RTsoft
Softwarová řešení na míru

Informační systémy na míru
Business analytika
Mobilní aplikace
Weby a E-shopy

- Tipy k PRIMARY klíči, sekvencím a UUID
- Jak efektivně pracovat s NULL sloupci
- Jak plně využívat možností CTE nebo Window funkcí
- K čemu využít TEMP tabulky a další různé vychytávky

Restaurace U Salzmannů, salónek v 1. patře

RTsoft
Softwarová řešení na míru

RTsoft s.r.o.

- Plzeňská softwarová společnost působící na trhu již přes 20 let
- Vývoj webových a mobilních aplikací na míru
- Více než 300 dokončených projektů
- Mezi naše partnery patří MND, Knihy Dobrovský (3× eshop roku), Sazka, Sportisimo, Zásilkovna a mnoho dalších
- Více než 40 vývojářů

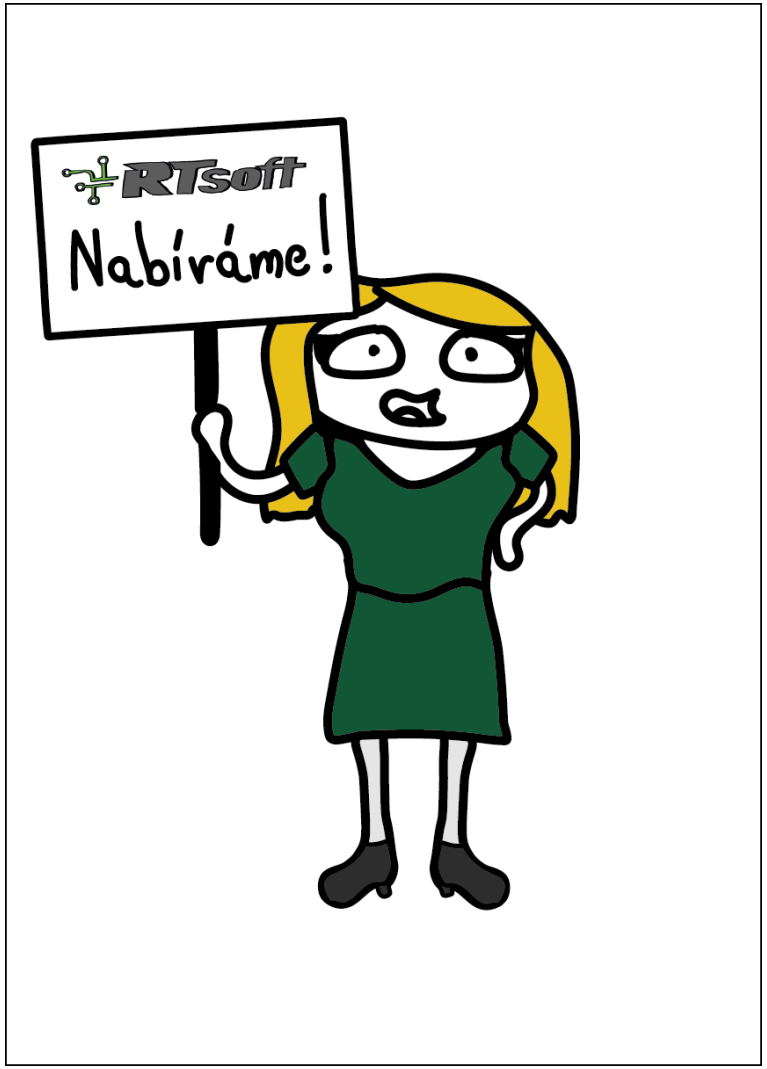
RTsoft
Softwarová řešení na míru

Informační systémy
Business
Mobilní
Weby

RTsoft
Softwarová řešení na míru

Informační systémy na míru
Business analytika
Mobilní aplikace
Weby a E-shopy

PHP Developer/ka
JS/React Developer/ka
Projektový manažer/ka



www.rtsoft.cz/kariera

brozikova@rtsoft.cz

Pavel Polívka

Software Developer

polivka@rtsoft.cz

www.rtsoft.cz

- Tipy k PRIMARY klíči, sekvencím a UUID
- Jak efektivně pracovat s NULL sloupci
- Jak plně využívat možností CTE nebo Window funkcí
- K čemu využít TEMP tabulky a další různé vychytávky

