SQL TAHÁK 1/10

SQL, neboli jazyk strukturovaných dotazů (Structured Query Language), je jazyk s kterým se dorozumíváme s databázemi. Umožňuje nám vybrat konkrétní data a vytvořit komplexní výstup dat z databáze. V dnešní době je tak SQL univerzálním jazykem pro data a je používán prakticky ve všech technologií, která data zpracovávají.

VZOREK DAT

COUNTRY

id	name	population	area
1	France	66600000	640680
2	Germany	80700000	357000
...	...	...	...

CITY

id	name	country_id	population	rating
1	Paris	1	2243000	5
2	Berlin	2	3460000	3
...	...	...	...	...

DOTAZOVÁNÍ NA JEDNU TABULKU

Načtení všech sloupců z tabulky COUNTRY:

```
SELECT *
FROM country;
```

Načtení sloupců ID a NAME z tabulky CITY:

```
SELECT id, name
FROM city;
```

Načtení názvů měst (sloupec NAME) z tabulky CITY,
srovnaného podle sloupce RATING
v defaultním vzestupném řazení (ASCending order):

```
SELECT name
FROM city
ORDER BY rating [ASC];
```

Načtení názvů měst (sloupec NAME) z tabulky CITY,
srovnaného podle sloupce RATING
v sestupném řazení (DESCending order):

```
SELECT name
FROM city
ORDER BY rating DESC;
```

PŘEZDÍVKY (ALIASES)

SLOUPCE (COLUMNS)

```
SELECT name AS city_name
FROM city;
```

TABULKY (TABLES)

```
SELECT co.name, ci.name
FROM city AS ci
JOIN country AS co
  ON ci.country_id = co.id;
```

FILTROVÁNÍ VÝSTUPU

POROVNÁVACÍ OPERÁTORY

Načtení názvů měst (sloupec NAME) z tabulky CITY,
která mají hodnoceni (RATING) vyšší než 3:

```
SELECT name
FROM city
WHERE rating > 3;
```

Načtení názvů měst (sloupec NAME) z tabulky CITY,
která se nejmenují Berlín, nebo Madrid:

```
SELECT name
FROM city
WHERE name != ‚Berlin‘
        AND name != ‚Madrid‘;
```

TEXTOVÉ OPERÁTORY

Načtení názvů měst (sloupec NAME) z tabulky CITY,
která začínají s P, nebo končí se S:

```
SELECT name
FROM city
WHERE name LIKE ‚P%‘
    OR name LIKE ‚%s‘;
```

Načtení názvů měst (sloupec NAME) z tabulky CITY,
která začínají libovolným písmenem a následuje „ublin“
(jako Dublin v Irsku nebo Lublin v Polsku):

```
SELECT name
FROM city
WHERE name LIKE ‚_ublin‘;
```

DALŠÍ OPERÁTORY

Načtení názvů měst (sloupec NAME) z tabulky CITY,
která mají 500 až 5 milionů obyvatel:

```
SELECT name
FROM city
WHERE population BETWEEN 500000 AND
5000000;
```

Načtení názvů měst (sloupec NAME) z tabulky CITY,
kterým nechybí hodnota v sloupci RATING

```
SELECT name
FROM city
WHERE rating IS NOT NULL;
```

Načtení názvů měst (sloupec NAME) z tabulky CITY,
která jsou v zemích (tabulka COUNTRY),
které mají hodnotu ID 1, 4, 7, nebo 8

```
SELECT name
FROM city
WHERE country_id IN (1, 4, 7, 8);
```

DOTAZOVANÍ NA VÍCE TABULEK

VNITŘNÍ PROPOJENÍ (INNER JOIN)

JOIN (přednastavená hodnota: INNER JOIN) vrátí řádky, které mají v obou tabulkách odpovídající hodnoty:

```
SELECT city.name, country.name
FROM city
[INNER] JOIN country
   ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	3	Iceland

PROPOJENÍ VLEVO (LEFT JOIN)

LEFT JOIN vrátí všechny řádky z levé tabulky s odpovídajícími řádky z pravé tabulky. Pokud neexistuje žádný odpovídající řádek, vrátí se hodnoty NULL jako hodnoty z druhé tabulky:

```
SELECT city.name, country.name
FROM city
LEFT JOIN country
   ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL

PROPOJENÍ VPRAVO (RIGHT JOIN)

RIGHT JOIN vrátí všechny řádky z pravé tabulky s odpovídajícími řádky z levé tabulky. Pokud neexistuje žádný odpovídající řádek, vrátí se hodnoty NULL jako hodnoty z levé tabulky:

```
SELECT city.name, country.name
FROM city
RIGHT JOIN country
   ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
NULL	NULL	NULL	3	Iceland

PLNÉ PROPOJENÍ (FULL JOIN)

FULL JOIN (nebo-li FULL OUTER JOIN) vrátí všechny řádky z obou tabulek – pokud v druhé tabulce není žádný odpovídající řádek, vrátí se hodnoty NULL:

```
SELECT city.name, country.name
FROM city
FULL [OUTER] JOIN country
   ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL
NULL	NULL	NULL	3	Iceland

KŘÍŽOVÉ PROPOJENÍ (CROSS JOIN)

CROSS JOIN vrátí všechny možné kombinace řádků z obou tabulek. K dispozici jsou dva způsoby zápisu:

```
SELECT city.name, country.name
FROM city
CROSS JOIN country;

SELECT city.name, country.name
FROM city, country;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
1	Paris	1	2	Germany
2	Berlin	2	1	France
2	Berlin	2	2	Germany

PŘIROZENÉ PROPOJENÍ (NATURAL JOIN)

NATURAL JOIN spojí tabulky podle všech sloupců se stejným názvem:

```
SELECT city.name, country.name
FROM city
NATURAL JOIN country;
```

CITY			COUNTRY	
country_id	id	name	name	id
6	6	San Marino	San Marino	6
7	7	Vatican City	Vatican City	7
5	9	Greece	Greece	9
10	11	Monaco	Monaco	10

NATURAL JOIN použil ke shodě řádků tyto sloupce:
CITY.ID, CITY.NAME, COUNTRY.ID, COUNTRY.NAME
NATURAL JOIN se v praxi používá velmi zřídka..

SESKUPOVÁNÍ A GROUP BY

GROUP BY seskupuje řádky, které mají v určených sloupcích stejné hodnoty. Vypočítává souhrny (agregáty) pro každou jedinečnou kombinaci hodnot.

CITY

id	name	country_id
1	Paris	1
101	Marseille	1
102	Lyon	1
2	Berlin	2
103	Hamburg	2
104	Munich	2
3	Warsaw	4
105	Cracow	4

→

CITY

country_id	count
1	3
2	3
4	2

SESKUPOVACÍ FUNKCE

AVG(*sloupec*) – průměrná hodnota pro řádky ve skupině
COUNT(*sloupec*) – počet hodnot pro řádky ve skupině
MAX(*sloupec*) – maximální hodnotu v rámci skupiny
MIN(*sloupec*) – minimální hodnotu v rámci skupiny
SUM(*sloupec*) – součet hodnot v rámci skupiny

PŘÍKLADY POUŽITÍ

Zjištění počtu měst:

```
SELECT COUNT(*)
FROM city;
```

Zjištění počtu měst s nenulovým hodnocením:

```
SELECT COUNT(rating)
FROM city;
```

Zjištění počtu rozdílných zemí (hodnot ID):

```
SELECT COUNT(DISTINCT country_id)
FROM city;
```

Zjištění nejmenší a největší země dle populace:

```
SELECT MIN(population), MAX(population)
FROM country;
```

Zjištění celkového poštu obyvatel měst, v příslušných zemích:

```
SELECT country_id, SUM(population)
FROM city
GROUP BY country_id;
```

Zjištění průměrného hodnocení měst v příslušných zemích, je-li průměr nad hodnotou 3:

```
SELECT country_id, AVG(rating)
FROM city
GROUP BY country_id
HAVING AVG(rating) > 3.0;
```

PODDOTAZY

Poddotaz je dotaz, který je vnořen do jiného dotazu nebo do jiného poddotazu. Existují různé typy poddotazů.

JEDNA HODNOTA

Nejjednodušší poddotaz vrátí přesně jeden sloupec a přesně jeden řádek.
Lze jej použít s operátory porovnání **=**, **<**, **<=**, **>** nebo **>=**.

Tento dotaz najde města se stejným hodnocením jako Paříž:

```
SELECT name
FROM city
WHERE rating = (
  SELECT rating
  FROM city
  WHERE name = 'Paris'
);
```

VÍCE HODNOT

Poddotaz může také vrátit více sloupců nebo více řádků.
Takové poddotazy lze použít s operátory **IN**, **EXISTS**, **ALL** nebo **ANY**.

Tento dotaz najde města v zemích, které mají více než 20 milionů obyvatel:

