Extra Stuff

CSC365

Common Table Expressions (CTE)

- Introduced into standard SQL in order to simplify various classes of SQL Queries for which a derived table was just unsuitable.
- CTE is a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.
- Syntax

  -
    ```
    WITH cte AS (SELECT 1 AS col_a, 2 AS col_b)

    SELECT * FROM cte AS t1 JOIN cte AS t2;
    ```

Not Supported in MySQL before v8!

Common Table Expressions (CTE) - WITH

```
WITH <CTE1 name> AS (
  SELECT ...
), <CTE2 name> AS (
  SELECT ...
)
SELECT * FROM <table>, <CTE1 name>, <CTE2 name> ...
```

CTE / WITH Example

```
WITH cte_wins AS (select t.id, t.name, count(g.date) as wins
from Game g RIGHT JOIN Team t ON t.id = g.home_team_id
and g.score_home > score_away or t.id = g.away_team_id
and g.score_away > g.score_home group by t.id)
```

```
SELECT id, name FROM Team WHERE id not in (SELECT
w1.id FROM cte_wins w1, cte_wins w2 WHERE w1.wins <
w2.wins);
```

WITH Recursive

WITH RECURSIVE allows a WITH query to refer to its *own* output. To sum the numbers from 1 to 100:

```
WITH RECURSIVE t(n) AS (
  SELECT 1
  UNION ALL
  SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

Window Functions

Window functions allow us to perform calculations across a result set. Somewhat like aggregation / GROUP BY, but window functions *do not* collapse rows. All rows are preserved.

Basic syntax (in SELECT clause):

Rows with the same value for <column list> fall into the same partition (similar to GROUP BY or DISTINCT)

<window function> OVER (PARTITION BY <column list> [ORDER BY <colums>])

Available window functions include familiar aggregates (SUM, MIN, MAX, COUNT, AVG) as well as a few new functions: RANK, ROW_NUMBER, etc.

```
mysql> SELECT
         year, country, product, profit,
         SUM(profit) OVER() AS total_profit,
         SUM(profit) OVER(PARTITION BY country) AS country_profit
       FROM sales
       ORDER BY country, year, product, profit;
+------+---------+------------+--------+--------------+----------------+
| year | country | product    | profit | total_profit | country_profit |
+------+---------+------------+--------+--------------+----------------+
| 2000 | Finland | Computer   |   1500 |         7535 |           1610 |
| 2000 | Finland | Phone      |    100 |         7535 |           1610 |
| 2001 | Finland | Phone      |     10 |         7535 |           1610 |
| 2000 | India   | Calculator |     75 |         7535 |           1350 |
| 2000 | India   | Calculator |     75 |         7535 |           1350 |
| 2000 | India   | Computer   |   1200 |         7535 |           1350 |
| 2000 | USA     | Calculator |     75 |         7535 |           4575 |
| 2000 | USA     | Computer   |   1500 |         7535 |           4575 |
| 2001 | USA     | Calculator |     50 |         7535 |           4575 |
| 2001 | USA     | Computer   |   1200 |         7535 |           4575 |
| 2001 | USA     | Computer   |   1500 |         7535 |           4575 |
| 2001 | USA     | TV         |    100 |         7535 |           4575 |
| 2001 | USA     | TV         |    150 |         7535 |           4575 |
+------+---------+------------+--------+--------------+----------------+
```

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 last 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

