ORACLE

ORACLE

Modern SQL in MySQL

Norvald H. Ryeng
Software Development Director
MySQL Optimizer Team
February 24, 2020

github.com/nryeng/abakus-v20

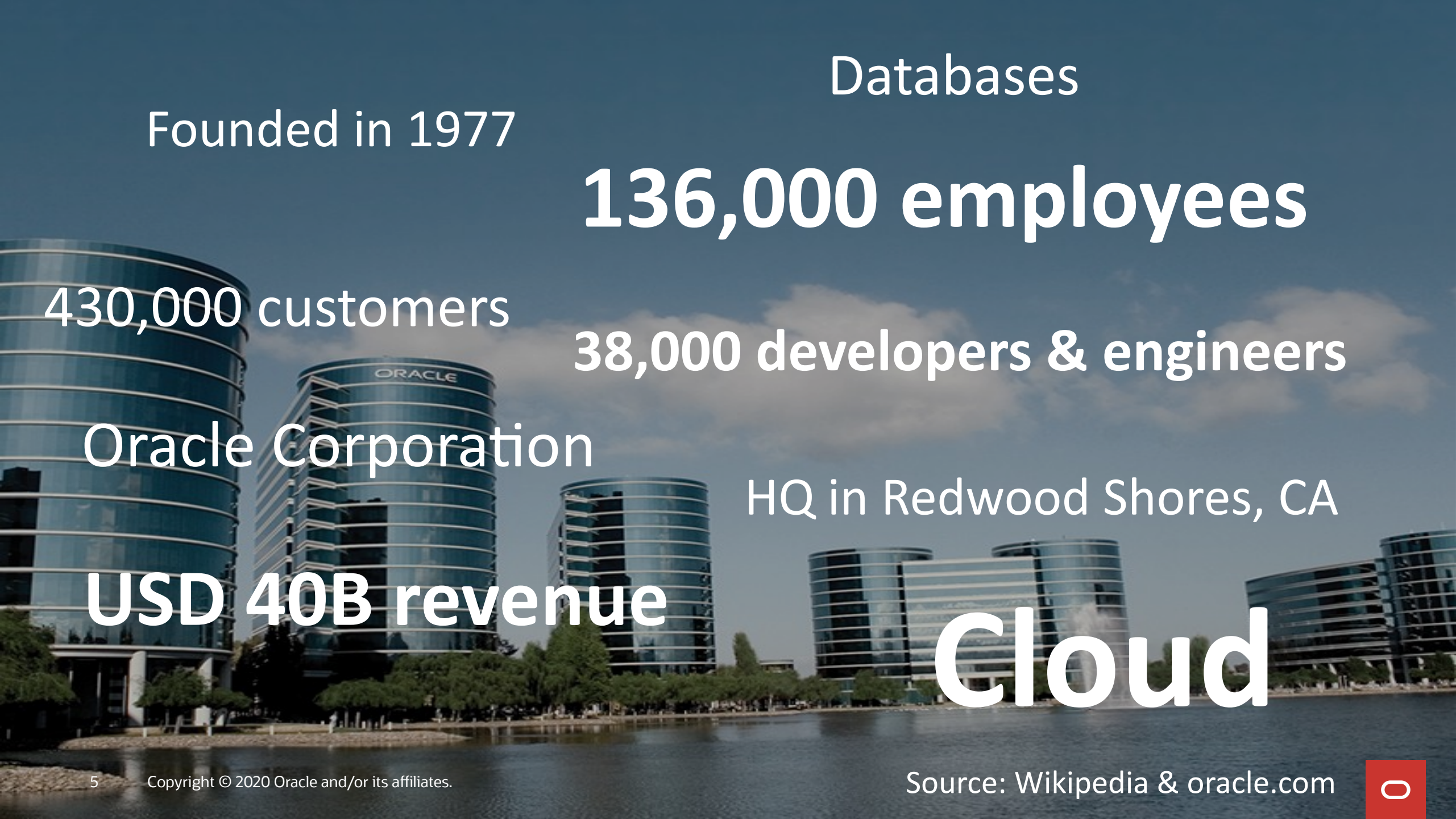
Program agenda

1. About Oracle
2. Intro to MySQL
3. Modern SQL
4. Common table expressions (CTEs)
5. Window functions
6. JSON in SQL
7. Spatial data