```
SELECT name
FROM city
WHERE country_id IN (
  SELECT country_id
  FROM country
  WHERE population > 20000000
);
```

SOUVISEJÍCÍ PODDOTAZ

Související poddotaz odkazuje na tabulky uvedenou ve vnějším dotazu.
Související poddotaz závisí na vnějším dotazu a nelze jej spustit nezávisle na vnějším dotazu.

Tento dotaz najde města s větším počtem obyvatel, než je průměrný počet obyvatel v zemi:

```
SELECT *
FROM city main_city
WHERE population > (
  SELECT AVG(population)
  FROM city average_city
  WHERE average_city.country_id = main_city.country_id
);
```

Tento dotaz najde země, které mají alespoň jedno město:

```
SELECT name
FROM country
WHERE EXISTS (
  SELECT *
  FROM city
  WHERE country_id = country.id
);
```

PROPOJOVACÍ OPERACE

Propojovací operace se používají ke spojení výsledků dvou nebo více dotazů do jednoho výsledku. Kombinované dotazy musí vrátit stejný počet sloupců a kompatibilní datové typy. Názvy odpovídajících sloupců se mohou lišit.

CYCLING

id	name	country
1	YK	DE
2	ZG	DE
3	WT	PL
...	...	...

SKATING

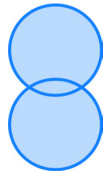
id	name	country
1	YK	DE
2	DF	DE
3	AK	PL
...	...	...

SPOJENÍ (UNION)

Spojení UNION kombinuje výsledky dvou sad výsledků a odstraňuje duplikáty.
Spojení UNION ALL neodstraní duplicitní řádky.
Tento dotaz zobrazuje německé cyklisty společně s německými bruslaři:

```
SELECT name
FROM cycling
WHERE country = 'DE'

UNION / UNION ALL
SELECT name
FROM skating
WHERE country = 'DE';
UNION circles
```

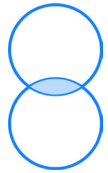


PRŮNIK (INTERSECT)

Spojení INTERSECT vrátí pouze řádky, které se objevují v obou sadách výsledků.
Tento dotaz zobrazí německé cyklisty, kteří jsou zároveň německými bruslaři:

```
SELECT name
FROM cycling
WHERE country = 'DE'

INTERSECT
SELECT name
FROM skating
WHERE country = 'DE';
INTERSECT circles
```

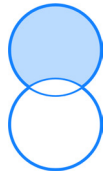


ROZDÍL (EXCEPT)

Spojení EXCEPT vrátí pouze řádky, které se objevují v první sadě výsledků, ale nezobrazují se v druhé sadě výsledků.
Tento dotaz zobrazí německé cyklisty, pokud nejsou zároveň německými bruslaři:

```
SELECT name
FROM cycling
WHERE country = 'DE'

EXCEPT / MINUS
SELECT name
FROM skating
WHERE country = 'DE';
EXCEPT circles
```



SPOJOVÁNÍ TABULEK (JOIN)

Funkce JOIN kombinuje data ze dvou tabulek.

TOY

toy_id	toy_name	cat_id
1	ball	3
2	spring	NULL
3	mouse	1
4	mouse	4
5	ball	1

CAT

cat_id	cat_name
1	Kitty
2	Hugo
3	Sam
4	Misty

JOIN obvykle kombinuje řádky se stejnými hodnotami pro zadané sloupce.
Obvykle jedna tabulka obsahuje primární klíč, což je sloupec nebo sloupce, které jednoznačně identifikují řádky v tabulce (sloupec cat_id v tabulce cat). Druhá tabulka obsahuje sloupec nebo sloupce, které odkazují na sloupce primárního klíče v první tabulce (sloupec cat_id v tabulce hraček). Takové sloupce jsou cizí klíče.
Podmínka JOIN je rovnost mezi sloupci primárního klíče v jedné tabulce a sloupci, které na ně odkazují ve druhé tabulce.

PŘÍKAZ JOIN

JOIN vrátí všechny řádky, které odpovídají podmínce ON. JOIN se také nazývá INNER JOIN

```
SELECT *
FROM toy
JOIN cat
  ON toy.cat_id = cat.cat_id;
```

toy_id	toy_name	cat_id	cat_id	cat_name
5	ball	1	1	Kitty
3	mouse	1	1	Kitty
1	ball	3	3	Sam
4	mouse	4	4	Misty

Existuje také jiná, starší syntaxe, **ale nedoporučuje se.**
Vypište spojené tabulky v klauzuli FROM a umístěte podmínky do klauzule WHERE.

```
SELECT *
FROM toy, cat
WHERE toy.cat_id = cat.cat_id;
```

PODMÍNKY SPOJENÍ

Podmínka pro JOIN nemusí být rovnost – může to být jakákoli podmínka, kterou chcete.
JOIN neinterpretuje podmínku, pouze kontroluje, zda ji řádky splňují.

Chcete-li odkazovat na sloupec v dotazu JOIN, musíte použít celý název sloupce:
nejprve název tabulky, poté tečku (.) a název sloupce:

```
  ON cat.cat_id = toy.cat_id
```

Název tabulky můžete vynechat a použít pouze název sloupce,
pokud je název sloupce jedinečný ve všech sloupcích ve spojených tabulkách.

PŘIROZENÉ SPOJENÍ (NATURAL JOIN)

Pokud tabulky obsahují sloupce se stejným názvem, můžete místo JOIN použít NATURAL JOIN.

```
SELECT *
FROM toy
NATURAL JOIN cat;
```

cat_id	toy_id	toy_name	cat_name
1	5	ball	Kitty
1	3	mouse	Kitty
3	1	ball	Sam
4	4	mouse	Misty

Společný sloupec se ve výsledkové tabulce objeví pouze jednou.
Poznámka: NATURAL JOIN se v reálném životě používá jen zřídka.

SPOJENÍ VLEVO (LEFT JOIN)

LEFT JOIN vrátí všechny řádky z levé tabulky se shodnými řádky z pravé tabulky.
Řádky bez shody jsou vyplněny hodnotami NULL.
LEFT JOIN se také nazývá LEFT OUTER JOIN.

```
SELECT *
FROM toy
LEFT JOIN cat
  ON toy.cat_id = cat.cat_id;
```

toy_id	toy_name	cat_id	cat_id	cat_name
5	ball	1	1	Kitty
3	mouse	1	1	Kitty
1	ball	3	3	Sam
4	mouse	4	4	Misty
2	spring	NULL	NULL	NULL

whole left table

SPOJENÍ VPRAVO (RIGHT JOIN)

RIGHT JOIN vrátí všechny řádky z pravé tabulky se shodnými řádky z levé tabulky.
Řádky bez shody jsou vyplněny hodnotami NULL.
RIGHT JOIN se také nazývá RIGHT OUTER JOIN.

```
SELECT *
FROM toy
RIGHT JOIN cat
  ON toy.cat_id = cat.cat_id;
```

toy_id	toy_name	cat_id	cat_id	cat_name
5	ball	1	1	Kitty
3	mouse	1	1	Kitty
NULL	NULL	NULL	2	Hugo
1	ball	3	3	Sam
4	mouse	4	4	Misty

whole right table

PLNÉ SPOJENÍ (FULL JOIN)

FULL JOIN vrátí všechny řádky z levé tabulky a všechny řádky z pravé tabulky.
Vyplní neodpovídající řádky hodnotami NULL.
FULL JOIN se také nazývá FULL OUTER JOIN.

```
SELECT *
FROM toy
FULL JOIN cat
  ON toy.cat_id = cat.cat_id;
```

toy_id	toy_name	cat_id	cat_id	cat_name
5	ball	1	1	Kitty
3	mouse	1	1	Kitty
NULL	NULL	NULL	2	Hugo
1	ball	3	3	Sam
4	mouse	4	4	Misty
2	spring	NULL	NULL	NULL

whole left table | whole right table

KŘÍŽOVÉ SPOJENÍ (CROSS JOIN)

CROSS JOIN vrátí všechny možné kombinace řádků z levé a pravé tabulky.

```
SELECT *
FROM toy
CROSS JOIN cat;
```

Další možnost zápisu:

```
SELECT *
FROM toy, cat;
CROSS JOIN
```

toy_id	toy_name	cat_id	cat_id	cat_name
1	ball	3	1	Kitty
2	spring	NULL	1	Kitty
3	mouse	1	1	Kitty
4	mouse	4	1	Kitty
5	ball	1	1	Kitty
1	ball	3	2	Hugo
2	spring	NULL	2	Hugo
3	mouse	1	2	Hugo
4	mouse	4	2	Hugo
5	ball	1	2	Hugo
1	ball	3	3	Sam
...	...	...	...	...

PŘEZDÍVKY SLOUPCŮ A TABULEK

Přezdívky (aliases) dávají dočasný název tabulce nebo sloupci v tabulce.

CAT AS c

cat_id	cat_name	mom_id	owner_id
1	Kitty	5	1
2	Hugo	1	2
3	Sam	2	2
4	Misty	1	NULL

OWNER AS o

id	name
1	John Smith
2	Danielle Davis

Alias sloupce přejmenuje sloupec ve výsledku. Alias tabulky přejmenuje tabulku v dotazu. Pokud definujete alias tabulky, musíte jej použít místo názvu tabulky všude v dotazu. Klíčové slovo AS je při definování aliasů volitelné.