Window Function Example

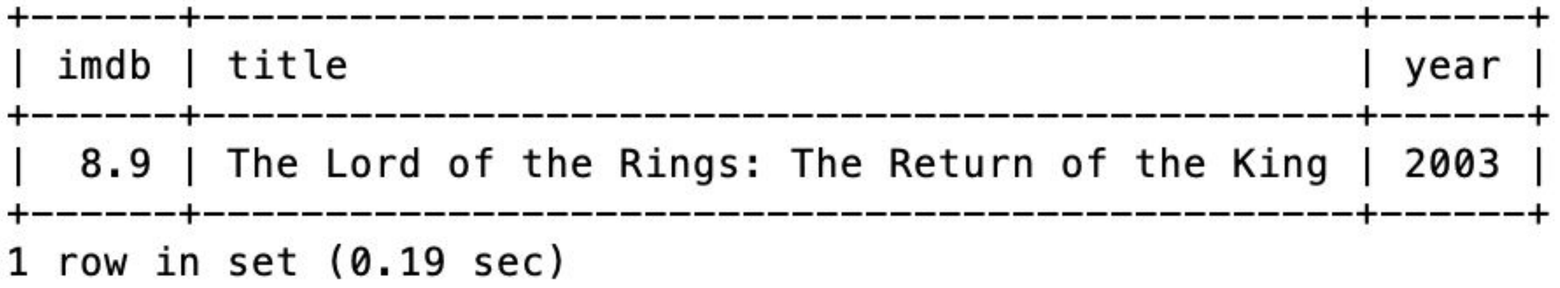
```
SELECT m1.imdb, m1.title, m1.year

FROM Movies m1

JOIN Movies m2 ON m1.imdb < m2.imdb

GROUP BY m1.imdb, m1.title, m1.year

HAVING COUNT(*) = 1;
```

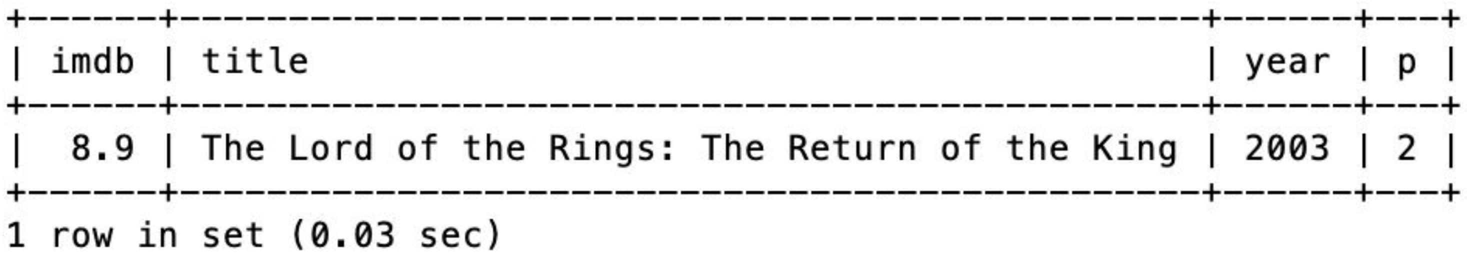


```
+------+------------------------------------------------+------+
| imdb | title                                          | year |
+------+------------------------------------------------+------+
|  8.9 | The Lord of the Rings: The Return of the King | 2003 |
+------+------------------------------------------------+------+
1 row in set (0.19 sec)
```

Window Function Example

I should have used RANK()

```
SELECT *
FROM (
    SELECT m.imdb, m.title, m.year,
    ROW_NUMBER() OVER(ORDER BY m.imdb desc) as p
    FROM Movies m) as t
WHERE t.p = 2;
```



```
+------+------------------------------------------+------+---+
| imdb | title                                    | year | p |
+------+------------------------------------------+------+---+
|  8.9 | The Lord of the Rings: The Return of the King | 2003 | 2 |
+------+------------------------------------------+------+---+
1 row in set (0.03 sec)
```

Window Function Example

```
SELECT m.director, AVG(m.imdb) as a,
RANK() OVER(ORDER BY AVG(m.imdb) desc)  as r
FROM Movies m GROUP BY m.director;
```

```
+---------------------------------+--------------------+------+
| director                        | a                  | r    |
+---------------------------------+--------------------+------+
| Christopher Nolan               |  8.640000152587891 |    1 |
| Quentin Tarantino               |                8.5 |    2 |
| Lee Unkrich                     |  8.300000190734863 |    3 |
| James Gunn                      |  8.100000381469727 |    4 |
| Alejandro G. Iñárritu           |  8.100000381469727 |    4 |
| Anthony Russo                   |                  8 |    6 |
| Peter Jackson                   |  7.949999928474426 |    7 |
| Don Hall                        |  7.900000095367432 |    8 |
| Martin Scorsese                 | 7.883333651224775  |    9 |
| Edward Zwick                    |  7.849999904632568 |   10 |
```

