na tabulku nebo i na výsledek jiného příkazu SELECT.

12 **Pro hodnotu NULL není možné použít operator =.**

13 SELECT ze SELECTu

Dotazy s LATERAL relacemi

Klauzule LATERAL umožňuje ke každému záznamu relace X připojit výsledek poddotazu (derivované tabulky), uvnitř kterého je možné použít referenci na relaci X. Místo derivované tabulky lze použít funkci, která vrací tabulku, a pak atribut(y) z relace X může být argumentem této funkce.

Pro každý záznam z tabulky a vrátí všechny záznamy z tabulky b, pro které platí, že atribut a je větší než dvojnásobek atributu b.

```
SELECT  *
 FROM a,
      LATERAL (SELECT *
                 FROM b
                WHERE a.a > 2 * b.b) x;
```

Pro každou hodnotu vrátí součet všech kladných celých čísel menších rovno této hodnotě:

```
SELECT a, sum(i)
  FROM a,
       LATERAL generate_series(1, a) g(i)
 GROUP BY a
 ORDER BY 1;
```

LATERAL join lze využít pro efektivní provedení úlohy nalezení top N pro každou skupinu:

```
SELECT *
 FROM okresy,
      LATERAL (SELECT *
                 FROM obce
                WHERE obce.okres_id = okresy.id
                ORDER BY pocet_obyvatel DESC
                LIMIT 3);
```

Analytické (window) funkce

Analytické funkce se počítají pro každý prvek definované podmnožiny, např. pořadí prvku v podmnožině. Na rozdíl od agregačních funkcí se podmnožiny nedefinují klauzulí GROUP BY, ale klauzulí PARTITION hned za voláním analytické funkce (v závorce za klíčovým slovem OVER). Mezi nejčastěji používané analytické funkce bude patřit funkce row_number (číslo řádku) nebo ranking (pořadí hodnoty), případně dense_rank a percent_rank.

Pozor – pro analytické funkce nelze použít klauzuli HAVING – filtrování hodnot se řeší použitím derivované tabulky.

Následující dotaz vybere deset nejdéle zaměstnaných pracovníků (na základě porovnání osobních čísel):

```
SELECT jmeno, prijmeni
   FROM (SELECT rank() OVER (ORDER BY id),
                jmeno, prijmeni
           FROM zamestnanci
          WHERE ukonceni_prac_pomeru IS NULL) s
  WHERE s.rank <= 10
```

Zobrazení dvou nejstarších zaměstnanců z každého oddělení:

```
SELECT jmeno, prijmeni
   FROM (SELECT rank() OVER (PARTITION BY sekce_id
                             ORDER BY vek DESC),
                jmeno, prijmeni
           FROM zamestnanci) s
  WHERE s.rank <= 2
```

PostgreSQL ve verzi 12

```
createdb jménodb
```

Vytvoří novou databázi

```
dropdb jménodb
```

odstraní existující databázi

```
psql jménodb[1]
```

spustí SQL konzoli

```
pg_dump jménodb > jméno_souboru
```

Vytvoří zálohu databáze

SQL konzole – psql

Umožní zadání SQL příkazu a zobrazí jeho výsledek.

Přehled důležitých příkazů

Každý příkaz začíná zpětným lomítkem "\" a není ukončen středníkem.

`\c jménodb`	přepnutí do jiné databáze
`\l`	zobrazí seznam databází
`\d objekt`	zobrazí popis objektu (tabulky, pohledu)
`\dt+`	zobrazí seznam tabulek
`\dv`	zobrazí seznam pohledů
`\df *filtr*`	zobrazí seznam funkcí
`\sf funkce`	zobrazí zdrojový kód funkce
`\i`	importuje soubor
`\h SQL`	zobrazí syntaxi SQL příkazu
`\?`	zobrazí seznam psql příkazů
`\q`	ukončí konzolu
`\x`	přepíná řádkové a sloupcové zobrazení
`\timing on`	zapíná měření času zpracování dotazu

Konfigurace konzole

Soubor *.psqlrc*

```
\set QUIET on

\setenv PAGER less[2]
\setenv LESS '-iMSx4 -RSFX -e'
\pset pager always
\pset linestyle unicode
\pset null 'NULL'
\set FETCH_COUNT 1000
\set HISTSIZE 5000
\timing
\set HISTFILE ~/.psql_history-:DBNAME
\set HISTCONTROL ignoredups
\set PROMPT1 '(%n@%M:%>) [%/] > '
\set ON_ERROR_ROLLBACK on
\set AUTOCOMMIT off[3]

\set QUIET off
```

1 Nastavením systémové proměnné PGDATABASE lze určit implicitní databázi
2 Alternativním pagerem může být pager *pspg*. https://github.com/okbob/pspg
3 Doporučeno pro produkci – vynucuje potvrzení změn explicitním COMMITem. Po vypnutí *autocommitu* se psql bude chovat podobně jako konzole *Oracle*.

Export a import dat

Příkaz COPY

Pomocí příkazu COPY můžeme číst a zapisovat soubory na serveru (pouze superuser) nebo číst ze **stdin** a zapisovat na **stdout**. Podobný příkaz `\copy` v **psql** umožňuje číst a zapisovat soubory na klientském počítači.