```
SELECT
  o.name AS owner_name,
  c.cat_name
FROM cat  AS c
JOIN owner AS o
  ON c.owner_id = o.id;
```

cat_name	owner_name
Kitty	John Smith
Sam	Danielle Davis
Hugo	Danielle Davis

PROPOJENÍ TABULEK (SELF JOIN)

Můžete připojit tabulky k sobě, například, abyste ukázali vztah rodič-dítě.

CAT AS child

cat_id	cat_name	owner_id	mom_id
1	Kitty	1	5
2	Hugo	2	1
3	Sam	2	2
4	Misty	NULL	1

CAT AS mom

cat_id	cat_name	owner_id	mom_id
1	Kitty	1	5
2	Hugo	2	1
3	Sam	2	2
4	Misty	NULL	1

Každý výskyt tabulky musí mít jiný alias. Před každým odkazem na sloupec musí být uveden příslušný alias tabulky.

```
SELECT
  child.cat_name AS child_name,
  mom.cat_name  AS mom_name
FROM cat AS child
JOIN cat AS mom
  ON child.mom_id = mom.cat_id;
```

child_name	mom_name
Hugo	Kitty
Sam	Hugo
Misty	Kitty

PROPOJENÍ TABULEK (NON-EQUI SELF JOIN)

Nerovnost v podmínce ON můžete použít například k zobrazení všech různých párů řádků.

TOY AS a

toy_id	toy_name	cat_id
3	mouse	1
5	ball	1
1	ball	3
4	mouse	4
2	spring	NULL

TOY AS b

cat_id	toy_id	toy_name
1	3	mouse
1	5	ball
3	1	ball
4	4	mouse
NULL	2	spring

```
SELECT
  a.toy_name AS toy_a,
  b.toy_name AS toy_b
FROM toy a
JOIN toy b
  ON a.cat_id < b.cat_id;
```

cat_a_id	toy_a	cat_b_id	toy_b
1	mouse	3	ball
1	ball	3	ball
1	mouse	4	mouse
1	ball	4	mouse
3	ball	4	mouse

NĚKOLIKANÁSOBNÉ PROPOJENÍ (MULTIPLE JOINS)

Můžete spojit více než dvě tabulky dohromady. Nejprve se spojí dvě tabulky, poté se k výsledku předchozího spojení připojí třetí tabulka.

TOY AS t

toy_id	toy_name	cat_id
1	ball	3
2	spring	NULL
3	mouse	1
4	mouse	4
5	ball	1

CAT AS c

cat_id	cat_name	mom_id	owner_id
1	Kitty	5	1
2	Hugo	1	2
3	Sam	2	2
4	Misty	1	NULL

OWNER AS o

id	name
1	John Smith
2	Danielle Davis

JOIN & JOIN

```
SELECT
  t.toy_name,
  c.cat_name,
  o.name AS owner_name
FROM toy t
JOIN cat c
  ON t.cat_id = c.cat_id
JOIN owner o
  ON c.owner_id = o.id;
```

toy_name	cat_name	owner_name
ball	Kitty	John Smith
mouse	Kitty	John Smith
ball	Sam	Danielle Davis

JOIN & LEFT JOIN

```
SELECT
  t.toy_name,
  c.cat_name,
  o.name AS owner_name
FROM toy t
JOIN cat c
  ON t.cat_id = c.cat_id
LEFT JOIN owner o
  ON c.owner_id = o.id;
```

toy_name	cat_name	owner_name
ball	Kitty	John Smith
mouse	Kitty	John Smith
ball	Sam	Danielle Davis
mouse	Misty	NULL

LEFT JOIN & LEFT JOIN

```
SELECT
  t.toy_name,
  c.cat_name,
  o.name AS owner_name
FROM toy t
LEFT JOIN cat c
  ON t.cat_id = c.cat_id
LEFT JOIN owner o
  ON c.owner_id = o.id;
```

toy_name	cat_name	owner_name
ball	Kitty	John Smith
mouse	Kitty	John Smith
ball	Sam	Danielle Davis
mouse	Misty	NULL
spring	NULL	NULL

PROPOJENÍ S VÍCE PODMÍNKAMI (JOIN WITH MULTIPLE CONDITIONS)

Můžete použít více podmínek JOIN pomocí klíčového slova ON jednou a klíčových slov AND tolikrát, kolikrát potřebujete.

CAT AS c

cat_id	cat_name	mom_id	owner_id	age
1	Kitty	5	1	17
2	Hugo	1	2	10
3	Sam	2	2	5
4	Misty	1	NULL	11

OWNER AS o

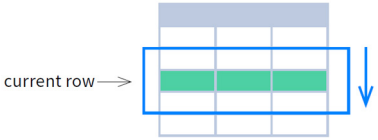
id	name	age
1	John Smith	18
2	Danielle Davis	10

```
SELECT
  cat_name,
  o.name AS owner_name,
  c.age  AS cat_age,
  o.age  AS owner_age
FROM cat c
JOIN owner o
  ON c.owner_id = o.id
 AND c.age < o.age;
```

cat_name	owner_name	age	age
Kitty	John Smith	17	18
Sam	Danielle Davis	5	10

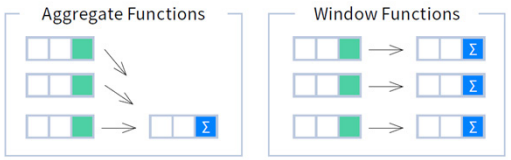
OKENÍ FUNKCE (WINDOWS FUNCTION)

Funkce okna počítají svůj výsledek na základě posuvného rámu okna, sady řádků, které nějak souvisí s aktuálním řádkem.



AGREGAČNÍ FUNKCE VS. FUNKCE OKNA

Na rozdíl od agregačních funkcí funkce okna nesbalují řádky.



ZÁPIS

```
SELECT city, month,
  sum(sold) OVER (
    PARTITION BY city
    ORDER BY month
    RANGE UNBOUNDED PRECEDING) total
FROM sales;
```

```
SELECT <column_1>, <column_2>,
  <window_function> OVER (
    PARTITION BY <...>
    ORDER BY <...>
    <window_frame>) <window_column_alias>
FROM <table_name>;
```

DEFINICE POJMENOVANÉHO OKNA

```
SELECT country, city,
  rank() OVER country_sold_avg
FROM sales
WHERE month BETWEEN 1 AND 6
GROUP BY country, city
HAVING sum(sold) > 10000
WINDOW country_sold_avg AS (
  PARTITION BY country
  ORDER BY avg(sold) DESC)
ORDER BY country, city;
```

