
ORACLE®

MySQL 8.0 RC
新武器前瞻

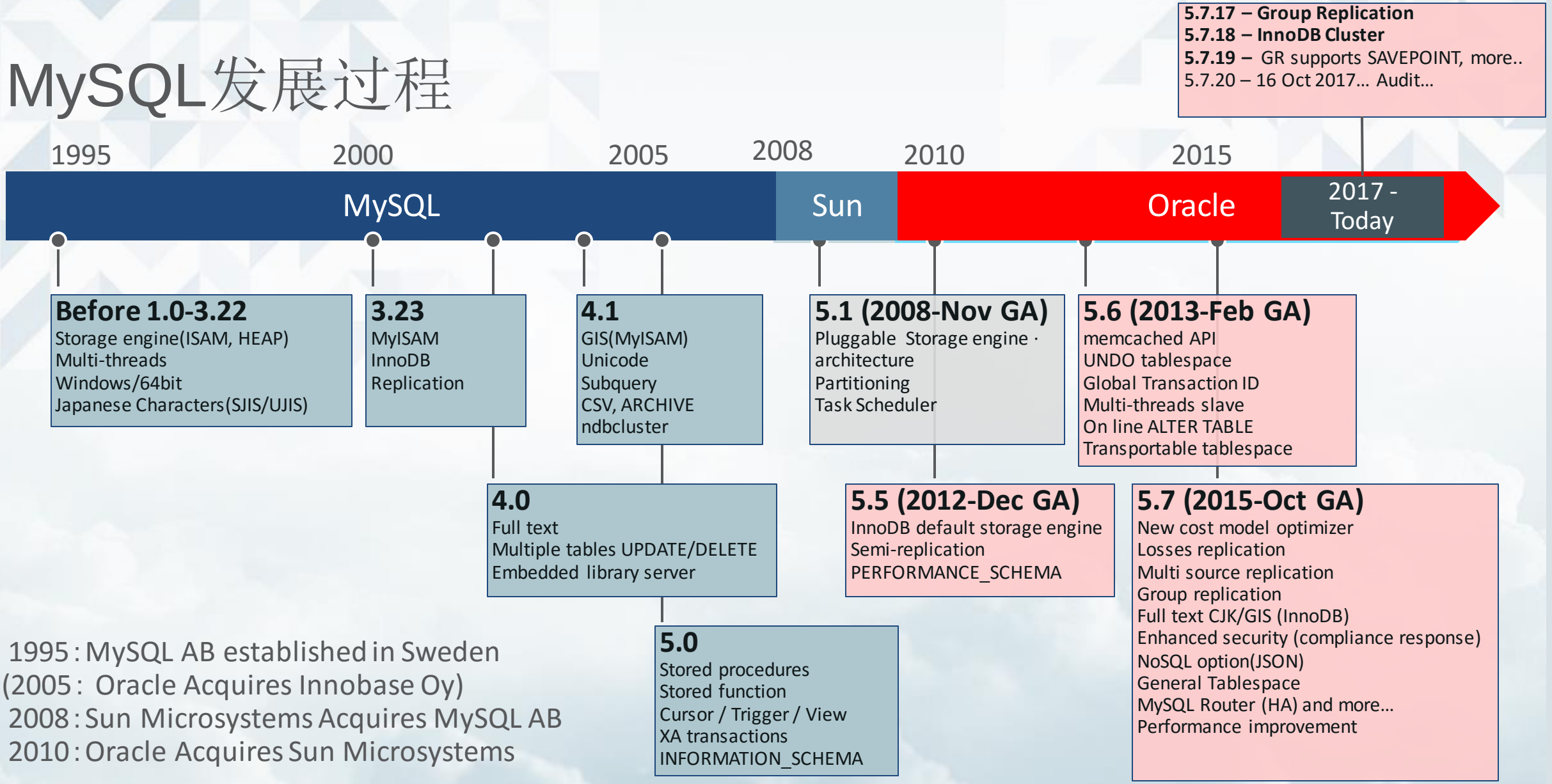
马楚成
ivan-cs.ma@oracle.com
2017-12-09

2017年
MYSQL技
术嘉年华

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

MySQL发展过程



- 1995 : MySQL AB established in Sweden
- (2005 : Oracle Acquires Innobase Oy)
- 2008 : Sun Microsystems Acquires MySQL AB
- 2010 : Oracle Acquires Sun Microsystems

MySQL 发展过程

MySQL 5.7

- 3x Better Performance
- Replication Enhancements
- JSON Support
- Improved Security

MySQL InnoDB Cluster

- MySQL Group Replication
- MySQL Router
- MySQL Shell

MySQL 8.0 (RC)

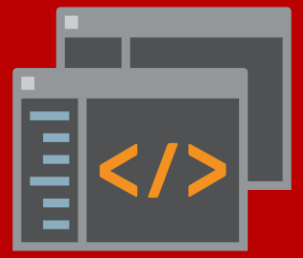
- Data Dictionary
- Roles
- Unicode
- CTEs
- Window Functions
- Security
- Replication