Tipy k PRIMARY klíči, sekvencím a UUID

PRIMARY klíče

- **Každá tabulka by měla mít primární klíč!**
- Je součástí všech ostatních indexů
- Volit co nejmenší datový typ
- Při skládání více sloupců použít raději unikátní index
- Cluster index – řádky jsou na disku uložené v vzestupném pořadí - **využívat sekvence!**

Sekvence – pouze MariaDB, v. 10.3+

- Alternativa k AUTO_INCREMENT – širší možnosti nastavení
- Při cachování může být rychlejší
- Lze se dotázat na poslední použitou hodnotu každé sekvence – není limitované jako LAST_INSERT_ID()
- Fyzicky jsou ukládány jako tabulky – a i se s nimi tak dá pracovat

```
CREATE [OR REPLACE] [TEMPORARY] SEQUENCE
[IF NOT EXISTS] sequence_name
[INCREMENT [BY|=] increment]
[MINVALUE [=] minvalue|NO MINVALUE|NOMINVALUE]
[MAXVALUE [=] maxvalue|NO MAXVALUE|NOMAXVALUE]
[START [WITH|=] start]
[CACHE [=] cache|NOCACHE] [CYCLE|NOCYCLE]
[table_options]
```

```
CREATE SEQUENCE `seq_product_review`
NOCACHE START 1 INCREMENT 0 MAXVALUE 10000
COMMENT 'auto_increment pro product_review';
```

```
SELECT NEXTVAL(`seq_product_review`);
SELECT LASTVAL(`seq_product_review`);
SELECT SETVAL(`seq_product_review`, 25, false);
ALTER SEQUENCE `seq_product_review` START 25
RESTART;

SELECT * FROM `seq_product_review`;
INSERT INTO `seq_product_review`(?) VALUES (?);
```

next_not_cached_value	minimum_value	maximum_value	start_value	increment	cache_size	cycle_option	cycle_count
1	1	10000	1	0	0	0	0

NEVyužití výhody clustered indexu

```
CREATE OR REPLACE TABLE `product_review`(
    `id`          INT UNSIGNED NOT NULL
        AUTO_INCREMENT PRIMARY KEY,
    `product_id` INT UNSIGNED NOT NULL,
    `review`      TEXT           NOT NULL,
    INDEX `product_id` (`product_id`)
);
```

Využití výhody clustered indexu

```
CREATE TABLE `product_review`(
    `id`          INT UNSIGNED NOT NULL
        DEFAULT NEXTVAL(`seq_product_review`),
    `product_id` INT UNSIGNED NOT NULL,
    `review`      TEXT         NOT NULL,
    PRIMARY KEY (`product_id`, `id`)
);
```

Clustered index - uložení dat na disku

Nevyužitý

id	product_id
1	1
2	2
3	1
4	1
5	3
6	1

Využitý

product_id	id
1	1
1	3
1	4
1	6
2	2
3	5

Aktualizace počtu zobrazení

```
CREATE TABLE `banner_views`(
    `banner_id` INT UNSIGNED NOT NULL PRIMARY
KEY,
    `views`     INT UNSIGNED NOT NULL
);
INSERT INTO `banner_views` (`banner_id`,
`views`) VALUES (?, 1)
ON DUPLICATE KEY UPDATE `views` = `views` + 1;
```

- Při vyšší frekvenci volání může nastat čekání na zámek

Aktualizace počtu zobrazení – rozptyl 10

```
CREATE TABLE `banner_views`(
    `banner_id` INT UNSIGNED      NOT NULL,
    `spread`    SMALLINT UNSIGNED NOT NULL,
    `views`     INT UNSIGNED      NOT NULL,
    PRIMARY KEY (`banner_id`, `spread`)
);
INSERT INTO `banner_views` (`banner_id`,
`spread`, `views`)
VALUES (?, FLOOR(RAND() * 10), 1)
ON DUPLICATE KEY UPDATE `views` = `views` + 1;
```