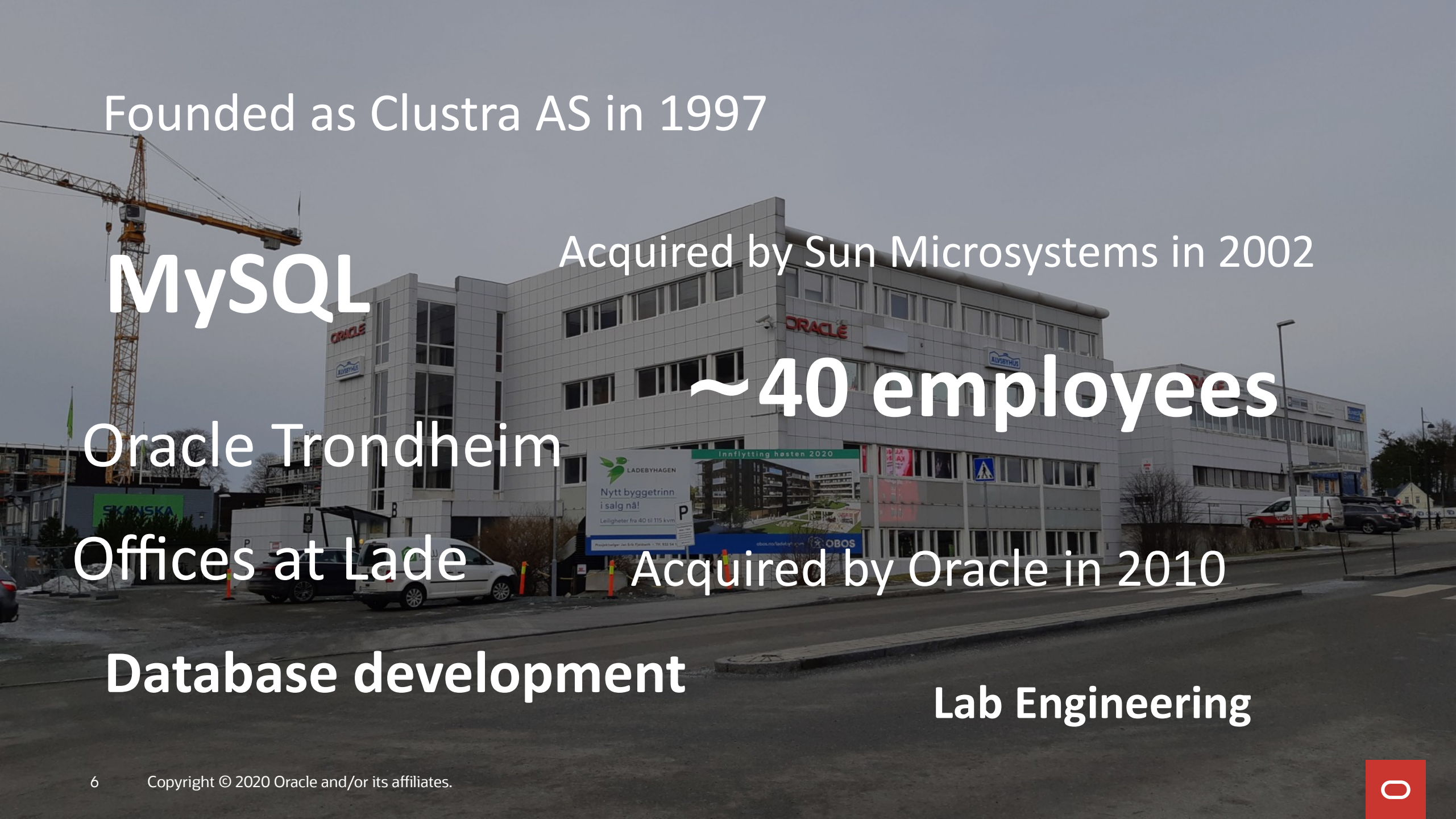
19:15 Bus to ØX Taproom

ORACLE

About Oracle



Source: Wikipedia & oracle.com



Founded as Clustra AS in 1997

MySQL

Acquired by Sun Microsystems in 2002

~40 employees

Oracle Trondheim

Offices at Lade