```
SELECT <column_1>, <column_2>,
  <window_function>() OVER <window_name>
FROM <table_name>
WHERE <...>
GROUP BY <...>
HAVING <...>
WINDOW <window_name> AS (
  PARTITION BY <...>
  ORDER BY <...>
  <window_frame>)
ORDER BY <...>;
```

PARTITION BY, ORDER BY, a window frame jsou volitelné položky.

LOGICKÉ POŘADÍ OPERACÍ V SQL

1. FROM, JOIN
2. WHERE
3. GROUP BY
4. aggregate functions
5. HAVING
6. window functions
7. SELECT
8. DISTINCT
9. UNION/INTERSECT/EXCEPT
10. ORDER BY
11. OFFSET
12. LIMIT/FETCH/TOP

Můžete použít funkce okna v SELECT a ORDER BY.
Funkce okna však nemůžete umístit kamkoli do klauzulí FROM, WHERE, GROUP BY nebo HAVING.

ROZDĚLENÍ (PARTITION BY)

PARTITION BY rozděluje řádky do více skupin, nazývaných oddíly, na které je aplikována funkce okna.

month	city	sold
1	Rome	200
2	Paris	500
1	London	100
1	Paris	300
2	Rome	300
2	London	400
3	Rome	400

PARTITION BY city

month	city	sold	sum
1	Paris	300	800
2	Paris	500	800
1	Rome	200	900
2	Rome	300	900
3	Rome	400	900
1	London	100	500
2	London	400	500

Výchozí nastavení:
Bez klauzule PARTITION BY je oddílem celá výsledná sada.

SEŘADIT (ORDER BY)

ORDER BY určuje pořadí řádků v každém oddílu, na který je aplikována funkce okna.

sold	city	month
200	Rome	1
500	Paris	2
100	London	1
300	Paris	1
300	Rome	2
400	London	2
400	Rome	3

PARTITION BY city ORDER BY month

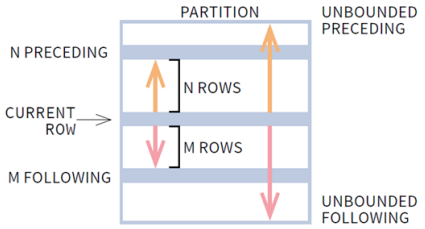
sold	city	month
300	Paris	1
500	Paris	2
200	Rome	1
300	Rome	2
400	Rome	3
100	London	1
400	London	2

Výchozí nastavení:
Bez klauzule ORDER BY je pořadí řádků v každém oddílu libovolné.

RÁM OKNA (WINDOW FRAME)

WINDOW FRAME je sada řádků, které nějak souvisí s aktuálním řádkem.
Rám okna se v rámci každé příčky hodnotí samostatně.

ŘÁDKY | ROZSAH | SKUPINY MEZI DOLNÍ A HORNÍ MEZÍ



Hranice může být kterákoli z těchto pěti možností:

- Neomezené předchozí (UNBOUNDED PRECEDING)
- Předchozí dle počtu (n PRECEDING)
- Aktuální řádek (CURRENT ROW)
- Následující dlr počtu (n FOLLOWING)
- Následující bez omezení (UNBOUNDED FOLLOWING)

Dolní hranice musí být stanovena před horní hranicí.

ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING

	city	sold	month
	Paris	300	1
	Rome	200	1
	Paris	500	2
	Rome	100	4
current row →	Paris	200	4
	Paris	300	5
	Rome	200	5
	London	200	5
	London	100	6
	Rome	300	6

1 row before the current row and 1 row after the current row

RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING

	city	sold	month
	Paris	300	1
	Rome	200	1
	Paris	500	2
	Rome	100	4
current row →	Paris	200	4
	Paris	300	5
	Rome	200	5
	London	200	5
	London	100	6
	Rome	300	6

values in the range between 3 and 5
ORDER BY must contain a single expression

GROUPS BETWEEN 1 PRECEDING AND 1 FOLLOWING

	city	sold	month
	Paris	300	1
	Rome	200	1
	Paris	500	2
	Rome	100	4
current row →	Paris	200	4
	Paris	300	5
	Rome	200	5
	London	200	5
	London	100	6
	Rome	300	6

1 group before the current row and 1 group after the current row regardless of the value

Zkratky a jejich význam

UNBOUNDED PRECEDING - mezi neohraničeným předchozím a aktuálním řádkem
n PRECEDING - mezi n předchozím a aktuálním řádkem
CURRENT ROW - mezi aktuálním řádkem a aktuálním řádkem
n FOLLOWING - mezi a aktuálním řádkem a n následujících
UNBOUNDED FOLLOWING - mezi aktuálním řádkem a neomezeným následujícím n PRECEDING

Výchozí nastavení Window Frame

Je-li specifikováno ORDER BY je rámeček mezi UNBOUNDED PRECEDING a CURRENT ROW

Bez ORDER BY specifikace rámeček mezi UNBOUNDED PRECEDING a UNBOUNDED FOLLOWING

SQL TAHÁK 6/10

SEZNAM OKENÍCH FUNKCÍ

Vyhodnocovací funkce:
- **avg()**
- **count()**
- **max()**
- **min()**
- **sum()**

Číslovací funkce:
- **row_number()**
- **rank()**
- **dense_rank()**

Distribuční funkce:
- **percent_rank()**
- **cume_dist()**

Analitické funkce:
- **lead()**
- **lag()**
- **ntile()**
- **first_value()**
- **last_value()**
- **nth_value()**

VYHODNOCOVACÍ FUNKCE (AGGREGATE F.)

- **avg(expr)**
průměrná hodnota pro řádky v rámci okna

- **count(expr)**
počet hodnot pro řádky v rámci okna

- **max(expr)**
maximální hodnota v rámu okna

- **min(expr)**
minimální hodnota v rámu okna

- **sum(expr)**
součet hodnot v rámci okna

ORDER BY a Window Frame:
Agregační funkce nevyžadují ORDER BY.
Přijímají definici okenního rámu (ROWS, RANGE, GROUPS).

ČÍSLOVACÍ FUNKCE (RANKING FUNCTION)

- **row_number()**
jedinečné číslo pro každý řádek v rámci oddílu, s různými čísly pro svázané hodnoty

- **rank()**
pořadí v rámci oddílu, s mezerami a stejným hodnocením pro nerozhodné hodnoty

- **dense_rank()**
hodnocení v rámci oddílu, bez mezer a stejné hodnocení pro shodné hodnoty

city	price	row_number	rank	dense_rank
		over(order by price)		
Paris	7	1	1	1
Rome	7	2	1	1
London	8.5	3	3	2
Berlin	8.5	4	3	2
Moscow	9	5	5	3
Madrid	10	6	6	4
Oslo	10	7	6	4

ORDER BY a Window Frame:
rank() a dense_rank() vyžadují ORDER BY, ale row_number() nevyžaduje ORDER BY.
Hodnotící funkce neakceptují definici Window Frame (ROWS, RANGE, GROUPS).

DISTRIBUČNÍ FUNKCE (DISTRIBUTION FUNCTION)

- **percent_rank()**
procentní pořadí řádku – hodnota v intervalu [0, 1]: (hodnocení-1) / (celkový počet řádků – 1)

- **cume_dist()**
kumulativní rozdělení hodnoty v rámci skupiny hodnot, tj. počet řádků s hodnotami menšími nebo rovnými hodnotě aktuálního řádku vydělený celkovým počtem řádků; hodnotu v intervalu (0, 1]

cume_dist() OVER(ORDER BY sold)

city	sold	cume_dist
Paris	100	0.2
Berlin	150	0.4
Rome	200	0.8
Moscow	200	0.8
London	300	1

80% of values are less than or equal to this one

percent_rank() OVER(ORDER BY sold)

city	sold	percent_rank
Paris	100	0
Berlin	150	0.25
Rome	200	0.5
Moscow	200	0.5
London	300	1

without this row 50% of values are less than this row's value

ORDER BY a Window Frame:
Distribuční funkce vyžadují ORDER BY. Neakceptují definici Window Frame (ROWS, RANGE, GROUPS).

ANALITICKÉ FUNKCE (ANALYTIC FUNCTION)

- **lead(expr, offset, default)**
hodnota pro řádky posunuté za aktuální; offset a default jsou volitelné;
výchozí hodnoty: offset = 1, výchozí = NULL

- **lag(expr, offset, default)**
hodnota pro řádky posunuté před aktuální; offset a default jsou volitelné;
výchozí hodnoty: offset = 1, výchozí = NULL

lag(sold) OVER(ORDER BY month)

month	sold	
1	500	NULL
2	300	500
3	400	300
4	100	400
5	500	100

order by month

lead(sold) OVER(ORDER BY month)

month	sold	
1	500	300
2	300	400
3	400	100
4	100	500
5	500	NULL

order by month

lag(sold, 2, 0) OVER(ORDER BY month)

month	sold	
1	500	0
2	300	0
3	400	500
4	100	300
5	500	400

order by month, offset=2

lead(sold, 2, 0) OVER(ORDER BY month)

month	sold	
1	500	400
2	300	100
3	400	500
4	100	0
5	500	0

order by month, offset=2

- **ntile(n)**
rozdělit řádky v rámci oddílu co nejrovnoměrněji do n skupin a každému řádku přiřadit číslo skupiny.

ntile(3)

city	sold	
Rome	100	1
Paris	100	1
London	200	1
Moscow	200	2
Berlin	200	2
Madrid	300	2
Oslo	300	3
Dublin	300	3

ORDER BY a Window Frame:
ntile(), lead() a lag() vyžadují ORDER BY. Neakceptují definici Window Frame (ROWS, RANGE, GROUPS).

- **first_value(expr)**
hodnota pro první řádek v rámci okna

- **last_value(expr)**
hodnota pro poslední řádek v rámci okna

first_value(sold) OVER
(PARTITION BY city ORDER BY month)

city	month	sold	first_value
Paris	1	500	500
Paris	2	300	500
Paris	3	400	500
Rome	2	200	200
Rome	3	300	200
Rome	4	500	200

last_value(sold) OVER
(PARTITION BY city ORDER BY month
RANGE BETWEEN UNBOUNDED PRECEDING
AND UNBOUNDED FOLLOWING)

city	month	sold	last_value
Paris	1	500	400
Paris	2	300	400
Paris	3	400	400
Rome	2	200	500
Rome	3	300	500
Rome	4	500	500

Poznámka: Obvykle chcete RANGE BETWEEN NEBOUNDED PRECEDING A NEBOUNDED FOLLOWING použít s last_value() . S výchozím rámcem okna pro ORDER BY, RANGE NEBOUNDED PRECEDING, last_value() vrací hodnotu pro aktuální řádek.

- **nth_value(expr, n)**
the value for the n-th row within the window frame; n must be an integer

nth_value(sold, 2) OVER
(PARTITION BY city ORDER BY month)

city	month	sold	nth_value
Paris	1	500	300
Paris	2	300	300
Paris	3	400	300
Rome	2	200	300
Rome	3	300	300
Rome	4	500	300
Rome	5	300	300
London	1	100	NULL

ORDER BY a Window Frame:
first_value(), last_value() a nth_value() nevyžadují ORDER BY. Přijímají definici okenního rámu (ROWS, RANGE, GROUPS).

FUNKCE PRO TEXT

|| - spojení textů a čísel do jednoho výsledného řetězce
Použij dvě rovné čáry **||**, jako operátor pro spojení dvou řetězců:

```
SELECT ‚Hi , || ‚there!‘;
-- výsledek: Hi there!
```

K řetězcům můžeš takto přidávat i čísla:

```
SELECT ‚‘ || 4 || 2;
-- výsledek: 42
```

Některé databáze implementují nestandardní řešení pro zřetězení řetězců, jako je CONCAT() nebo CONCAT_WS(). Proto je dobré zkontrolovat dokumentaci pro vaši konkrétní databázi.

LIKE - hedání a porovnání části řetězce
Pomocí znaku _ nahraďte libovolný jednotlivý znak.
Pomocí znaku % nahraďte libovolný počet znaků (včetně 0 znaků).

Vyhledej všechna jména, která začínají libovolným písmenem následovaným „atherine“:

```
SELECT name
FROM names
WHERE name LIKE ‚_atherine‘;
```

Načti všechna jména, která končí na „a“:

```
SELECT name
FROM names
WHERE name LIKE ‚%a‘;
```

LENGTH - vrátí hodnotu počtu znaků v řetězci:

```
SELECT LENGTH(‚LearnSQL.com‘);
-- výsledek: 12
```

LOWER - převede všechna písmena na malá:

```
SELECT LOWER(‚LEARNSQL.COM‘);
-- výsledek: learnsql.com
```

UPPER - převede všechna písmena na velká:

```
SELECT UPPER(‚LearnSQL.com‘);
-- výsledek: LEARNSQL.COM
```

INITCAP - převeďte všechna písmena na malá a všechna první písmena na velká (není implementováno v MySQL a SQL Server):

```
SELECT INITCAP(‚edgar frank ted cODD‘);
-- výsledek: Edgar Frank Ted Codd
```

SUBSTRING - Získání části řetězce:

```
SELECT SUBSTRING(‚LearnSQL.com‘, 9);
-- výsledek: .com

SELECT SUBSTRING(‚LearnSQL.com‘, 0, 6);
-- výsledek: Learn
```

REPLACE - nahrazení části řetězce:

```
SELECT REPLACE(‚LearnSQL.com‘, ‚SQL‘,
‚Python‘);
-- výsledek: LearnPython.com
```

FUNKCE PRO ČÍSLA

ZÁKLADNÍ OPERACE

oužijte +, -, *, / k základním početním operacím
Chcete-li získat počet sekund za týden:

```
SELECT 60 * 60 * 24 * 7; -- výsledek: 604800
```

CASTING - změní typ čísla
Funkce CAST() umožňuje změnit typ hodnoty téměř na cokoliv (celé číslo, číselné hodnoty, dvojitá přesnost, varchar a mnoho dalších).
Získejte číslo jako celé číslo (bez zaokrouhlení):

```
SELECT CAST(1234.567 AS integer);
-- výsledek: 1234
```

Změňte typ sloupce na dvojnásobnou přesnost:

```
SELECT CAST(column AS double precision);
```

MOD - vrátí zbytek po dělení

```
SELECT MOD(13, 2);
-- výsledek: 1
```

ROUND - zaokrouhlení
Zaokrouhlení čísla na nejbližší celé číslo:

```
SELECT ROUND(1234.56789);
-- výsledek: 1235
```

Zaokrouhlení čísla na tři desetinná místa:

```
SELECT ROUND(1234.56789, 3);
-- výsledek: 1234.568
```

PostgreSQL vyžaduje, aby první argument byl typu numeric – v případě potřeby je možné číslo přetypovat.

CEIL - zaokrouhlení směrem nahoru:

```
SELECT CEIL(13.1); -- výsledek: 14
SELECT CEIL(-13.9); -- výsledek: -13
```

Funkce CEIL(x) vrací nejmenší celé číslo, které není menší než x. V SQL Server se funkce nazývá CEILING().

FLOOR - zaokrouhlení směrem dolu:

```
SELECT FLOOR(13.8); -- výsledek: 13
SELECT FLOOR(-13.2); -- výsledek: -14
```

Funkce FLOOR(x) vrací největší celé číslo, které není větší než x.

TRUNC - zaokrouhlení směrem k 0 bez ohledu na znaménko čísla:

```
SELECT TRUNC(13.5); -- výsledek: 13
SELECT TRUNC(-13.5); -- výsledek: -13
```

TRUNC(x) funguje stejným způsobem jako CAST(x AS integer). V MySQL se funkce nazývá TRUNCATE().

ABS - získání absolutní hodnoty čísla:

```
SELECT ABS(-12); -- výsledek: 12
```

SQRT - získání druhé odmocniny čísla

```
SELECT SQRT(9); -- výsledek: 3
```

NULL

Chcete-li načíst všechny řádky s chybějící hodnotou ve sloupci ceny:

```
WHERE cena IS NULL
```

Chcete-li načíst všechny řádky s vyplněným sloupcem hmotnosti:

```
WHERE vaha IS NOT NULL
```

Proč byste neměli použít cena = NULL nebo vaha != NULL? Protože databáze nevědí, zda jsou tyto výrazy pravdivé nebo nepravdivé, jsou tak vyhodnoceny jako hodnoty NULL.
Navíc, pokud použijete funkci nebo zřetězení na sloupec, který má v některých řádcích hodnotu NULL, bude se šířit. Podívej se:

domain	LENGTH(domain)
LearnSQL.com	12
LearnPython.com	15
NULL	*NULL*
vertabelo.com	13

COALESCE(x, y, ...) - nahrazení hodnoty NULL ve výstupu z databáze:

```
SELECT
 domain,
 COALESCE(domain, ‚domain missing‘)
FROM contacts;
```

domain	coalesce
LearnSQL.com	LearnSQL.com
NULL	domain missing

Funkce COALESCE() přebírá libovolný počet argumentů a vrací hodnotu prvního argumentu, který není NULL.

NULLIF(x, y) - ochrana před dělením nulou:

```
SELECT
 last_month,
 this_month,
 this_month * 100.0
  / NULLIF(last_month, 0)
  AS better_by_percent
FROM video_views;
```

last_month	this_month	better_by_percent
723786	1085679	150.0
0	178123	*NULL*

Funkce NULLIF(x, y) vrátí NULL, pokud je x stejné jako y, jinak vrátí hodnotu x.

CASE - WHEN

Základní verze CASE WHEN kontroluje, zda se hodnoty rovnají (např. pokud je poplatek roven 50, vrátí se ‚normální‘). Pokud v CASE WHEN není odpovídající hodnota, bude vrácena ELSE hodnota (např. pokud je poplatek roven 49, zobrazí se: není k dispozici.

```
SELECT
 CASE poplatek
  WHEN 50 THEN ‚normalní cena‘
  WHEN 10 THEN ‚sleva‘
  WHEN  0 THEN ‚zadarmo‘
  ELSE ‚není k dispozici‘
 END AS tariff
FROM ticket_types;
```

Nejoblíbenějším typem je hledaný CASE WHEN – ten vám umožní předat podmínky (jak byste je napsali do klauzule WHERE), vyhodnotí je v pořadí a poté vrátí hodnotu pro první splněnou podmínku.