UUID - Universally Unique Identifier

- 128 bitový identifikátor (UUID v.1)
- Složen z: time low/mid/high, random, MAC
- Příklad: 550e8400-e29b-41d4-a716-446655440000
- CHAR(36), tzn. 108/144 B (utf8/utf8mb4)
- Ukládat jako BINARY(16) – zejména u PRIMARY KEY
- PK - využívat variantu 6 s prohozenou time low a high

UUID – PRIMARY - výhody a nevýhody

- Náhrada za AUTO_INCREMENT - rychlejší
- Lze jej generovat i z kódu aplikace

- Vyšší nároky na ukládání (oproti INT/BIGINT) – indexy
- Nelze paralelně volat hromadný INSERT – GAP LOCK
- Nelze využívat v Statement-based replikaci

UUID – BINARY, varianta 6

- Pouze v MySQL, v 8.0+ přidán parametr „swap_flag“:
- `UUID_TO_BIN(UUID(), 1)`
- `BIN_TO_UUID(\`binary_uuid_column\`, 1)`

- V PHP knihovna https://github.com/ramsey/uuid:

```
$uuidBin = hex2bin(Uuid::uuid6()->getHex()->toString());
```

UUID – BINARY varianta 6 v MariaDB

```
CREATE FUNCTION `uuid_to_bin_shuffled`
    (`uuid` CHAR(36)) RETURNS BINARY(16)
    COMMENT 'UUID v6 BINARY' DETERMINISTIC
    RETURN
        UNHEX(CONCAT(
                SUBSTR(`uuid`, 15, 4),
                SUBSTR(`uuid`, 10, 4),
                SUBSTR(`uuid`, 1, 8),
                SUBSTR(`uuid`, 20, 4),
                SUBSTR(`uuid`, 25)));
```

UUID – převod z BINARY v MariaDB

```
CREATE FUNCTION `uuid_from_bin`
(`uuid_in_bin` BINARY(16)) RETURNS CHAR(36)
    COMMENT 'UUID from BINARY' DETERMINISTIC
    RETURN
        LCASE(CONCAT_WS('-',
            HEX(SUBSTR(`uuid_in_bin`, 1, 4)),
            HEX(SUBSTR(`uuid_in_bin`, 5, 2)),
            HEX(SUBSTR(`uuid_in_bin`, 7, 2)),
            HEX(SUBSTR(`uuid_in_bin`, 9, 2)),
            HEX(SUBSTR(`uuid_in_bin`, 11))));
```

Jak efektivně pracovat s NULL sloupci

Definice sloupců s NULL

- Ke sloupci je přidán příznak s informací, jestli je NULL

Column	Column_is_null
10	☐ IS NULL
1	☐ IS NULL
	☑ IS NULL

Jak pracovat s NULLable sloupci

- Využívat konstrukce „column IS [NOT] NULL“
- Využívat NULL-safe equal operátor **<=>**
- Využívat funkce:
  - IFNULL(column, value_to_return_when_is_null)
  - COALESCE(column, column1, …) – vrátí první not-NULL hodnotu

Jak se chovají NULL hodnoty

- Aritmetické operace s NULL vždy vrací NULL
- ORDER BY - NULL řádky na prvním místě (při ASC)
- Agregační funkce NULL hodnoty ignorují
- Funkce pracují s NULL hodnotami různým způsobem
- Používat funkce určené pro práci s NULL

Řazení sloupců s NULL hodnotami

- ORDER BY - NULL řádky na prvním místě (při ASC)

```
SELECT * FROM `address`
ORDER BY `last_name` IS NULL, `last_name`;

SELECT * FROM `address`
ORDER BY `last_name` DESC;
```

Příklady

```
SET @a = 5, @b = 20, @c = NULL;
SELECT CONCAT(@a, '_', @b); -- 5_20
SELECT CONCAT(@a, '_', @b, '_', @c); -- NULL
SELECT @a > @c; -- NULL
SELECT @a * @b * @c; -- NULL
SELECT @c = @c; -- NULL
SELECT @c <=> @c; -- 1
SELECT !(@c <=> @c); -- 0
```

Příklady

```
SET @a = 5, @b = 20, @c = NULL;
SELECT LEAST(@a, @b, @c); -- NULL
SELECT GREATEST(@a, @b, @c); -- NULL
SELECT GREATEST(
    COALESCE(@a, @b, @c), -- 5
    COALESCE(@b, @a, @c), -- 20
    COALESCE(@c, @a, @b) -- 5
); -- 20;
```