Acquired by Oracle in 2010

Database development

Lab Engineering

ORACLE

Intro to MySQL

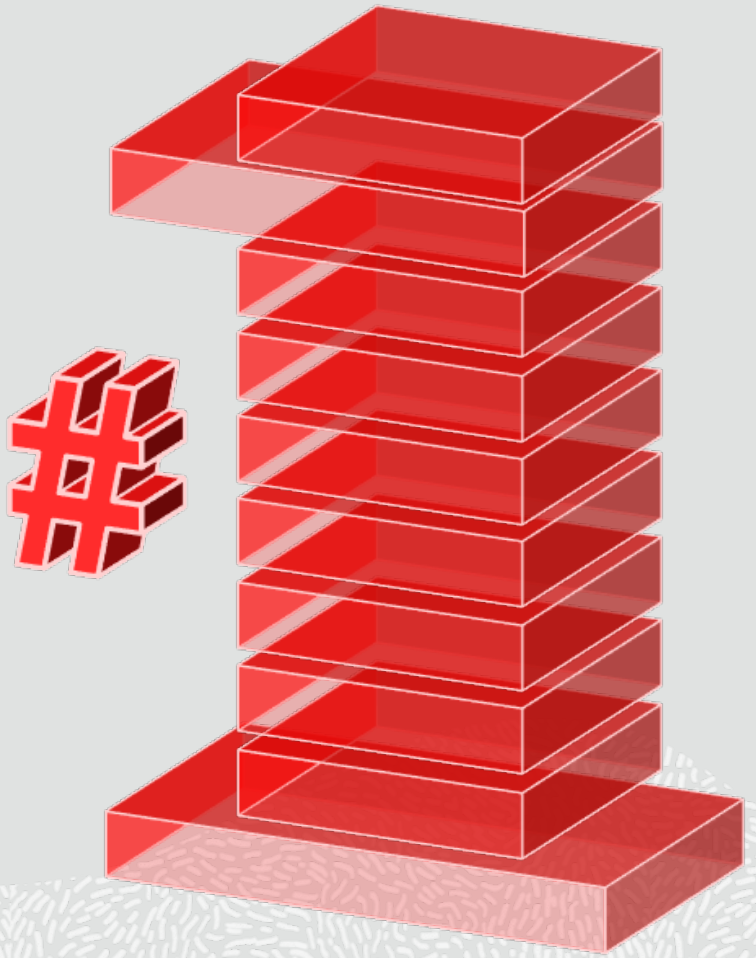
MySQL Server

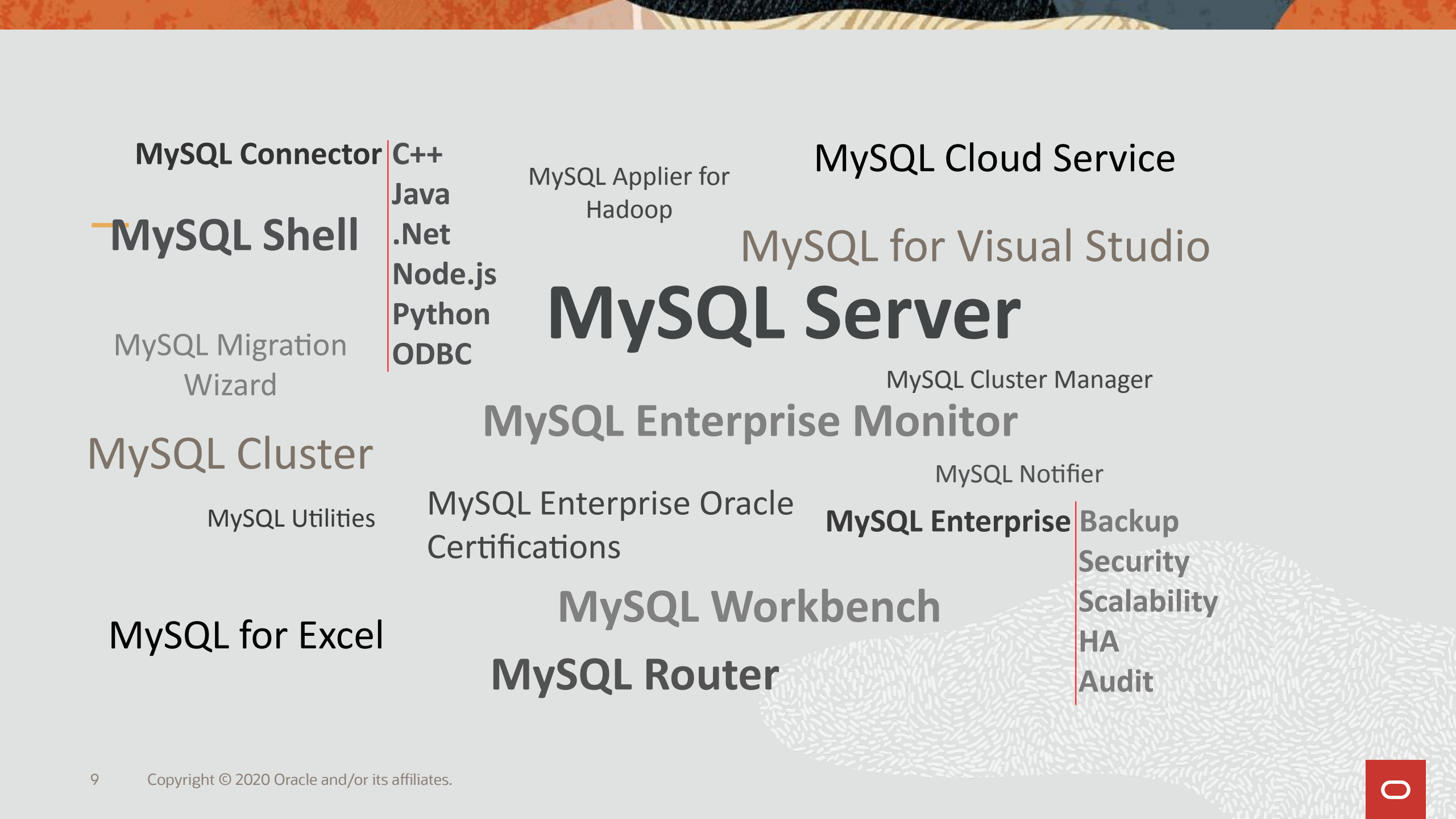
- **A robust ACID compliant DBMS**
- **Dual license**