Window Function Examples

```
SELECT DeptID, EmpID, AnnualSalary,
  AVG(AnnualSalary) OVER (PARTITION BY DeptID) as DeptAverage
FROM employee;

SELECT DeptID, EmpID, HireDate,
  RANK() OVER (PARTITION BY DeptID ORDER BY HireDate) as
HireOrder
FROM employee
where DeptID = 'Engineering';
-- How might we list just the most recently hired employee(s) in each department?
-- RANK() vs DENSE_RANK()
```

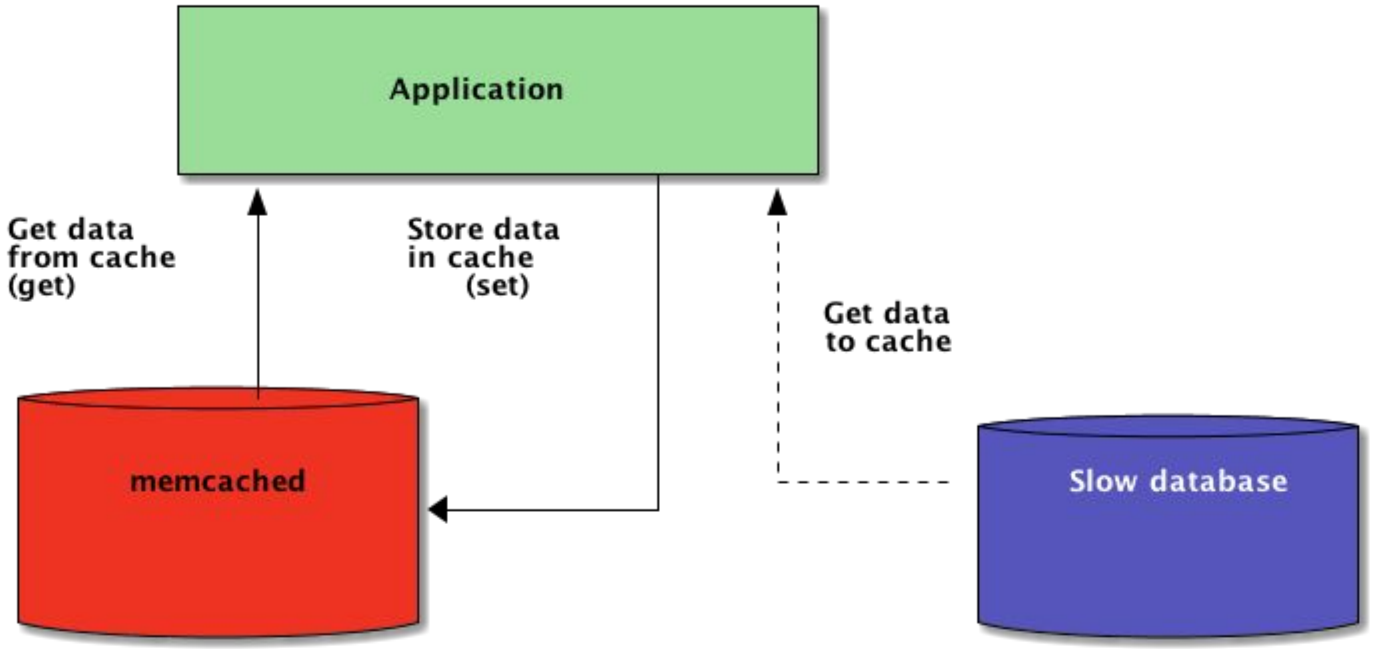
Window Function Summary

- When compared to subqueries...
  - Window functions can improve performance
  - Code is typically more compact and readable
  - Additional capabilities/functions (RANK(), ROW_NUMBER(), etc.)
- Not supported by all RDBMSs / versions
  - Available only in the latest version (8.0) of MySQL
  - No support in SQLite