Na co nezapomenout

- Nepoužívat funkce (např. IFNULL) ve WHERE nebo v ORDER BY – nepoužije se případný index
- Vždy zkontrolovat, jestli je daný sloupec NULLable a z dokumentaci ověřit chování použité funkce
- Nepoužívat zbytečné podmínky na NOT NULL:

~~`number` IS NOT NULL AND~~ `number` > 10

Jak plně využívat možností CTE nebo Window funkcí

CTE – Common Table Expression

- MySQL 8+ a MariaDB 10.2.1+
- Alternativa k subquery
- Obdoba ke CREATE TEMPORARY TABLE
- 2 typy: standardní a rekurzivní
- CTE lze využít v dotazu vícekrát, a to i v jiné CTE
- Větší přehlednost SQL dotazu

```
WITH [RECURSIVE] cte_name [(col_name [, col_name] ...)] AS (subquery)
[, cte_name [(col_name [, col_name] ...)] AS (subquery)] ...
```

- Profilové volby pro rekurzivní CTE:
  - cte_max_recursion_depth (1000)
  - max_execution_time (0)
- MariaDB: max_recursive_iterations (v10.6+: 1000)

CTE – číselná řada pro naplnění tabulky

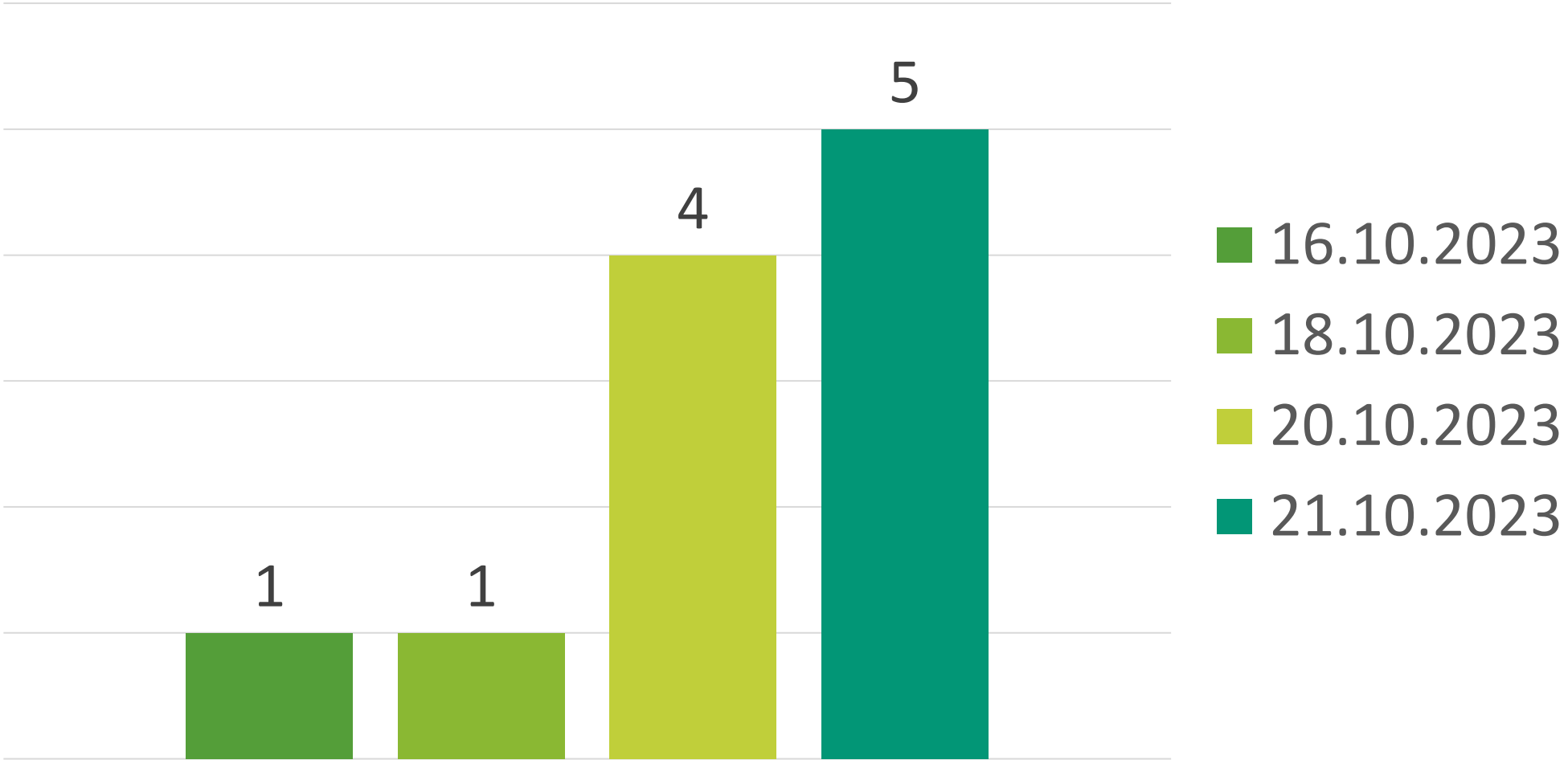
```
CREATE TABLE `my_odd_table`(
    `id` INT UNSIGNED NOT NULL PRIMARY KEY);

INSERT INTO `my_odd_table` (`id`)
WITH RECURSIVE `seq_odd_100` AS (
    SELECT 1 AS `id`
    UNION ALL
    SELECT `id` + 2 FROM `seq_odd_100`
WHERE `id` < 99)
SELECT `id` FROM `seq_odd_100`;
```