  Community: Open source (GPLv2)

  https://github.com/mysql/

  Commercial: Proprietary license
- **The world's most popular open source DBMS**
- **Abundantly present in major Linux distributions and hosting providers**
- **Used by many of the largest web properties**





ORACLE

Modern SQL

The history of SQL

- **1974 SQL appears**
- **SQL-86 (ANSI)**
  Defines pronunciation as "es queue el"
- **SQL-87 (ISO)**
- **SQL-89**
- **SQL-92**
  Major revision
  Transaction isolation levels
  VARCHAR
  DATE

"Modern SQL"

- **SQL:1999**
  Common table expressions (CTEs)
  Object orientation
- **SQL:2003**
  XML
  Window functions
- **SQL:2006**
  More XML
- **SQL:2008**
- **SQL:2011**
  Focus on temporal features
- **SQL:2016**
  JSON

Modern SQL

- **No longer strictly relational**
  - Object-relational model
  - XML
  - JSON
- **Complex data types**
- **The best of all worlds**
  - Structured and semi-structured
  - Schemaful and schemaless
- **Rich analytical queries**
  - OLAP
  - CTEs
  - Window functions
- **Temporal support**
  - Versioned tables
  - Temporal predicates
- **Spatial support**
  - SQL/MM Part 3: Spatial

ORACLE

Common table expressions (CTEs)

CTEs as aliases / ad hoc views

- Give a name to a subquery
- Use it as a table
- Improves readability
- Similar to a view but for a single query

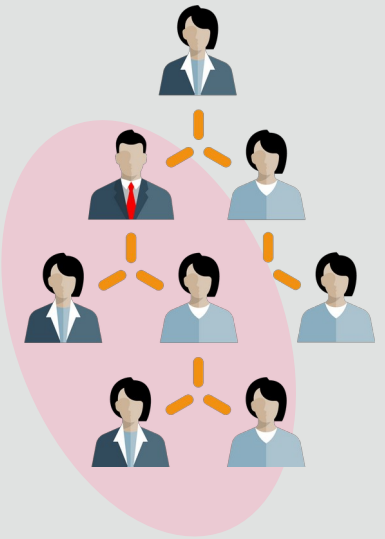
```
WITH course_info AS (
   SELECT
      id AS course_id,
      courses.name AS course_name,
      employees.number AS lecturer_id,
      employees.name AS lecturer_name
   FROM courses, employees
   WHERE lecturer = number
)
SELECT *
FROM course_info
WHERE lecturer_name = 'Bob';
```