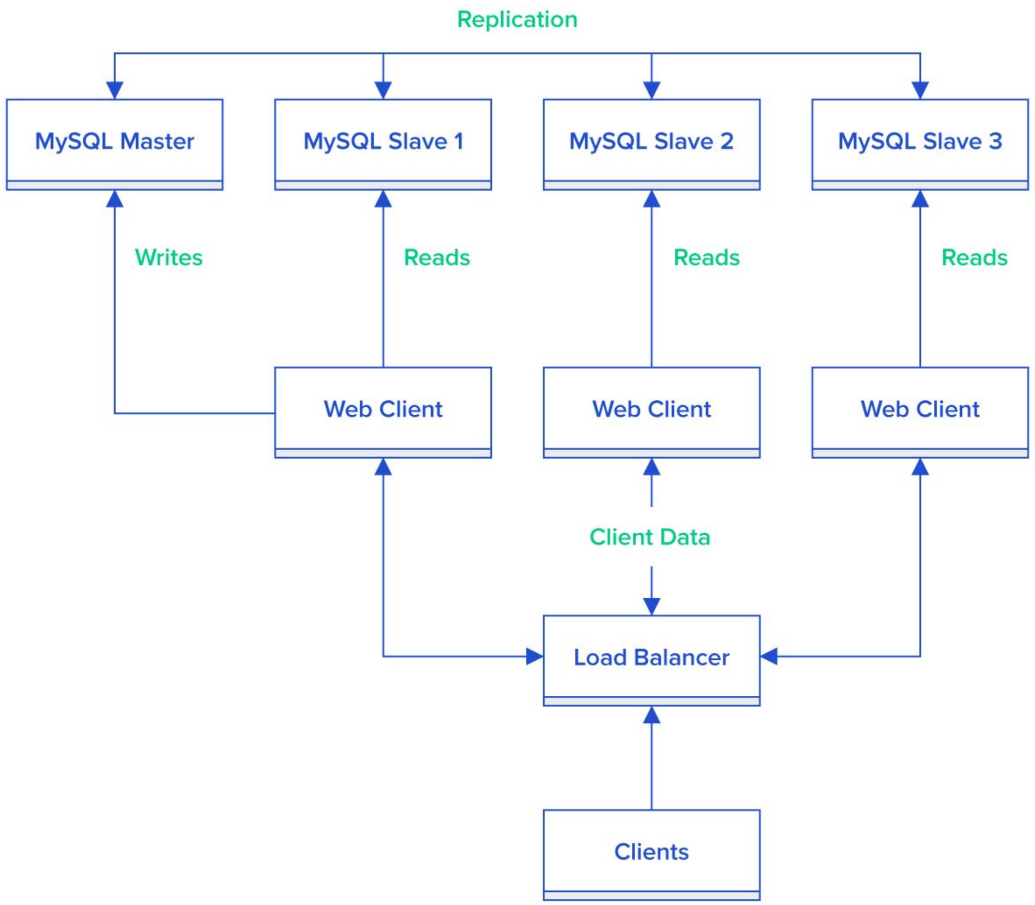
Read more at: https://dev.mysql.com/doc/refman/8.0/en/window-functions.html