CTE – zjištění chybějících záznamů

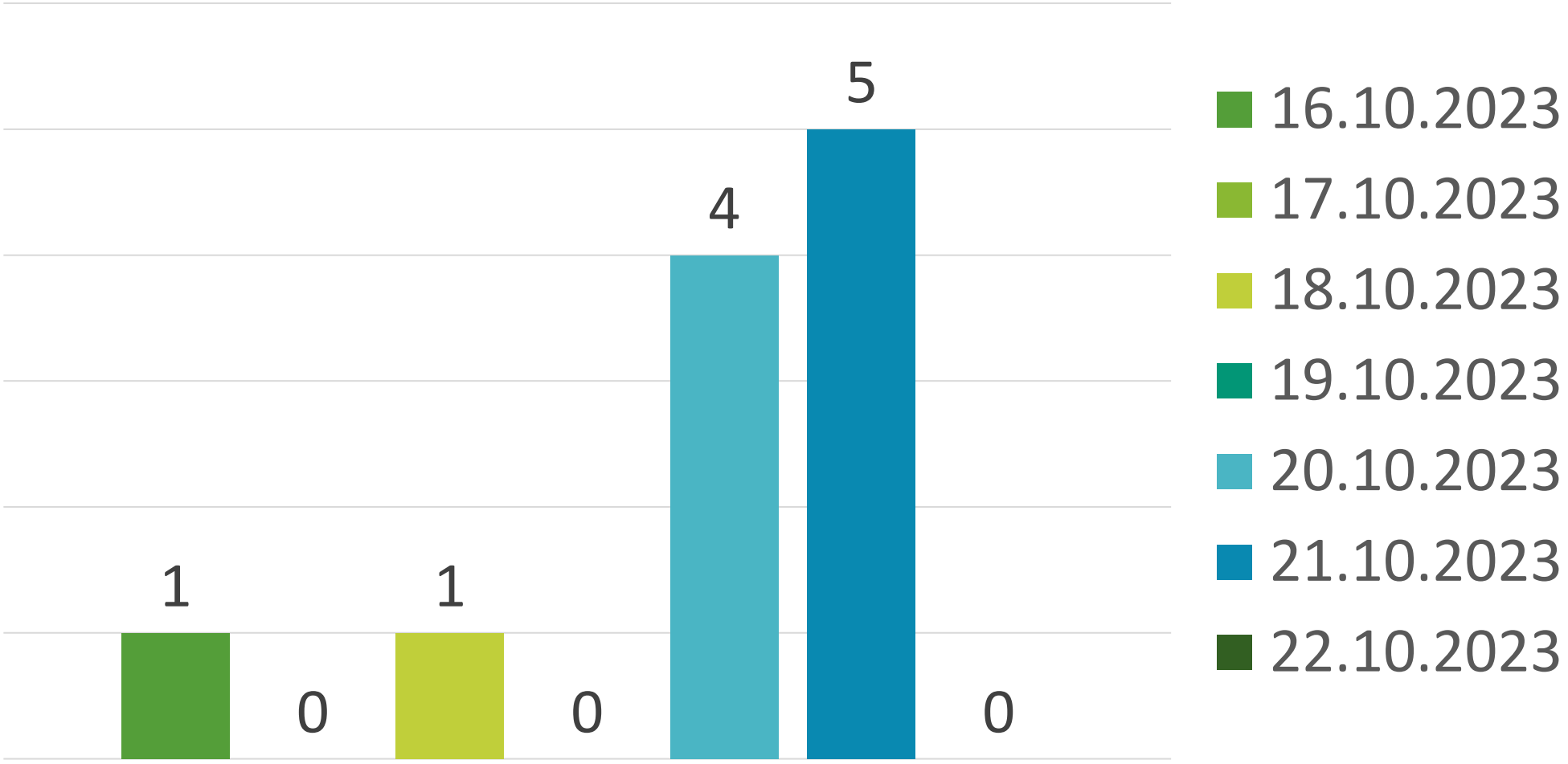
```
WITH RECURSIVE `seq_100` AS (
    SELECT 1 AS `id`
    UNION ALL
    SELECT `id` + 1 FROM `seq_100`
WHERE `id` < 100)
SELECT `seq_100`.`id` FROM `seq_100`
    LEFT JOIN `my_odd_table` USING (`id`)
WHERE `my_odd_table`.`id` IS NULL
```