Recursive CTEs

```
WITH RECURSIVE cte AS (
      SELECT … FROM table_name        /* "seed" SELECT */
   UNION [DISTINCT|ALL]
      SELECT … FROM cte, table_name  /* "recursive" SELECT */
) SELECT … FROM cte
```

- **A recursive CTE refers to itself in a subquery**
- **The "seed" SELECT is executed once to create the initial data subset**
- **The "recursive" SELECT is executed repeatedly**
  - Stops when iteration doesn't produce any new rows
  - Set cte_max_recursion_depth to limit recursion (default 1000)

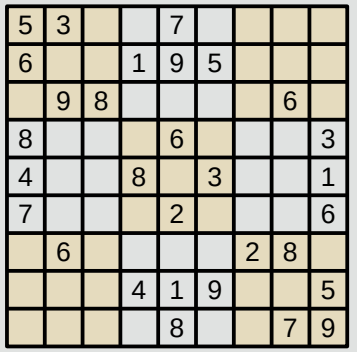
Recursive CTEs



- Queries on hierarchical data
- Traverse trees

```
WITH RECURSIVE subordinates AS (
      SELECT number, name
      FROM employees
      WHERE manager = (SELECT number FROM employees WHERE name = 'Betty')
   UNION ALL
      SELECT e.number, e.name
      FROM employees AS e JOIN subordinates AS s ON e.manager = s.number
)
SELECT name FROM subordinates;
```

Solving sudokus with MySQL CTEs



https://www.percona.com/blog/2017/11/22/sudoku-recursive-common-table-expression-solver/

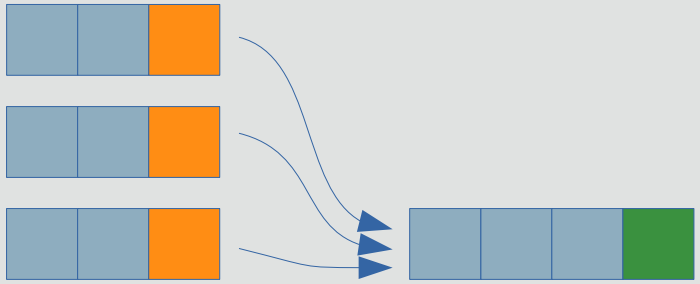
```
WITH RECURSIVE
   input (sud) AS (SELECT '53..7....6..195....98....6.8...6...34..8.3..17...2...6.6....28....419..5....8..79'),
   digits (z, lp) AS (SELECT '1', 1 UNION ALL SELECT CAST(lp + 1 AS CHAR), lp + 1 FROM digits WHERE lp < 9),
   x (s, ind) AS (
      SELECT sud, INSTR(sud, '.') FROM input
      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 FROM x WHERE ind=0;
```

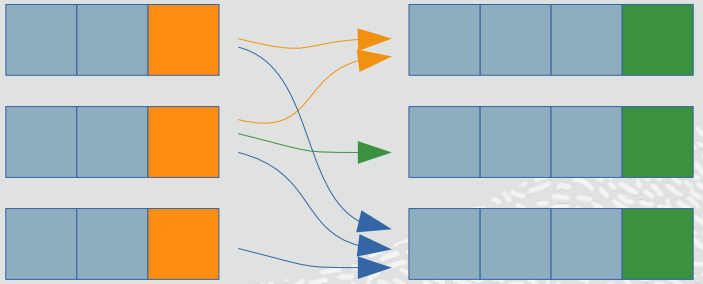
Window functions

Window functions

- Similar to aggregation functions
- Computes one value based on multiple rows
- Does not group



Aggregation function



Window function

Window functions example

```
SELECT
   name,
   dept_id,
   salary,
   SUM(salary) OVER (PARTITION BY dept_id) AS dept_total
FROM employee
ORDER BY dept_id, name;
```

The **OVER** keyword signals a window function

PARTITION ⇒ disjoint set of rows in result set

name	dept_id	salary
Alice	NULL	1000000
Betty	10	950000
Camilla	10	640000
Chris	10	650000
Dave	10	500000
Denise	10	51000
Chandler	10	690000
Charlie	20	700000
Cherise	20	750000
Bob	30	900000

Window functions example

```
SELECT
   name,
   dept_id,
   salary,
   SUM(salary) OVER (PARTITION BY dept_id) AS dept_total
FROM employee
ORDER BY dept_id, name;
```