```
SELECT
 CASE
  WHEN score >= 90 THEN ‚A‘
  WHEN score >  60 THEN ‚B‘
  ELSE ‚F‘
 END AS grade
FROM test_results;
```

Zde všichni studenti, kteří dosáhli alespoň 90 bodů, dostanou A, ti se skóre vyšším než 60 (a nižším než 90) dostanou B a zbytek obdrží F.

ŘEŠENÍ PROBLÉMŮ

Celočíselné dělení

Když nevidíte desetinná místa, která očekáváte, znamená to, že dělíte dvě celá čísla. Přenést jedničku na desetinné číslo:`CAST(123 AS decimal) / 2`

Dělení nulou

Chcete-li se této chybě vyhnout, ujistěte se, že jmenovatel není roven 0. Pomocí funkce NULLIF() můžete nahradit 0 hodnotou NULL, což bude mít za následek hodnotu NULL pro celý výraz:
`count / NULLIF(count_all, 0)`

Nepřesné výpočty

Pokud provádíte výpočty pomocí reálných čísel (s plovoucí desetinnou čárkou), skončíte s některými nepřesnostmi. Je to proto, že tento typ je určen pro vědecké výpočty, jako je například výpočet rychlosti. Kdykoli potřebujete přesnost (například práci s peněžními hodnotami), použijte desítkový / číselný typ (nebo peníze, pokud jsou k dispozici).

Chyby při zaokrouhlování se zadanou přesností

Většina databází si nebude stěžovat, ale pokud ano, zkontrolujte dokumentaci. Pokud například chcete určit přesnost zaokrouhlení v PostgreSQL, hodnota musí být číselného typu.

AGREGACE A SKUPENÍ

- **COUNT(expr)** – počet hodnot pro řádky ve skupině
- **SUM(expr)** – součet hodnot v rámci skupiny
- **AVG(expr)** – průměrnou hodnotu pro řádky ve skupině
- **MIN(expr)** – minimální hodnotu v rámci skupiny
- **MAX(expr)** – maximální hodnotu v rámci skupiny

Chcete-li získat počet řádků v tabulce:

```
SELECT COUNT(*)
FROM city;
```

Chcete-li získat počet hodnot ve sloupci, které nejsou NULL:

```
SELECT COUNT(rating)
FROM city;
```

Chcete-li získat počet jedinečných hodnot ve sloupci:

```
SELECT COUNT(DISTINCT country_id)
FROM city;
```

GROUP BY

CITY	
name	**country_id**
Paris	1
Marseille	1
Lyon	1
Berlin	2
Hamburg	2
Munich	2
Warsaw	4
Cracow	4

→

CITY	
country_id	**count**
1	3
2	3
4	2

Výše uvedený příklad – počet měst v každé zemi:

```
SELECT name, COUNT(country_id)
FROM city
GROUP BY name;
```

Průměrné hodnocení města:

```
SELECT city_id, AVG(rating)
FROM ratings
GROUP BY city_id;
```

Časté chyby: COUNT(*) a LEFT JOIN

Když se připojíte k tabulkám takto: klient LEFT JOIN projekt a chcete získat počet projektů pro každého klienta, kterého znáte, COUNT(*) vrátí 1 pro každého klienta, i když jste pro něj nikdy nepracovali. Je to proto, že jsou stále přítomni v seznamu, ale s NULL v polích souvisejících s projektem po JOIN. Chcete-li získat správný počet (namísto NULL, číslo nula pro klienty, pro které jste nikdy nepracovali), spočítejte hodnoty ve sloupci druhé tabulky, např. COUNT(název_projektu).

DATUM A ČAS

Existují 3 hlavní typy související s časem: datum, čas a časové razítko. Čas je vyjádřen pomocí 24hodinových hodin a může být tak vágní jako pouhá hodina a minuty (např. 15:30 – 15:30) nebo tak přesný jako mikrosekundy a časové pásmo (jak je uvedeno níže):

```
2021-12-31 14:39:53.662522-05
|  date  | |      time      |
|          timestamp         |
YYYY-mm-dd HH:MM:SS.ssssss±TZ
```

14:39:53.662522-05 je téměř 14:40. CDT (např. v Chicagu; v UTC by to bylo 19:40). Písmena ve výše uvedeném příkladu představují:

V části pro den:

- **YYYY** – rok (formát: 4 číslice).
- **mm** – měsíc (formát: 2číslice, 1 číslice s nulou nazačátku)
- **dd** – den (formát: 2číslice, 1 číslice s nulou nazačátku)

V části pro čas:

- **HH** – hodiny (24h formát, 2číslice, 1 číslice s nulou nazačátku)
- **MM** – minuty (formát: 2číslice, 1 číslice s nulou nazačátku)
- **SS** – vteřiny (formát: 2číslice, 1 číslice s nulou nazačátku)
- **ssssss** – menší části sekundy – lze je vyjádřit pomocí 1 až 6 číslic. (Můžeme vynechat)
- **±TZ** – časové pásmo. Musí začínat + nebo - a používat dvě číslice vzhledem k UTC. (Můžeme vynechat)

Kolik je hodin?
Chcete-li odpovědět na tuto otázku v SQL, můžete použít:

- **CURRENT_TIME** – zjistit, kolik je hodin.
- **CURRENT_DATE** – chcete-li zjistit dnešní datum. (GETDATE() na serveru SQL Server.)
- **CURRENT_TIMESTAMP** – chcete-li získat časové razítko s dvěmi výše uvedenými hodnotami.

Vytváření hodnot

Chcete-li vytvořit datum, čas nebo časové razítko, jednoduše napište hodnotu jako řetězec a přetypujte ji na správný typ.

```
SELECT CAST(‚2021-12-31‘ AS date);
SELECT CAST(‚15:31‘ AS time);
SELECT CAST(‚2021-12-31 23:59:29+02‘ AS
timestamp);
SELECT CAST(‚15:31.124769‘ AS time);
```

Pozor na poslední příklad – bude interpretován jako 15 minut 31 sekund a 124769 mikrosekund! Vždy je dobré napsat oo výslovně pro hodiny: ‚00:15:31.124769‘.

V jednoduchých podmínkách můžete přeskočit CASTing – databáze bude vědět, co má dělat:

```
SELECT airline, flight_number, departure_
time
FROM airport_schedule
WHERE departure_time < ‚12:00‘;
```

INTERVALY

Poznámka: V SQL Serveru nejsou intervaly implementovány – použijte funkce DATEADD() a DATEDIFF().

Chcete-li získat nejjednodušší interval, odečtěte jednu časovou hodnotu od druhé:

```
SELECT CAST(‚2021-12-31 23:59:59‘ AS
timestamp) - CAST(‚2021-06-01 12:00:00‘ AS
timestamp);
-- výsledek: 213 days 11:59:59
```

Chcete-li definovat interval: INTERVAL ‚1‘ DAY
Tento zápis se skládá ze tří prvků: klíčového slova INTERVAL, hodnoty v uvozovkách a klíčového slova časové části (v jednotném čísle). Můžete použít následující časové části: YEAR, MONTH, WEEK, DAY, HOUR, MINUTE a SECOND. V MySQL vynechejte uvozovky. Pomocí operátoru + nebo - se můžete připojit k mnoha různým INTERVALům:

```
INTERVAL ‚1‘ YEAR + INTERVAL ‚3‘ MONTH
```

V některých databázích existuje jednodušší způsob, jak získat výše uvedenou hodnotu. A přijímá tvary v množném čísle!

```
INTERVAL ‚1 year 3 months‘
```

Ve standardním SQL existují dvě další syntaxe:

Syntax	What it does
`INTERVAL 'x-y' YEAR TO MONTH`	`INTERVAL 'x year y month'`
`INTERVAL 'x-y' DAY TO SECOND`	`INTERVAL 'x day y second'`

V MySQL, napište year_month namísto YEAR TO MONTH, a day_second namísto DAY TO SECOND.

Chcete-li získat poslední den v měsíci, přidejte jeden měsíc a odečtěte jeden den:

```
SELECT CAST(‚2021-02-01‘ AS date)
    + INTERVAL ‚1‘ MONTH
    - INTERVAL ‚1‘ DAY;
```

Chcete-li získat všechny události na příští tři měsíce ode dneška:

```
SELECT event_date, event_name
FROM calendar
WHERE event_date BETWEEN CURRENT_
DATE AND CURRENT_DATE + INTERVAL ‚3‘
MONTH;
```

Chcete-li získat část datumu:

```
SELECT EXTRACT(YEAR FROM birthday)
FROM artists;
```

Jedna z možných vrácených hodnot: 1946. V SQL Server použijte funkci DATEPART(část, datum).

ČASOVÁ PÁSMA

Ve standardu SQL nemůže mít typ data přidružené časové pásmo, ale typy času a časového razítka ano. Ve skutečném světě nemají časová pásma bez data velký význam, protože posun se může v průběhu roku měnit kvůli letnímu času. Nejlepší je tedy pracovat s hodnotami časového razítka.

Při práci s časovým razítkem typu s časovou zónou (zkr. timestamptz) můžete zadat hodnotu ve svém místním časovém pásmu a ta se po vložení do tabulky převede na časové pásmo UTC. Později, když ji vyberete z tabulky, převede se zpět na vaše místní časové pásmo. Toto je imunní vůči změnám časového pásma.

AT TIME ZONE

Chcete-li pracovat mezi různými časovými pásmy, použijte klíčové slovo AT TIME ZONE.

Pokud použijete tento formát:

```
{timestamp without time zone} AT TIME ZONE
{time zone}
```

pak databáze načte časové razítko v určeném časovém pásmu a převede ho na časové pásmo místní zobrazení.
Vrací čas ve formátu časové značky s časovou zónou.

Pokud použijete tento formát:

```
{timestamp with time zone} AT TIME ZONE
{time zone}
```

pak databáze převede čas v jednom časovém pásmu do cílového časového pásma určeného AT TIME ZONE. Vrací čas ve formátu časové značky bez časového pásma, v cílovém časovém pásmu.

Časové pásmo můžete definovat pomocí oblíbených zkratek: UTC, MST nebo GMT,
nebo podle kontinentu/města, jako je: America/New_York, Europe/London, Asia/Tokyo.

Příklady

Místní časové pásmo jsme nastavili na ‚Amerika/New_York‘:

```
SELECT TIMESTAMP ‚2021-07-16 21:00:00‘ AT
TIME ZONE ‚America/Los_Angeles‘;
-- výsledek: 2021-07-17 00:00:00-04
```

Zde databáze bere časové razítko bez časového pásma a říká se, že je v čase Los Angeles, který se pak pro zobrazení převede na místní čas – New York. To odpovídá na otázku „V kolik hodin mám zapnout televizi, když show začíná v 21:00 v Los Angeles?“

```
SELECT TIMESTAMP WITH TIME ZONE ‚2021-
06-20 19:30:00‘ AT TIME ZONE ‚Australia/
Sydney‘;
-- výsledek: 2021-06-21 09:30:00
```

Zde databáze získá časové razítko specifikované v místním časovém pásmu a převede ho na čas v Sydney (všimněte si, že nevrátilo časové pásmo.) To odpovídá na otázku „Kolik je hodin v Sydney, když je 7 :30 tady?“

TAHÁK NA MySQL - VÝCUC

MySQL je populární open-source systém pro správu relačních databází známý pro své snadné použití a škálovatelnost. Někdy budete potřebovat malou pomoc při práci na projektu. Zde je výcuc toho nejdůležitějšího.

PŘIPOJENÍ K MYSQL SERVERU

Připojte se k serveru MySQL pomocí uživatelského jména a hesla pomocí klienta příkazového řádku mysql.
MySQL vás vyzve k zadání hesla:

```
mysql -u [username] -p
```

Chcete-li se připojit ke konkrétní databázi na serveru MySQL pomocí uživatelského jména a hesla:

```
mysql -u [username] -p [database]
```

Export dat pomocí nástroje mysqldump:

```
mysqldump -u [username] -p \
 [database] > data_backup.sql
```

Chcete-li klienta opustit:

```
quit or exit
```

Úplný seznam příkazů:

```
help
```

VYTVÁŘENÍ A ZOBRAZOVÁNÍ DATABÁZÍ

Chcete-li vytvořit databázi:

```
CREATE DATABASE zoo;
```

Seznam všech databází na serveru:

```
SHOW DATABASES;
```

Chcete-li použít zadanou databázi:

```
USE zoo;
```

Chcete-li odstranit zadanou databázi:

```
DROP DATABASE zoo;
```

Chcete-li vypsat všechny tabulky v databázi:

```
SHOW TABLES;
```

Chcete-li získat informace o konkrétní tabulce:

```
DESCRIBE animal;
```

Vypisuje názvy sloupců, datové typy, výchozí hodnoty a další informace o tabulce.

VYTVÁŘENÍ TABULEK

Chcete-li vytvořit tabulku:

```
CREATE TABLE habitat (
 id INT,
 name VARCHAR(64)
);
```

Použijte AUTO_INCREMENT k automatickému zvýšení ID s každým novým záznamem. Sloupec AUTO_INCREMENT musí být definován jako primární nebo jedinečný klíč:

```
CREATE TABLE habitat (
 id INT PRIMARY KEY AUTO_INCREMENT,
 name VARCHAR(64)
);
```

Chcete-li vytvořit tabulku s cizím klíčem:

```
CREATE TABLE animal (
 id INT PRIMARY KEY AUTO_INCREMENT,
 name VARCHAR(64),
 species VARCHAR(64),
 age INT,
 habitat_id INT,
 FOREIGN KEY (habitat_id)
  REFERENCES habitat(id)
);
```

ÚPRAVA TABULEK

Use the ALTER TABLE statement to modify the table structure.

Chcete-li změnit název tabulky:

```
ALTER TABLE animal RENAME pet;
```

Postup přidání sloupce do tabulky:

```
ALTER TABLE animal
ADD COLUMN name VARCHAR(64);
```

Chcete-li změnit název sloupce:

```
ALTER TABLE animal
RENAME COLUMN id TO identifier;
```

Chcete-li změnit typ dat sloupce:

```
ALTER TABLE animal
MODIFY COLUMN name VARCHAR(128);
```

Chcete-li odstranit sloupec:

```
ALTER TABLE animal
DROP COLUMN name;
```

Postup smazání tabulky:

```
DROP TABLE animal;
```

DOTAZOVÁNÍ NA DATA

Chcete-li vybrat data z tabulky, použijte příkaz SELECT.
Příklad dotazu s jednou tabulkou:

```
SELECT species, AVG(age) AS average_age
FROM animal
WHERE id != 3
GROUP BY species
HAVING AVG(age) > 3
ORDER BY AVG(age) DESC;
```

Příklad dotazu s více tabulkami:

```
SELECT city.name, country.name
FROM city
[INNER | LEFT | RIGHT] JOIN country
 ON city.country_id = country.id;
```

Použijte +, -, *, / k základnímu počítání.
Chcete-li získat počet sekund za týden:

```
SELECT 60 * 60 * 24 * 7; -- výsledek: 604800
```

AGREGACE A SKUPENÍ

AVG(expr) – průměrná hodnota expr pro skupinu.
COUNT(expr) – počet hodnot výrazu v rámci skupiny.
MAX(expr) – maximální hodnota hodnot expr v rámci skupiny.
MIN(expr) – minimální hodnota hodnot expr v rámci skupiny.
SUM(expr) – součet hodnot expr v rámci skupiny.

Chcete-li spočítat řádky v tabulce:

```
SELECT COUNT(*)
FROM animal;
```

Chcete-li spočítat hodnoty, které nejsou NULL ve sloupci:

```
SELECT COUNT(name)
FROM animal;
```

Chcete-li počítat jedinečné hodnoty ve sloupci:

```
SELECT COUNT(DISTINCT name)
FROM animal;
GROUP BY
```

Jak spočítat zvířata podle druhů:

```
SELECT species, COUNT(id)
FROM animal
GROUP BY species;
```

Chcete-li získat průměrný, minimální a maximální věk podle stanoviště:

```
SELECT habitat_id, AVG(age),
    MIN(age), MAX(age)
FROM animal
GROUP BY habitat;
```

VLOŽENÍ DAT

Chcete-li vložit data do tabulky, použijte příkaz INSERT:

```
INSERT INTO habitat VALUES
(1, ‚River‘),
(2, ‚Forest‘);
```

Můžete určit sloupce, do kterých se data přidávají. Zbývající sloupce jsou vyplněny výchozími hodnotami nebo hodnotami NULL.

```
INSERT INTO habitat (name) VALUES
(‚Savanna‘);
```

AKTUALIZACE DAT

Chcete-li aktualizovat data v tabulce, použijte příkaz UPDATE:

```
UPDATE animal
SET
 species = ‚Duck‘,
 name = ‚Quack‘
WHERE id = 2;
```

VYMAZÁNÍ DAT

Chcete-li odstranit data z tabulky, použijte příkaz DELETE:

```
DELETE FROM animal
WHERE id = 1;
```

Tím se odstraní všechny řádky splňující podmínku WHERE.
Chcete-li odstranit všechna data z tabulky, použijte příkaz

```
TRUNCATE TABLE:
TRUNCATE TABLE animal;
```

PŘETYPOVÁNÍ

Čas od času je potřeba změnit typ hodnoty.
K tomu použijte funkci CAST().
V MySQL můžete přetypovat na tyto datové typy:

CHAR	NCHAR	BINARY	DATE
DATETIME	DECIMAL	DOUBLE	FLOAT
REAL	SIGNED	UNSIGNED	TIME
YEAR	JSON	spatial_type	

Chcete-li získat číslo jako celé číslo se znaménkem:

```
SELECT CAST(1234.567 AS signed);
-- výsledek: 1235
```

Chcete-li změnit typ sloupce na dvojitý:

```
SELECT CAST(column AS double);
```

TEXTOVÉ FUNKCE

FILTROVÁNÍ VÝSTUPU

Chcete-li načíst názvy měst, které nejsou Berlín:

```
SELECT name
FROM city
WHERE name != ‚Berlin‘;
```

TEXTOVÉ OPERÁTORY

Chcete-li načíst názvy měst, které začínají na „P“ nebo končí na „s“:

```
SELECT name
FROM city
WHERE name LIKE ‚P%‘ OR name LIKE ‚%s‘;
```

Chcete-li načíst názvy měst, které začínají libovolným písmenem, za kterým následuje „ublin“ (jako Dublin v Irsku nebo Lublin v Polsku):

```
SELECT name
FROM city
WHERE name LIKE ‚_ublin‘;
```

ZŘETĚZENÍ

Pomocí funkce CONCAT() zřetězit dva řetězce:

```
SELECT CONCAT(‚Hi ‚, ‚there!‘);
-- výsledek: Hi there!
```

Pokud je některý z řetězců NULL, výsledek je NULL:

```
SELECT CONCAT(Great ‚, ‚day‘, NULL);
-- výsledek: NULL
```

MySQL umožňuje specifikovat oddělovací znak (separátor) pomocí funkce CONCAT_WS(). Oddělovač je umístěn mezi zřetězené hodnoty:

```
SELECT CONCAT_WS(, ‚, 1, ‚Olivier‘, ‚Norris‘);
-- výsledek: 1 Olivier Norris
```

DALŠÍ UŽITEČNÉ FUNKCE TEXTU

Chcete-li zjistit počet znaků v řetězci:

```
SELECT LENGTH(‚LearnSQL.com‘);
-- výsledek: 12
```

Chcete-li převést všechna písmena na malá písmena:

```
SELECT LOWER(‚LEARNSQL.COM‘);
-- výsledek: learnsql.com
```

Chcete-li převést všechna písmena na velká:

```
SELECT UPPER(‚LearnSQL.com‘);
-- výsledek: LEARNSQL.COM
```

Chcete-li získat pouze část řetězce:

```
SELECT SUBSTRING(‚LearnSQL.com‘, 9);
-- výsledek: .com
SELECT SUBSTRING(‚LearnSQL.com‘, 1, 5);
-- výsledek: Learn
```

Chcete-li nahradit část řetězce:

```
SELECT REPLACE(‚LearnSQL.com‘, ‚SQL‘,
‚Python‘);
-- výsledek: LearnPython.com
```

NUMERICKÉ FUNKCE

Chcete-li získat zbytek dělení:

```
SELECT MOD(13, 2); -- výsledek: 1
```

Zaokrouhlení čísla na nejbližší celé číslo:

```
SELECT ROUND(1234.56789); -- výsledek:
1235
```

Zaokrouhlení čísla na tři desetinná místa:

```
SELECT ROUND(1234.56789, 3);
-- výsledek: 1234.568
```

Zaokrouhlení čísla nahoru:

```
SELECT CEIL(13.1); -- výsledek: 14
SELECT CEIL(-13.9); -- výsledek: -13
```

Funkce CEIL(x) vrací nejmenší celé číslo, které není menší než x.
Zaokrouhlení čísla dolů:

```
SELECT FLOOR(13.8); -- výsledek: 13
SELECT FLOOR(-13.2); -- výsledek: -14
```

Funkce FLOOR(x) vrací největší celé číslo, které není větší než x.
Zaokrouhlení směrem k o bez ohledu na znaménko čísla:

```
SELECT TRUNCATE(13.56, 0); -- výsledek: 13
SELECT TRUNCATE(-13.56, 1); -- výsledek:
-13.5
```

Chcete-li získat absolutní hodnotu čísla:

```
SELECT ABS(-12); -- výsledek: 12
```

Chcete-li získat druhou odmocninu čísla:

```
SELECT SQRT(9); -- výsledek: 3
```

UŽITEČNÉ FUNKCE NULL

Chcete-li načíst názvy měst, jejichž hodnoty hodnocení nechybí:

```
SELECT name
FROM city
WHERE rating IS NOT NULL;
```

COALESCE(x; y; ...)

Chcete-li nahradit NULL v dotazu něčím smysluplným:

```
SELECT domain,
    COALESCE(domain, ‚domain missing‘)
FROM contacts;
```

Funkce COALESCE() přebírá libovolný počet argumentů a vrací hodnotu prvního argumentu, který není NULL.

NULLIF(x, y)

Chcete-li se zachránit před dělením o chybami:

```
SELECT last_month, this_month,
 this_month * 100.0
  / NULLIF(last_month, 0)
  AS better_by_percent
FROM video_views;
```

Funkce NULLIF(x, y) vrátí NULL, pokud se x rovná y, jinak vrátí hodnotu hodnoty x.

DATUM A ČAS

V MySQL existuje 5 hlavních typů souvisejících s časem:
DATE TIME DATETIME TIMESTAMP YEAR

DATE – ukládá rok, měsíc a den ve formátu RRRR-MM-DD.

TIME – ukládá hodiny, minuty a sekundy ve formátu HH:MM:SS.

DATETIME – ukládá datum a čas ve formátu RRRR-MM-DD HH:MM:SS. Podporovaný rozsah je ‚1000-01-01 00:00:00‘ až ‚9999-12-31 23:59:59‘.

TIMESTAMP – ukládá datum a čas. Rozsah je ‚1970-01-01 00:00:01‘ UTC až ‚2038-01-19 03:14:07‘ UTC. MySQL převádí hodnoty TIMESTAMP z aktuálního časového pásma na UTC pro uložení a zpět z UTC do aktuálního časového pásma pro vyhledání.

YEAR – ukládá rok ve formátu RRRR.

INTERVALY

Interval je doba mezi dvěma časovými body.
Chcete-li definovat interval: `INTERVAL 1 DAY`

Tato syntaxe se skládá z klíčového slova INTERVAL, hodnoty a klíčového slova časové části (YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, MICROSECOND).

Pomocí operátoru + nebo - můžete kombinovat různé INTERVALY:

```
INTERVAL 1 YEAR + INTERVAL 3 MONTH
```

Můžete také použít standardní syntaxi SQL:

```
INTERVAL ‚1-3‘ YEAR_MONTH
-- 1 year and 3 months

INTERVAL ‚3-12‘ HOUR_MINUTE
-- 3 hours 12 minutes
```

KOLIK JE HODIN?

Chcete-li odpovědět na tuto otázku, použijte:
CURRENT_TIME or CURTIME – zobrazí se aktuální čas.
CURRENT_DATE or CURDATE – chcete-li získat aktuální datum.
NOW() or CURRENT_TIMESTAMP – chcete-li získat aktuální časové razítko s oběma výše uvedenými.

VYTVÁŘENÍ HODNOT

Chcete-li vytvořit datum, čas nebo datum a čas, napište hodnotu jako řetězec a přetypujte ji na správný typ.

```
SELECT CAST(‚2021-12-31‘ AS date),
 CAST(‚15:31‘ AS time),
 CAST(‚2021-12-31 23:59:29‘ AS datetime);
```

V jednoduchých podmínkách můžete odlévání přeskočit; databáze ví, co máte na mysli.

```
SELECT airline, flight_no, departure_time
FROM airport_schedule
WHERE departure_time < ‚12:00‘;
```

VRÁCENÍ ČÁSTÍ DAT

Chcete-li vyjmout část data, použijte funkce ROK, MĚSÍC, TÝDEN, DEN, HOUR atd.

```
SELECT YEAR(CAST(‚2021-12-31‘ AS date));
-- výsledek: 2021
SELECT MONTH(CAST(‚2021-12-31‘ AS date));
-- výsledek: 12
SELECT DAY(CAST(‚2021-12-31‘ AS date));
-- výsledek: 31
```

ARITMETIKA DATUMU

Chcete-li přidat nebo odečíst interval od DATE, použijte funkci ADDDATE():

```
ADDDATE(‚2021-10-31‘, INTERVAL 2 MONTH);
-- výsledek: ‚2021-12-31‘

ADDDATE(‚2014-04-05‘, INTERVAL -3 DAY);
-- výsledek: ‚2014-04-02‘
```

Chcete-li přidat nebo odečíst interval od TIMESTAMP nebo DATETIME, použijte funkci TIMESTAMPADD():

```
TIMESTAMPADD(MONTH, 2,
 ‚2014-06-10 07:55:00‘);
-- výsledek: ‚2014-08-10 07:55:00‘

TIMESTAMPADD(MONTH, -2,
 ‚2014-06-10 07:55:00‘);
-- výsledek: ‚2014-04-10 07:55:00‘
```

Chcete-li přidat nebo odečíst ČAS od DATETIME, použijte funkci ADDTIME():

```
ADDTIME(‚2018-02-12 10:20:24‘, ‚12:43:02‘);
-- výsledek: ‚2018-02-12 23:03:26‘

ADDTIME(‚2018-02-12 10:20:24‘, ‚-12:43:02‘);
-- výsledek: ‚2018-02-11 21:37:22‘
```

Chcete-li najít rozdíl mezi dvěma daty, použijte funkci DATEDIFF():

```
DATEDIFF(‚2015-01-01‘, ‚2014-01-02‘);
-- výsledek: 364
```

Chcete-li najít rozdíl mezi dvěma časy, použijte funkci TIMEDIFF():

```
SELECT TIMEDIFF(‚09:30:00‘, ‚07:55:00‘);
-- výsledek: ‚01:35:00‘
```

Chcete-li zjistit rozdíl mezi dvěma daty a časem (v dané jednotce času), použijte funkci TIMESTAMPDIFF(). Zde je příklad s rozdílem uvedeným v týdnech:

```
SELECT TIMESTAMPDIFF(
 WEEK, ‚2018-02-26‘, ‚2018-03-21‘
); -- výsledek: 3
```