Vypitá piva ve 42. týdnu 2023



```
WITH RECURSIVE beer_stats_week_42 AS (
  SELECT `day`, COUNT(beer) as beer_count
  FROM beer_stats WHERE
  `day` BETWEEN '2023-10-16' AND '2023-10-22'
  GROUP BY `day`),
`week_42` AS (
  SELECT '2023-10-16' AS `day`
  UNION ALL
  SELECT DATE_ADD(`day`, INTERVAL 1 DAY) as `day`
  FROM `week_42` WHERE `day` < '2023-10-22')
SELECT day`, IFNULL(`beer_count`, 0) FROM `week_42`
  LEFT JOIN `beer_stats_week_42` USING (`day`);
```

Vypitá piva ve 42. týdnu 2023



```
SELECT `su`.`id`,
       `s1`.`id` AS `s1_id`,
       `s2`.`id` AS `s2_id`,
       `s3`.`id` AS `s3_id`,
       `s4`.`id` AS `s4_id`
FROM `struct_usage` `su`
/* Čtyři úrovně budou určitě stačit… */
INNER JOIN `struct` `s1`
    ON `su`.`struct_id` = `s1`.`id`
LEFT JOIN `struct` `s2` ON `s1`.`p_id` = `s2`.`id`
LEFT JOIN `struct` `s3` ON `s2`.`p_id` = `s3`.`id`
LEFT JOIN `struct` `s4` ON `s3`.`p_id` = `s4`.`id`;
```

```
WITH RECURSIVE `struct_cte` AS (
  SELECT `id`, `p_id`, 1 AS `level`,
         CAST(`id` AS CHAR) AS `path`
  FROM `struct`
  UNION ALL
  SELECT `scte`.`id`, `s`.`p_id`, `level` + 1,
         CONCAT(`scte`.`path`, '.', `s`.`id`)
  FROM `struct_cte` AS `scte`
    JOIN `struct` AS `s` ON `scte`.`p_id` = `s`.`id`
)
SELECT * FROM `struct_cte`;
```

id	s1_id	s2_id	s3_id	s4_id
2	10	6	3	1

id	p_id	level	path
10	6	1	10
10	3	2	10.6
10	1	3	10.6.3
10		4	10.6.3.1

ChatGPT Prompt – tabulka sctruct

- Vytvoření a naplnění tabulky struct:
  - Generate SQL scripts to create a table named 'struct' with 'id' and 'p_id' columns to store a hierarchical structure, and then insert dummy data into it for three levels.
- Můžete zkusit i prompt na různé rekurzivní CTE, ale bohužel tam mohou být drobné chybky