MySQL 8.0：新武器前瞻 功能更强的数据库

移动应用

地理空间（GPS/GIS）应用

Emoji / Unicode字符 😃



开发能力

混合数据模型，

提供更多和方便的API来处理数据

数据分析

新加的 CTE/WINDOW 数据SQL分析功能 -提供实时数据分析

24x7 at Scale

扩展和稳定

提高安全性并减少停机时间，群组复制的大方向。

ORACLE

MySQL 8.0 :移动应用

增强的地理空间-GIS支持

- 更方便/容易存储和处理GIS数据 （移动GPS数据）
- MySQL 5.7已经以Boost.Geometry为基础
- MySQL 8.0中的地理空间SRS ID的支持

Unicode 默认配置

- **`utf8mb4`** 对 Emoji 😃 的支持。MySQL8.0的默认字符
- 性能提升达到16x
- 基于Unicode 9.0
- 新的排序规则 -基于 UCA， 可对口音，大小写 有区分

MySQL 8.0：开发能力

数据类型

JSON Datatype

以高效的更新性能无缝地管理RDBMS表中的“非结构化”数据

SQL 函数

JSON Functions

提供SQL函数来搜索和修改JSON。 可对JSON用JSON_TABLE（）转换成表格式

混合API

MySQL X DevAPI

提供更灵活的SQL和NoSQL混合CRUD API开发

MySQL 8.0 :数据分析功能

Common Table Expressions (CTEs)

- 以“WITH”字句，来表达子查询
- 提高可读性和性能
- 递归CTE的支持

```
WITH tickets_filtered AS (
 SELECT tickets.*, seats.doc
 FROM tickets
 INNER JOIN seats ON
  tickets.seat_id = seats.id
 WHERE tickets.event_id = 3
)
SELECT * FROM tickets_filtered
 WHERE doc->"$.section" = 201\G
```

Window 函数

- 常用的数据分析功能，如数据排名
- 通过滚动聚合进行计算

```
SELECT name, dept_id, salary,
 RANK() OVER w AS `rank`
FROM employee
 WINDOW w AS
  (PARTITION BY dept_id
   ORDER BY salary DESC);
```

ORACLE

MySQL 8.0：性能提升

避免卡住

SELECT FOR UPDATE的**NOWAIT**和**SKIP LOCKED**选项可更好地处理热点争用

Invisible Indexes

优化器对隐藏的索引，可作“软删除”和“分阶段发布”

Performance Schema

默认情况下允许启用更多的工具，并且添加索引时更好地响应视图

会话 SQL 提示

使用新的提示选项SET_VAR在单个语句的持续时间内设置会话变量

降序索引 DESC

按降序排列的组合索引中的排序顺序提供更快的性能

加强的提示

提示控制连接表索引和索引合并而不需要重新构造查询

MySQL 8.0 :扩展性更高

InnoDB Dedicated Server

自动调整InnoDB配置，非常适合虚拟机和云部署

配置参数变量更云化

使用SET PERSIST保留服务器变量，并方便从information schema查看更改源头

资源组 ResourceGroup

通过线程和CPU之间的资源组获得更好的效率和/或性能

列的直方图

为优化器提供有关列值分布的信息，建构更好的查询计划

更好的成本估算

成本模型针对新存储技术和优化器进行了优化，并提供了对存储器缓冲区的成本估算

CATS调度算法

“Contention-Aware Transaction Scheduling”是InnoDB中的默认调度算法，用于提升性能

MySQL 8.0：更稳定 + 更安全

MySQL InnoDB Cluster

一体高可用方案，群组复制提供分布式回复，冲突检查和群组服务器配置

数据字典

提高事务性元数据存储库的可靠性和一致性

安全功能

新加功能 - SQL Role， 动态权限，TDE 加密的优化

MySQL 8.0：安全功能的优化

SQL Role

易于管理用户和应用程序权限
以及符合SQL标准

控制列表ACL 的原子性

基于InnoDB的新数据字典
可以使ACL原子化

动态权限

提供更细化的访问控制管理级别，
以减少以root用户的使用

TDE – 对日志加密

除了表空间文件之外，对REDO，UNDO和二进制日志 以AES 256加密

优化Password

用密码历史建立密码重用策略，使用缓存更快

MySQL 8.0 两大分析武器前瞻

Window 函数

Common Table Expressions (CTEs)

Window 分析功能介绍

- 自2003年，SQL标准的一部分
- 收到的数据分析功能要求-很多很多
- 提高可读性，查询性能/速度更快

什么是 WINDOW 分析

	Emp_no	Salary	from_date	to_date
WINDOW	11	1000	2000-1-1	2002-10-1
	11	2000	2002-10-1	2003-5-1
	11	3000	2003-5-1	2005-5-30
	11	4000	2015-1-1	NULL
WINDOW	20	1100	2000-1-1	2002-10-1
	20	2300	2002-10-1	2003-5-1
	20	3400	2003-5-1	2005-5-30
	20	4500	2015-1-1	NULL
	31	2000	2008-1-1	2010-2-1
	31	3000	2015-3-1	NULL
	…	…	…	…

Emp_no	Salary	Max(from_date)
11	4000	2015-1-1
20	4500	2015-1-1
31	. . .	. . .