The **OVER** keyword signals a window function

PARTITION ⇒ disjoint set of rows in result set

name	dept_id	salary	dept_total
Alice	NULL	1000000	1000000
Betty	10	950000	3250000
Camilla	10	640000	3250000
Chris	10	650000	3250000
Dave	10	500000	3250000
Denise	10	51000	3250000
Chandler	10	690000	3250000
Charlie	20	700000	2140000
Cherise	20	750000	2140000
Bob	30	900000	900000

ORACLE

JSON in SQL

JSON in MySQL

- **A special data type for JSON documents**
  - Checks that it is valid JSON
  - Native data type in MySQL
- **JSON Schema as column constraint**
- **Functions to retrieve/update data**
- **Indexing for fast access**
  - Using generated columns
- **Based on the SQL/JSON standard**

JSON functions

- JSON_ARRAY_APPEND()
- JSON_ARRAY_INSERT()
- JSON_ARRAY()
- JSON_CONTAINS_PATH()
- JSON_CONTAINS()
- JSON_DEPTH()
- JSON_EXTRACT()
- JSON_INSERT()
- JSON_KEYS()
- JSON_LENGTH()
- JSON_MERGE[_PRESERVE]()
- JSON_OBJECT()
- JSON_QUOTE()
- JSON_REMOVE()
- JSON_REPLACE()
- JSON_SEARCH()
- JSON_SET()
- JSON_TYPE()
- JSON_UNQUOTE()
- JSON_VALID()
- JSON_PRETTY()
- JSON_STORAGE_SIZE()
- JSON_STORAGE_FREE()
- JSON_ARRAYAGG()
- JSON_OBJECTAGG()
- JSON_MERGE_PATCH()
- JSON_TABLE()

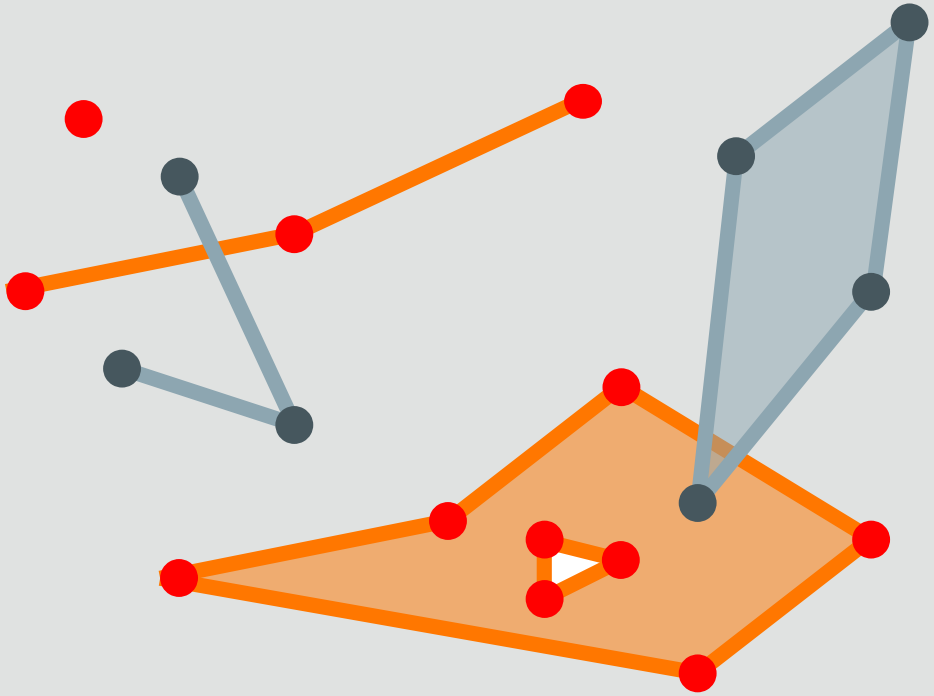
ORACLE

Spatial data

Spatial data

- **Geometric object**
  - Point, linestrings, polygons
  - Collections
- **Spatial reference systems (SRSs)**
- **Storage and indexing**
- **Functions for processing geometries**
- **Query optimization**

https://github.com/nryeng/mysql-8.0-gis-demos



Thank you

Norvald H. Ryeng

Software Development Director
MySQL Optimizer Team

Safe harbor statement

The preceding 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, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.