Window funkce

```
Fn() OVER ([window_name] [partition_clause]
[order_clause] [frame_clause])
```

- Obdoba k GROUP BY, ale v řádkovém výstupu
- Celá řada funkcí různého typu:
  - Agregační – MIN(), AVG(), COUNT(), …
  - Řádkové – ROW_NUMBER(), LEAD(), LAG() , …
  - Analytické – RANK(), NTILE() , …

Příklad Window agregační funkce

```
SELECT `day`,
       `beer`,
       COUNT(`beer`) OVER ()
            AS `beer_cnt`,
       COUNT(`beer`) OVER (PARTITION BY `day`)
            AS `beer_day_cnt`
FROM `beer_stats`;
```

day	beer	beer_cnt	beer_day_cnt
16.10.2023	1	11	1
18.10.2023	1	11	1
20.10.2023	1	11	4
20.10.2023	1	11	4
20.10.2023	1	11	4
20.10.2023	1	11	4
21.10.2023	1	11	5
21.10.2023	1	11	5
21.10.2023	1	11	5
21.10.2023	1	11	5
21.10.2023	1	11	5

Příklad Window řádkových funkcí

```
SELECT
  `user_id`,
  `address_id`,
  ROW_NUMBER() OVER (PARTITION BY `user_id`
  ORDER BY `type` DESC, `id` DESC) AS `row_number`,
  FIRST_VALUE(`id`) OVER (PARTITION BY `user_id`
  ORDER BY `type` DESC, `id` DESC) AS `first_id`
FROM `user_to_address`
```

Příklad Window LEAD funkce

```
SELECT
 `zone_id`,`weight_from`,`price`,
 LEAD(`weight_from`) OVER (PARTITION BY `zone_id`
 ORDER BY `weight_from`) AS `weight_to`
FROM
    `zones_weight_prices`
ORDER BY
    `zone_id`, `weight_from`;
```

zone_id	weight_from	price	weight_to
1	0.00	5.00	**5.01**
1	**5.01**	10.00	**10.01**
1	**10.01**	15.00	
2	0.00	6.00	**4.01**
2	**4.01**	11.00	**12.01**
2	**12.01**	16.00	
3	0.00	7.00	**8.01**
3	**5.01**	12.00	**18.01**
3	**10.01**	17.00	

K čemu využít TEMP tabulky a další různé vychytávky

TEMP tabulky

- Lze definovat úložiště – disk/paměť
- Mohou obsahovat indexy
- Jsou dostupné pouze pro danou db session
- Po ukončení db session se automaticky smažou
- Mohou se jmenovat stejně jako již existující tabulky

TEMP tabulky – možnosti využití

- Zrychlení pomalých hromadných DML dotazů
- Zjednodušení složitějších SQL dotazů
- Forma cache (zejména ve variantě uložené v paměti)

```
CREATE TEMPORARY TABLE name ENGINE=MEMORY
    AS SELECT ...
```

MariaDB – Sequence Storage Engine

- Tabulky **seq**_FROM_**to**_TO nebo **seq**_FROM_**to**_TO_**step**_STEP

```
SHOW ENGINES;

SELECT * FROM seq_1_to_5;
SELECT * FROM seq_1_to_15_step_3;
SELECT '2023-10-16' + INTERVAL (seq) DAY
    FROM seq_0_to_6;
```

Jmenné zámky

- Uživatelský zámek, plošný přes všechny db sessions

```
SELECT GET_LOCK('lock1', 10);
SELECT RELEASE_LOCK('lock1');
SELECT IS_FREE_LOCK('lock1`);
SELECT IS_USED_LOCK('lock1`);
SELECT RELEASE_ALL_LOCKS();
```

RTsoft
Softwarová řešení na míru

Děkuji za pozornost

PeoplePath
IT workshop

Rozkvět PHP

Lokální vývoj s použitím Dockeru

6. 12. 2023 v 18:00

Restaurace U Salzmannů

Salonek v 1. patře

Vstup a občerstvení zdarma