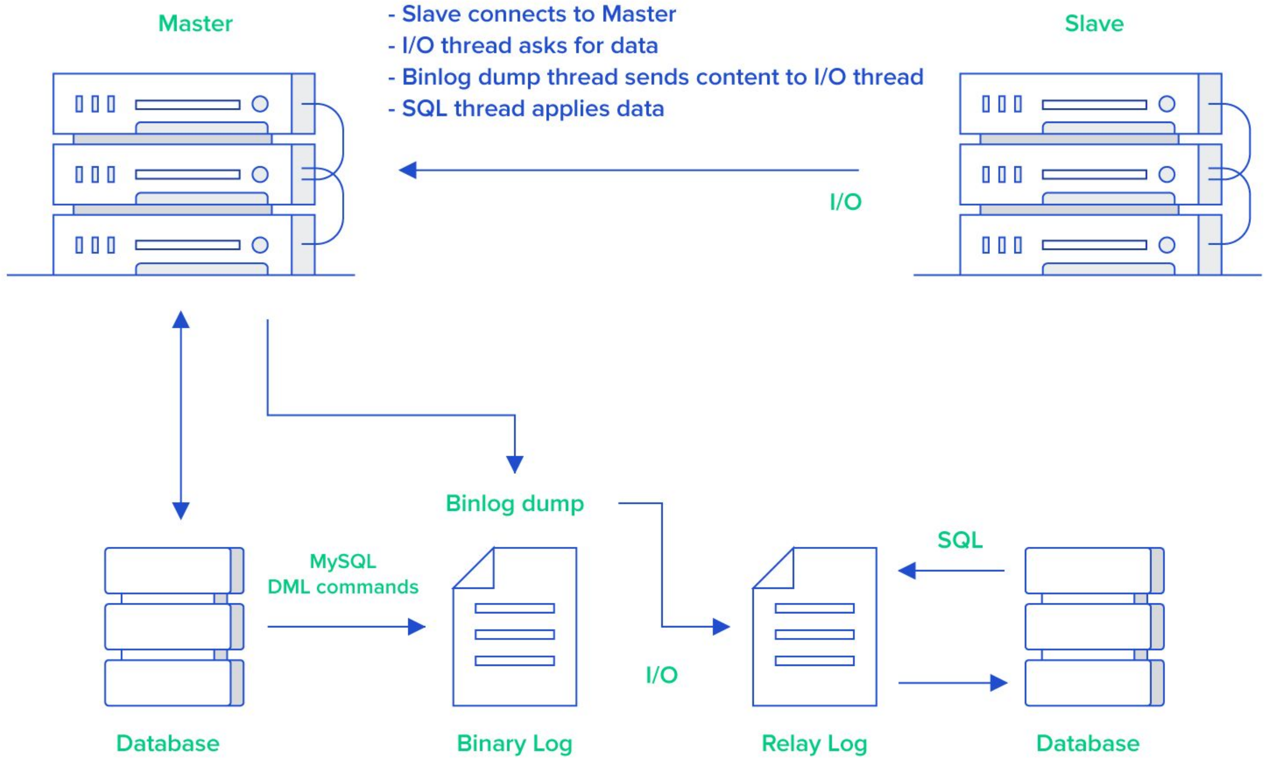
High Performance and Availability

Caching


Application
Get data from cache (get)
Store data in cache (set)
Get data to cache
memcached
Slow database

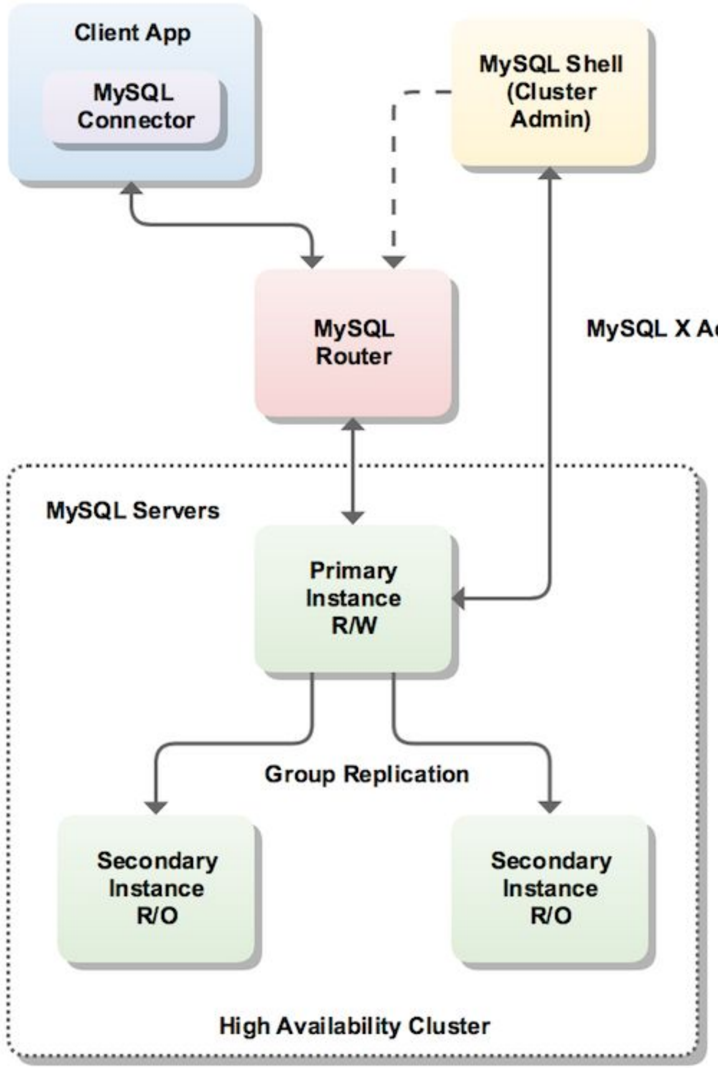
Replication




Master
- Slave connects to Master
- I/O thread asks for data
- Binlog dump thread sends content to I/O thread
- SQL thread applies data
Slave
I/O
Binlog dump
SQL
MySQL
DML commands
I/O
Database
Binary Log
Relay Log
Database