Table : salaries (emp_no, salary, from_date, to_date) 2,844K rows in salaries – output 300K rows

- select s1.emp_no, s1.from_date, s1.salary
  from salaries s1,
  (select s2.emp_no, max(s2.from_date) as fdate from salaries s2 group by s2.emp_no) as s3
  where s1.emp_no = s3.emp_no
  and s1.from_date = s3.fdate;

- Duration 00:00:47.388499

- select distinct emp_no,
  FIRST_VALUE(salary) **OVER** w as latest_salary
  from salaries
  WINDOW w as
  (partition by emp_no
  order by from_date desc) ;

- Duration : 00:00:23.035922

更快，更好看的SQL

```
SELECT name o_name, department_id,
       salary AS o_salary,
       (SELECT SUM(salary) AS sum
        FROM employee
        WHERE salary <= o_salary AND NOT
             (salary = o_salary AND
              o_name > name)) AS sum
FROM employee
ORDER BY sum, name;
```

name	department_id	salary	sum
Dag	10	NULL	NULL
Frederik	10	60000	60000
Jon	10	60000	120000
Lena	20	65000	185000
Paula	20	65000	250000
Michael	10	70000	320000
William	30	70000	390000
Nils	NULL	75000	465000
Nils	20	80000	545000
Erik	10	100000	645000
Rose	30	300000	945000

```
SELECT name, department_id, salary,
       SUM(salary) OVER w AS sum
FROM employee
WINDOW w AS (ORDER BY salary, name
             ROWS UNBOUNDED PRECEDING)
ORDER BY sum, name;
```

可读性更清楚

WINDOW 数据分析

- 从一组数据中,
  - 由分区和框架定义的窗口
  - 得到汇总数据

- 什么是分区 (Partition)
- 什么事框架 (Frame)
- 什么是窗口函数 (Window Function)
  - 提示 – 以 OVER 关键字

WINDOW 函数– 语法

- SELECT …,
- <WINDOW Function (e.g. sum(salary)) OVER <window_sec> | <window_name>
- FROM ….
- WINDOW <window_name> AS (

  [partition_clause] [order_clause] [frame_clause]

)

- <window function>可以是聚合函数，包括SUM（），AVG（），MIN（），MAX（）和COUNT（）。 。如果没有PARTITION BY子句，查询结果集将为分区。

```
SELECT name, department_id, salary
FROM employee
ORDER BY department_id, name;
```

```
+----------+---------------+--------+
| name     | department_id | salary |
+----------+---------------+--------+
| Nils     |          NULL |  75000 |
| Dag      |            10 |   NULL |
| Erik     |            10 | 100000 |
| Frederik |            10 |  60000 |
| Jon      |            10 |  60000 |
| Michael  |            10 |  70000 |
| Lena     |            20 |  65000 |
| Nils     |            20 |  80000 |
| Paula    |            20 |  65000 |
| Rose     |            30 | 300000 |
| William  |            30 |  70000 |
+----------+---------------+--------+
```

```
SELECT department_id, SUM(salary)
   FROM employee GROUP BY department_id
   ORDER BY department_id;
```

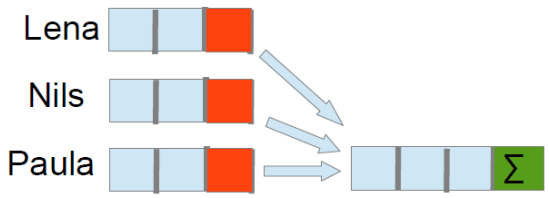
```
+---------------+-------------+
| department_id | SUM(salary) |
+---------------+-------------+
|          NULL |       75000 |
|            10 |      290000 |
|            20 |      210000 |
|            30 |      370000 |
+---------------+-------------+
```

```
SELECT name, department_id, salary
FROM employee
ORDER BY department_id, name;
```

```
+----------+---------------+--------+
| name     | department_id | salary |
+----------+---------------+--------+
| Nils     |          NULL |  75000 |
| Dag      |            10 |   NULL |
| Erik     |            10 | 100000 |
| Frederik |            10 |  60000 |
| Jon      |            10 |  60000 |
| Michael  |            10 |  70000 |
| Lena     |            20 |  65000 |
| Nils     |            20 |  80000 |
| Paula    |            20 |  65000 |
| Rose     |            30 | 300000 |
| William  |            30 |  70000 |
+----------+---------------+--------+
```

```
SELECT department_id, SUM(salary)
   FROM employee GROUP BY department_id
   ORDER BY department_id;
```

```
+---------------+-------------+
| department_id | SUM(salary) |
+---------------+-------------+
|          NULL |       75000 |
|            10 |      290000 |
|            20 |      210000 |
|            30 |      370000 |
+---------------+-------------+
```



资料：Name, Salary
结果没能保留数据

WINDOW – 分析结果也保留源数据

```
SELECT name, department_id, salary,
       SUM(salary) OVER () sum
FROM employee
ORDER BY department_id, name;
```