Export tabulky zaměstnanci do CSV souboru

```
COPY zamestnanci TO '/tmp/zam.csv'
   CSV HEADER
   DELIMITER ';' FORCE QUOTE *;
```

Import tabulky zaměstnanci z domovského adresáře uživatele (v konzoli)

```
\copy zamestnanci from ~/zamestnanci.dta
```

pg_dump – zajímavé parametry

Příkaz `pg_dump` slouží k jednoduchému zálohování databáze[4].

`-f`	specifikuje cílový soubor
`-a`	exportuje pouze data
`-s`	exportuje pouze definice
`-c`	odstraní objekty před jejich importem
`-C`	vloží příkaz pro vytvoření nové databáze
`-t`	exportuje pouze jmenovanou tabulku
`-T`	neexportuje uvedenou tabulku
`--disable-triggers`	během importu blokuje triggery
`--inserts`	generuje příkazy INSERT místo COPY
`-Fc`	záloha je průběžně komprimovaná s dodatečnými meta informacemi[5]

Základní konfigurace PostgreSQL

Soubor *postgresql.conf*

Po instalaci PostgreSQL je nutné nastavit několik málo konfiguračních parametrů, které ovlivňují využití operační paměti (výchozí nastavení je zbytečně úsporné).

```
shared_buffers= 2GB
```

velikost paměti pro uložení datových stránek (1/5..1/3 RAM[6])

```
work_mem = 10MB
```

limit paměti pro běžnou manipulaci s daty (10..100MB)

```
maintenance_work_mem = 200MB
```

limit paměti pro údržbu (100MB ..)

```
effective_cache_size = 6GB
```

odhad objemu dat cache (2/3 RAM)

```
max_connections = 100
```

max počet přihlášených uživatelů (často zbytečně vysoké)

4 Příkaz pg_dump nezálohuje uživatele. K tomuto účelu se používá příkaz pg_dumpall s parametrem -r. Zálohování příkazem pg_dump je vhodné pro databáze do velikosti cca 50GB. Pro větší databáze je praktičtější použít jiné metody zálohování.
5 Pro obnovu je nutné použít pg_restore, obnovit lze i každou vybranou tabulku.
6 Doporučené hodnoty platí pro tzv dedikovaný server – tj počítač, který je vyhrazen primárně pro provoz databáze s 8GB RAM.

Mělo by platit[7]:

```
shared_buffers + 2 * work_mem * max_connection <= 2/3 RAM
shared_buffers + 2 * maintenance_work_mem <= 1/2 RAM
max_connections <= 10 * (počet_CPU)
```

Pokud dochází k intenzivnímu zápisu, může mít smysl zvýšit hodnotu *max_wal_size*. Pokud velikost transakčního logu přesáhne tuto hranici, dojde k provedení CHECKPOINTu. Vyšší hodnota znamená nižší frekvenci checkpointů a naopak. Výchozí hodnota 1GB je pro obvyklé použití dostatečná.

```
max_wal_size = 1GB
```

Po CHECKPOINTu lze zahodit transakční logy vztahené k času před CHECKPOINTem. Za optimální frekvenci CHECKPOINTů se považuje 5 – 15 min.

```
listen_addresses = '*'
```

```
A pro vzdálený přístup povolit TCP
```

SQL

Nejdůležitějším SQL příkazem je příkaz SELECT. Při zápisu je nutné dodržovat pořadí jednotlivých klauzulí:

```
SELECT AVG(a.sloupec1), b.sloupec4
   FROM tabulka1 a
        JOIN tabulka2 b
        ON a.sloupec1 = b.sloupec2
  WHERE b.sloupec3 = 'něco'
  GROUP BY b.sloupec4
  HAVING AVG(a.sloupec1) > 100
  ORDER BY 1
  LIMIT 10
```

Sjednocení, průnik, rozdíl relací

Pro relace (tabulky) existují operace sjednocení (UNION), průnik (INTERSECT) a rozdíl (EXCEPT). Častou operací je sjednocení relací – výsledků dvou příkazů SELECT – operace sloučí řádky (a zároveň odstraní případné duplicitní řádky). Podmínkou je stejný počet sloupců a konvertibilní datové typy slučovaných relací.

Vybere 10 nejstarších zaměstnanců bez ohledu zdali se jedná o interního nebo externího zaměstnance:

```
SELECT jmeno, prijmeni, vek
   FROM zamestnanci
UNION[8]
SELECT jmeno, prijmeni, vek
  FROM externi_zamestnanci
 ORDER BY vek DESC[9]
 LIMIT 10;
```

CASE

Konstrukce CASE se používá pro transformace hodnot – zobrazení, bez nutnosti definovat vlastní funkce. Existují dva zápisy – první hledá konstantu, v druhém se hledá platný výraz:

```
SELECT CASE sloupec WHEN 0 THEN 'NE'
                    WHEN 1 THEN 'ANO' END
  FROM tabulka;
```

7 Jedná se o orientační hodnoty určené pro počáteční konfiguraci "typického použití" databáze.
8 Při použití UNION ALL nedochází k odstranění duplicitních řádků – což může zrychlit vykonání dotazu.
9 Klauzule ORDER BY se aplikuje na výsledek algebraických operací