Read more at: https://dev.mysql.com/doc/refman/8.0/en/replication.html

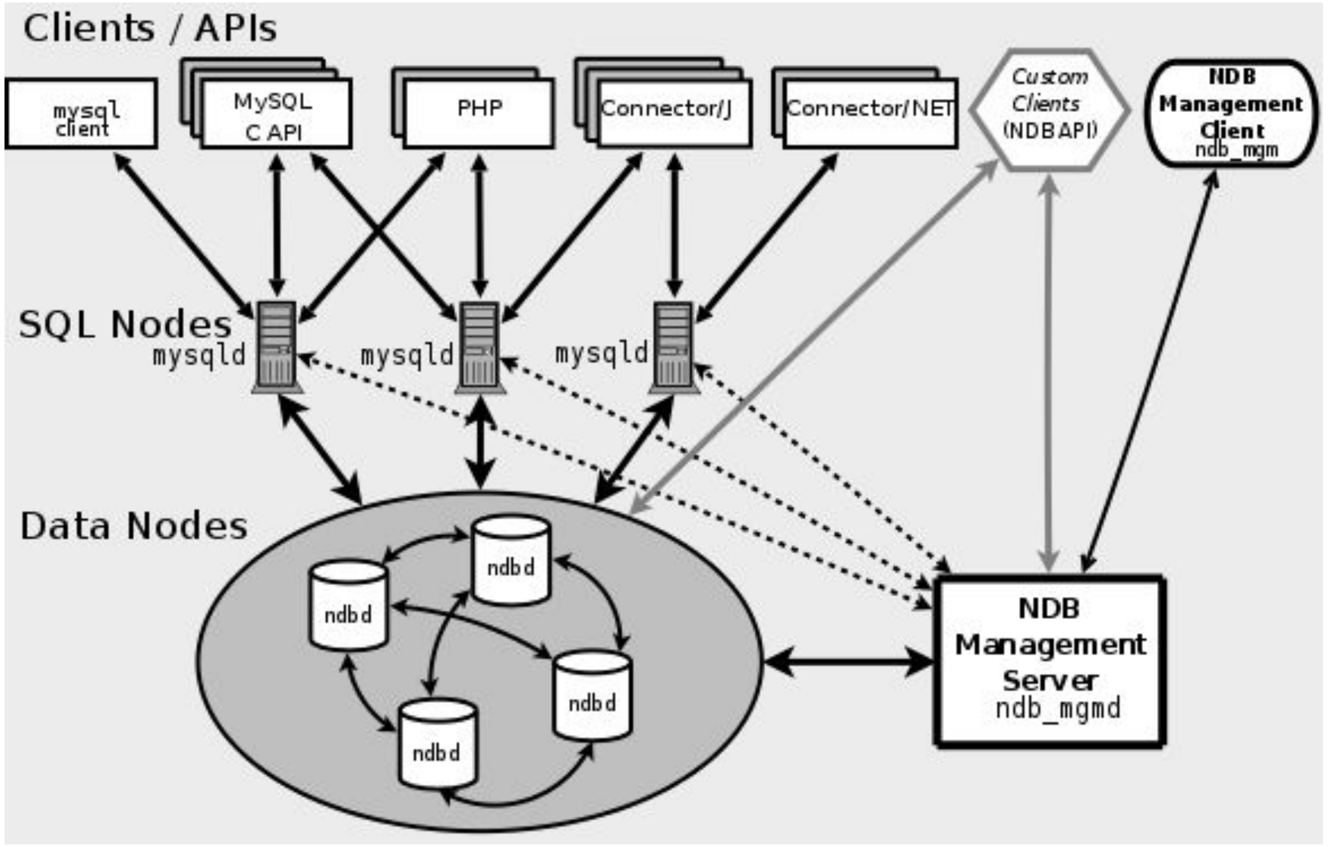
INNODB Cluster



Read more at

https://dev.mysql.com/doc/refman/8.0/en/mysql-innodb-cluster-userguide.html

NDB Cluster



Read more at : https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html