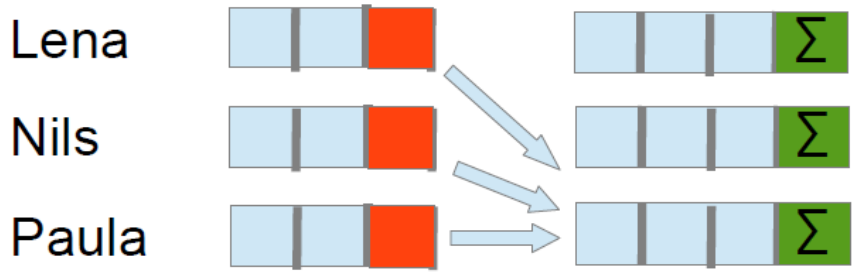
name	department_id	salary	sum
Nils	NULL	75000	945000
Dag	10	NULL	945000
Erik	10	100000	945000
Frederik	10	60000	945000
Jon	10	60000	945000
Michael	10	70000	945000
Lena	20	65000	945000
Nils	20	80000	945000
Paula	20	65000	945000
Rose	30	300000	945000
William	30	70000	945000

```
SELECT name, salary, department_id,
       SUM(salary) OVER (PARTITION BY department_id) sum
FROM employee
ORDER BY department_id;
```

WINDOW 函数 -增加的列
它的值=从窗口框(frame)中读取和汇总数据

分区数据➔partition by department_id
Frame ➔ EMPTY

汇总数据 ➔ sum(salary)



默认WINDOW分区

```
SELECT name, department_id, salary,
       SUM(salary) OVER () sum
FROM employee
ORDER BY department_id, name;
```

name	department_id	salary	sum
Nils	NULL	75000	945000
Dag	10	NULL	945000
Erik	10	100000	945000
Frederik	10	60000	945000
Jon	10	60000	945000
Michael	10	70000	945000
Lena	20	65000	945000
Nils	20	80000	945000
Paula	20	65000	945000
Rose	30	300000	945000
William	30	70000	945000

没有指定分区 - 意味着所有的数据

累计总和

```
SELECT name, department_id, salary,
       SUM(salary) OVER (ORDER BY department_id, name
                         ROWS BETWEEN UNBOUNDED PRECEDING
                         AND CURRENT ROW) sum
FROM employee
ORDER BY department_id, name;
```

name	department_id	salary	sum
Nils	NULL	75000	75000
Dag	10	NULL	75000
Erik	10	100000	175000
Frederik	10	60000	235000
Jon	10	60000	295000
Michael	10	70000	365000
Lena	20	65000	430000
Nils	20	80000	510000
Paula	20	65000	575000
Rose	30	300000	875000
William	30	70000	945000

分区 – 默认整个结果集

FRAME – ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

累计总和

```
SELECT name, department_id, salary,
       SUM(salary) OVER (ORDER BY department_id, name
                         ROWS BETWEEN UNBOUNDED PRECEDING
                         AND CURRENT ROW) sum
FROM employee
ORDER BY department_id, name;
```

name	department_id	salary	sum
Nils	NULL	75000	75000
Dag	10	NULL	75000
Erik	10	100000	175000
Frederik	10	60000	235000
Jon	10	60000	295000
Michael	10	70000	365000
Lena	20	65000	430000
Nils	20	80000	510000
Paula	20	65000	575000
Rose	30	300000	875000
William	30	70000	945000

分区 – 默认整个结果集

FRAME – ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

累计总和 – 分区

```
SELECT name, department_id, salary,
       SUM(salary) OVER (PARTITION BY department_id
                         ORDER BY name
                         ROWS BETWEEN UNBOUNDED PRECEDING
                         AND CURRENT ROW) sum
FROM employee
ORDER BY department_id, name;
```

name	department_id	salary	sum
Nils	NULL	75000	75000
Dag	10	NULL	NULL
Erik	10	100000	100000
Frederik	10	60000	160000
Jon	10	60000	220000
Michael	10	70000	290000
Lena	20	65000	65000
Nils	20	80000	145000
Paula	20	65000	210000
Rose	30	300000	300000
William	30	70000	370000

分区 – PARTITION By department_id

FRAME – ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

新分区 – 重设

累计总和 – 分区

```
SELECT name, department_id, salary,
       SUM(salary) OVER (PARTITION BY department_id
                         ORDER BY name
                         ROWS BETWEEN UNBOUNDED PRECEDING
                         AND CURRENT ROW) sum
FROM employee
ORDER BY department_id, name;
```

name	department_id	salary	sum
Nils	NULL	75000	75000
Dag	10	NULL	NULL
Erik	10	100000	100000
Frederik	10	60000	160000
Jon	10	60000	220000
Michael	10	70000	290000
Lena	20	65000	65000
Nils	20	80000	145000
Paula	20	65000	210000
Rose	30	300000	300000
William	30	70000	370000

分区 – PARTITION By department_id

FRAME – ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

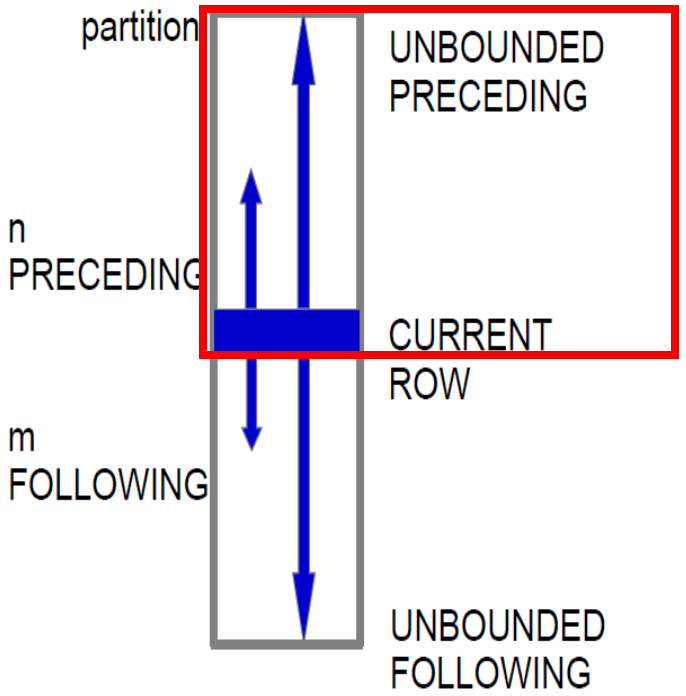
分区 （department_id=10)

分区 （department_id=20)

FRAME – 定义框架

- 分区 – Partition
  - Frame →子集

```
SELECT name, department_id, salary,
            SUM(salary) OVER  ( PARTITION BY department_id
                                ORDER BY name
                                ROWS BETWEEN UNBOUNDED PRECEDING
                                AND CURRENT ROW) sum
FROM employee
ORDER BY department_id, name;
```



FRAME – 定义框架

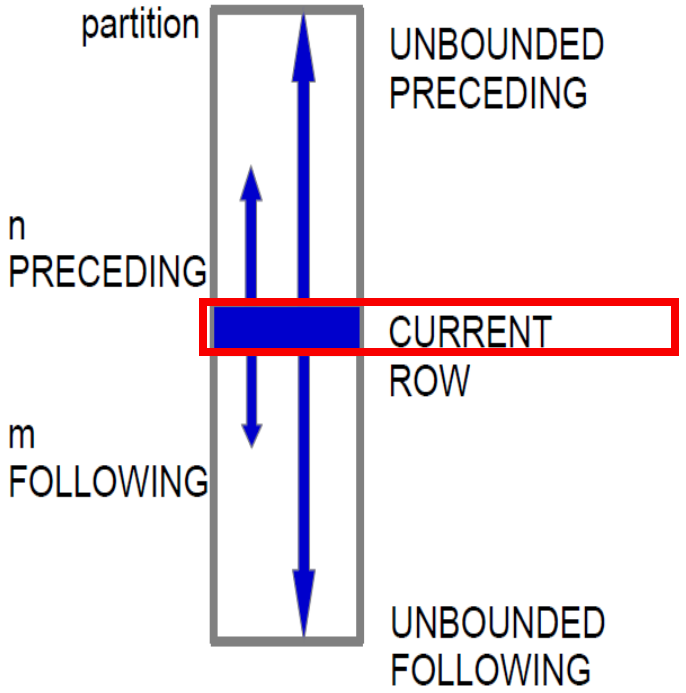
- ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) sum

- **RANGE CURRENT ROW**

- ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING

- ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING

  RANGE INTERVAL 6 DAY PRECEDING AND CURRENT ROW



FRAME – 定义框架

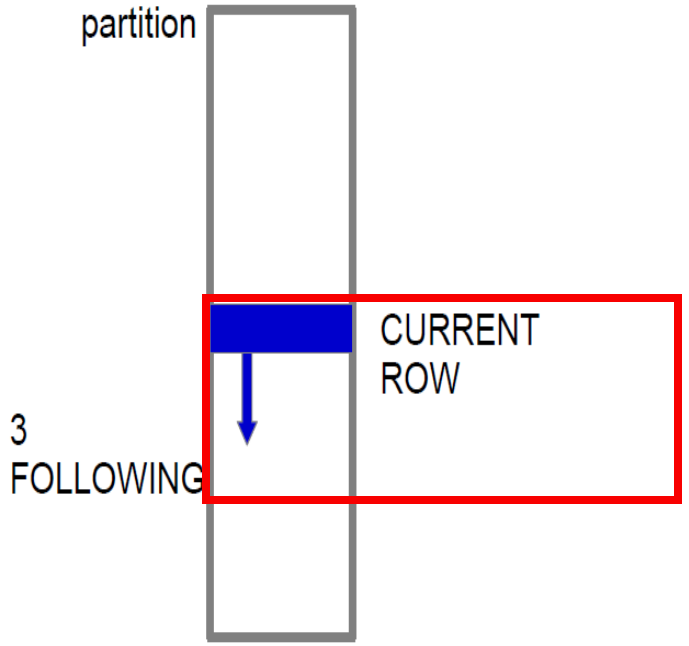
- ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) sum

- RANGE CURRENT ROW

- **ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING**

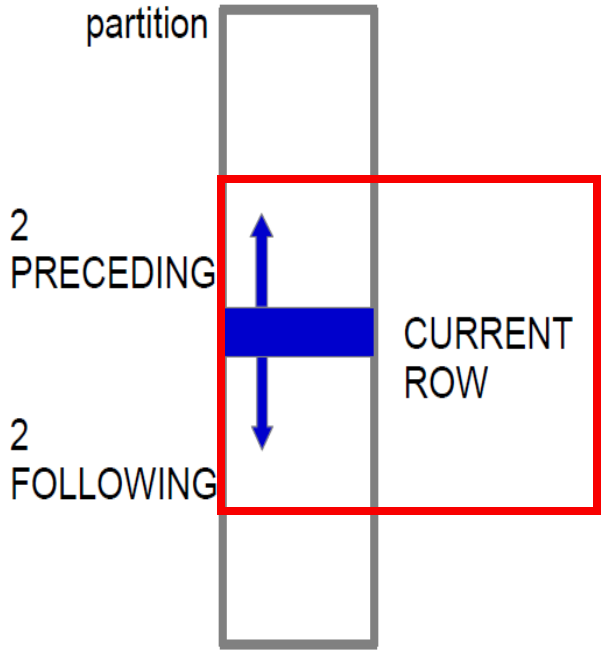
- ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING

  RANGE INTERVAL 6 DAY PRECEDING AND CURRENT ROW



FRAME – 定义框架

- ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) sum
- RANGE CURRENT ROW
- ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING
- **ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING**
- RANGE INTERVAL 6 DAY PRECEDING AND CURRENT ROW



FRAME – 定义框架

- ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) sum
- RANGE CURRENT ROW
- ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING
- ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING

ROWS vs RANGE

定义边界 – PHYSICAL (ROWS) / LOGICAL (RANGE)

RANGE – ORDER BY 是必要

例子：过去的一周

```
ORDER BY date
RANGE BETWEEN INTERVAL 6 DAY PRECEDING
AND CURRENT ROW
```

默认设定：

```
OVER (ORDER BY n) == OVER ( ORDER BY n
                    RANGE BETWEEN UNBOUNDED PRECEDING
                    AND CURRENT ROW)
```

RANGE INTERVAL 6 DAY PRECEDING AND CURRENT ROW

SQL 分析 -移动平均图表

```
CREATE TABLE sales (
    id INT AUTO_INCREMENT PRIMARY KEY,
    date DATE,
    sale INT); …;
SELECT * FROM sales;
```

id	date	sale
1	2017-03-01	200
2	2017-04-01	300
3	2017-05-01	400
4	2017-06-01	200
5	2017-07-01	600
6	2017-08-01	100
7	2017-03-01	400
8	2017-04-01	300
9	2017-05-01	500
10	2017-06-01	400
11	2017-07-01	600
12	2017-08-01	150

SQL 分析 -移动平均图表

```
SELECT MONTH(date), SUM(sale)
FROM sales
GROUP BY MONTH(date);
```

MONTH(date)	SUM(sale)
3	600
4	600
5	900
6	600
7	1200
8	250

id	date	sale
1	2017-03-01	200
2	2017-04-01	300
3	2017-05-01	400
4	2017-06-01	200
5	2017-07-01	600
6	2017-08-01	100
7	2017-03-01	400
8	2017-04-01	300
9	2017-05-01	500
10	2017-06-01	400
11	2017-07-01	600
12	2017-08-01	150

SQL 分析 -移动平均图表

```
SELECT MONTH(date), SUM(sale),
        AVG(SUM(sale)) OVER w AS sliding_avg
FROM sales
GROUP BY MONTH(date)
WINDOW w AS (ORDER BY MONTH(date)
        RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING);
```

MONTH(date)	SUM(sale)	sliding_avg
3	600	600.0000
4	600	
5	900	
6	600	
7	1200	
8	250	

id	date	sale
1	2017-03-01	200
2	2017-04-01	300
3	2017-05-01	400
4	2017-06-01	200
5	2017-07-01	600
6	2017-08-01	100
7	2017-03-01	400
8	2017-04-01	300
9	2017-05-01	500
10	2017-06-01	400
11	2017-07-01	600
12	2017-08-01	150

SQL 分析 -移动平均图表

```
SELECT MONTH(date), SUM(sale),
        AVG(SUM(sale)) OVER w AS sliding_avg
FROM sales
GROUP BY MONTH(date)
WINDOW w AS (ORDER BY MONTH(date)
        RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING);
```

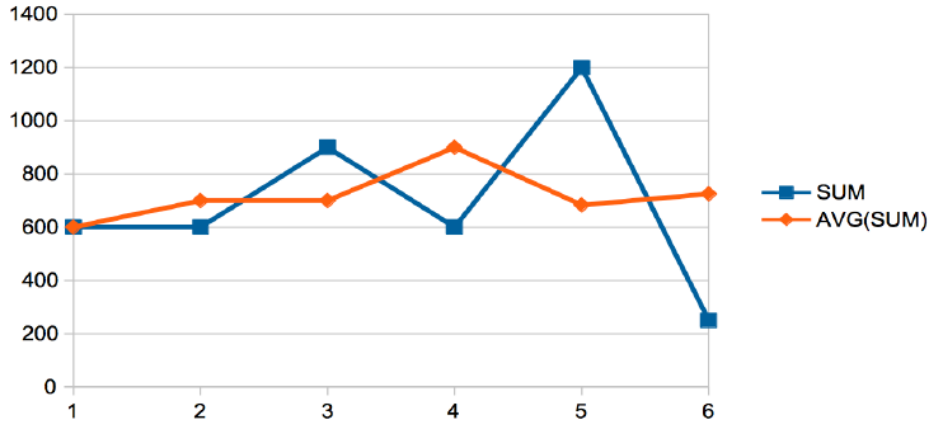
MONTH(date)	SUM(sale)	sliding_avg
3	600	600.0000
4	600	700.0000
5	900	
6	600	
7	1200	
8	250	

id	date	sale
1	2017-03-01	200
2	2017-04-01	300
3	2017-05-01	400
4	2017-06-01	200
5	2017-07-01	600
6	2017-08-01	100
7	2017-03-01	400
8	2017-04-01	300
9	2017-05-01	500
10	2017-06-01	400
11	2017-07-01	600
12	2017-08-01	150

SQL 分析 -移动平均图表

```
SELECT MONTH(date), SUM(sale),
        AVG(SUM(sale)) OVER w AS sliding_avg
FROM sales
GROUP BY MONTH(date)
WINDOW w AS (ORDER BY MONTH(date)
        RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING);
```

MONTH(date)	SUM(sale)	sliding_avg
3	600	600.0000
4	600	700.0000
5	900	700.0000
6	600	900.0000
7	1200	683.3333
8	250	725.0000



WINDOW 函数

支持部分GROUP BY 的聚合函数 –
SUM, AVG, COUNT, MAX, MIN, STD, STDDEV, STDDEV_POP, STD_SAMP, VAR_POP, VAR_SAMP, VARIANCE

Name	Description
CUME_DIST()	Cumulative distribution value
DENSE_RANK()	Rank of current row within its partition, without gaps
FIRST_VALUE()	Value of argument from first row of window frame
LAG()	Value of argument from row lagging current row within partition
LAST_VALUE()	Value of argument from first row of window frame
LEAD()	Value of argument from row leading current row within partition
NTH_VALUE()	Value of argument from N-th row of window frame
NTILE()	Bucket number of current row within its partition.
PERCENT_RANK()	Percentage rank value
RANK()	Rank of current row within its partition, with gaps
ROW_NUMBER()	Number of current row within its partition

MySQL 8.0 - Common Table Expression

- “With” 语句 简化 编写SQL
- 支持 Recursive and Non-Recursive

Non-Recursive CTE

```
WITH t1 AS
(SELECT * FROM tblA
   WHERE a='b'
)
SELECT * FROM t1;
```

Recursive CTE

```
WITH RECURSIVE qn AS
( SELECT 1 AS a
  UNION ALL
  SELECT 1+a FROM qn WHERE a<10
)
SELECT * FROM qn;
```

a
1
2
3
4
5
6
7
8
9
10

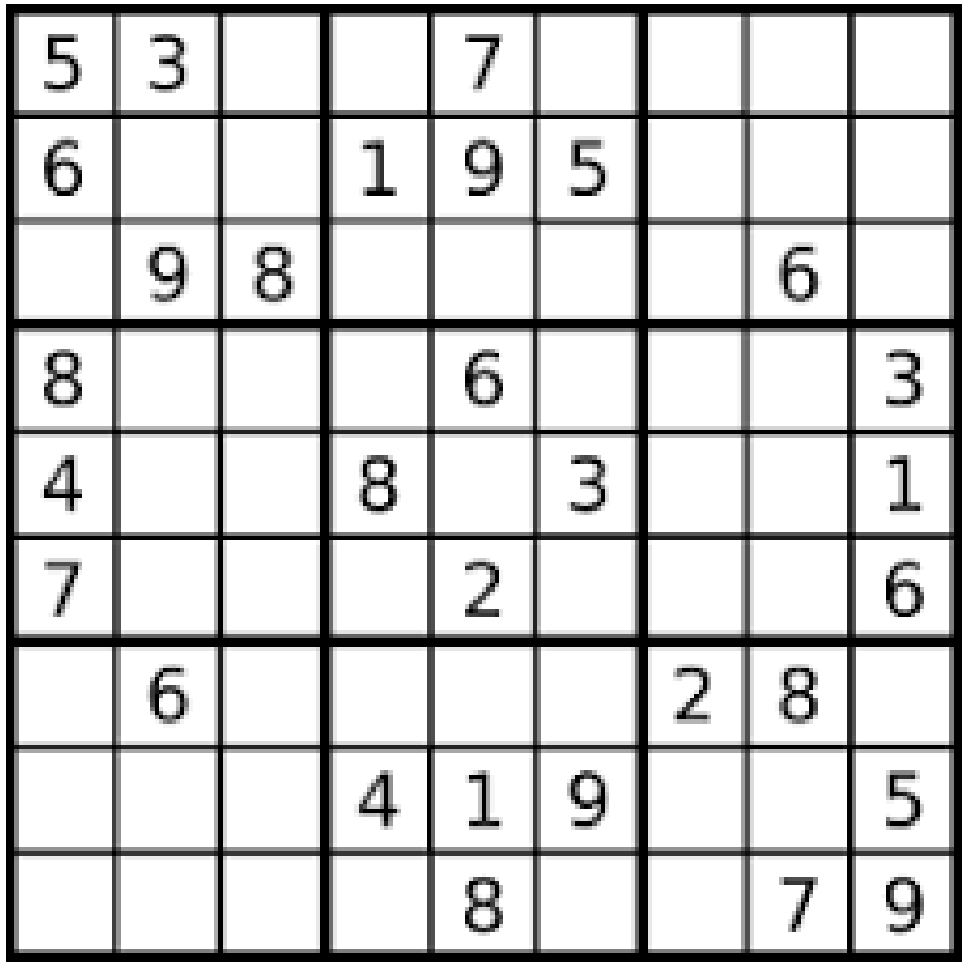
Common Table Expressions

- non-recursive – 定义查询路线，提升性能

```
WITH
   cte1 AS
      (SELECT a,b FROM table1 WHERE b = 'Koufax'),
   cte2 AS
      (SELECT a,b FROM table2)
 SELECT cte1.a, cte1.b FROM cte1, cte2
 WHERE cte1.a = cte2.a;
```

1	Koufax

SUDOKU Solver



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

SUDOKU Solver

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

```
+-------------------------------------------------------------------------------+
| myproblem                                                                     |
+-------------------------------------------------------------------------------+
| 53..7....6..195....98....6.8...6...34..8.3..17...2...6.6....28....419..5....8..79 |
+-------------------------------------------------------------------------------+
+-----------+    +-----------+
| sud       |    | ans       |
+-----------+    +-----------+
| 53..7.... |    | 534678912 |
| 6..195... |    | 672195348 |
| .98....6. |    | 198342567 |
| 8...6...3 |    | 859761423 |
| 4..8.3..1 |    | 426853791 |
| 7...2...6 |    | 713924856 |
| .6....28. |    | 961537284 |
| ...419..5 |    | 287419635 |
| ....8..79 |    | 345286179 |
+-----------+    +-----------+
+----------------------------+-------------------------+
| start_time                 | timediff(sysdate(6),@t) |
+----------------------------+-------------------------+
| 2017-11-30 17:33:24.075760 | 00:00:00.172058         |
+----------------------------+-------------------------+
```

以SQL 定义 SUDOKU 题目

```
SELECT myproblem :=
'53..7....6..195....98....6.8...6...34..8.3..17...2...6.6....28....419..5....8..79' ;
```

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

以递归式 CTE 解答SUDOKU 难题

```
WITH RECURSIVE
      input(sud) as (  select @myproblem ),
      digits(z,lp) as (  select '1', 1
            union all
            select cast(lp+1 as char), lp+1 from digits where lp<9 ),
      x(s,ind) as (  select sud, instr(sud,'.') from input
          union all
          select concat(substr(s,1, ind-1), z, substr(s, ind+1)),
              instr( concat(substr(s,1,ind-1), z, substr(s, ind+1)), '.')
              from x, digits as z
              where ind> 0
              and not exists (
                    select 1 from digits as lp
                    where z.z = substr(s, ((ind-1) div 9) *9 + lp, 1)
                    or z.z = substr(s, ((ind-1)%9) + (lp-1) *9 + 1, 1)
                    or z.z = substr(s, (((ind-1) div 3) %3) * 3
                          + ((ind-1) div 27 ) * 27 + lp
                          + ((lp-1) div 3) * 6, 1)
                    )
              )
SELECT  s as ans FROM x  where ind=0 ;
```

```
+-----------------------------------------------------------------------------+
| ans                                                                         |
+-----------------------------------------------------------------------------+
| 534678912672195348198342567859761423426853791713924856961537284287419635345286179 |
+-----------------------------------------------------------------------------+
```

以递归式 CTE 解答SUDOKU 难题

再拆开 9 X 9 方格

```
+-----------+
| ans       |
+-----------+
| 534678912 |
| 672195348 |
| 198342567 |
| 859761423 |
| 426853791 |
| 713924856 |
| 961537284 |
| 287419635 |
| 345286179 |
+-----------+
```

```
WITH RECURSIVE
      input(sud) as (  select @myproblem ),
      digits(z,lp) as (  select '1', 1
            union all
            select cast(lp+1 as char), lp+1 from digits where lp<9 ),
      x(s,ind) as (  select sud, instr(sud,'.') from input
          union all
          select concat(substr(s,1, ind-1), z, substr(s, ind+1)),
              instr( concat(substr(s,1,ind-1), z, substr(s, ind+1)), '.')
              from x, digits as z
              where ind> 0
              and not exists (
                    select 1 from digits as lp
                    where z.z = substr(s, ((ind-1) div 9) *9 + lp, 1)
                    or z.z = substr(s, ((ind-1)%9) + (lp-1) *9 + 1, 1)
                    or z.z = substr(s, (((ind-1) div 3) %3) * 3
                          + ((ind-1) div 27 ) * 27 + lp
                          + ((lp-1) div 3) * 6, 1)
                    ) ),
      my19(n) AS (
            SELECT 1 AS n
            UNION ALL
            SELECT 1+n FROM my19 WHERE n<9)
SELECT substr(s,(n-1)*9 + 1,9) as ans from x, my19 where ind=0 ;
```

总结

- MySQL 8.0 RC 已经 有 SQL WINDOW 函数的支持
- SQL的WINDOW 函数 提供更好的分析工具
- 可读性更高，查询性能/速度更快
- Common Table Expression (CTE)可以大大帮助分析
- MySQL 8.0 RC 还有更多的新功能
  - New Design - Data Dictionary
  - GIS SRS ID 支持
  - Security – ROLE, 动态权限，
  - 高可用 – MySQL InnoDB Cluster
  - …

MySQL 8.0 RC1 分享 - 总结

Download

https://dev.mysql.com/downloads/mysql/8.0.html

What's New in MySQL 8.0

https://dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html

MySQL 8.0 RC1 – Highlights

https://mysqlserverteam.com/mysql-8-0-rc1-highlights/

Integrated Cloud

Applications & Platform Services


ORACLE®