ORACLE

Oracle数据库创新21c
之 融合数据库新特性

Oracle
李帅

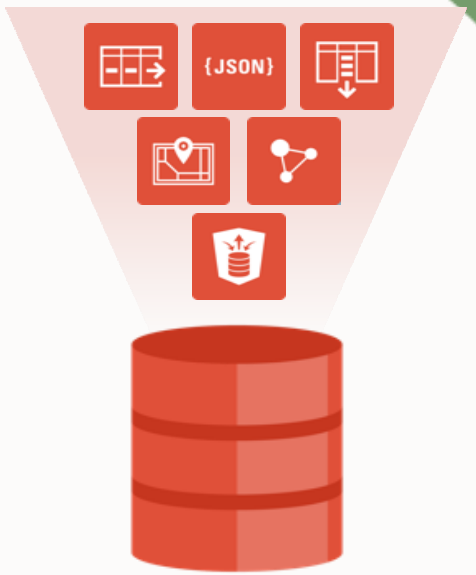
Oracle 免责申明

以下内容旨在概述 Oracle 产品的总体发展方向。该内容仅供参考，不可纳入任何合同。本文档不承诺提供任何材料、代码或功能，也不应将其作为购买决策的依据。

此处所述有关 Oracle 产品任何特性或功能的开发、发布、时间安排和定价可能会发生变更，且均由甲骨文公司自行决定。

策略 |为客户提供全球最佳的多模数据库

面向未来，应对不断变化的业务需求



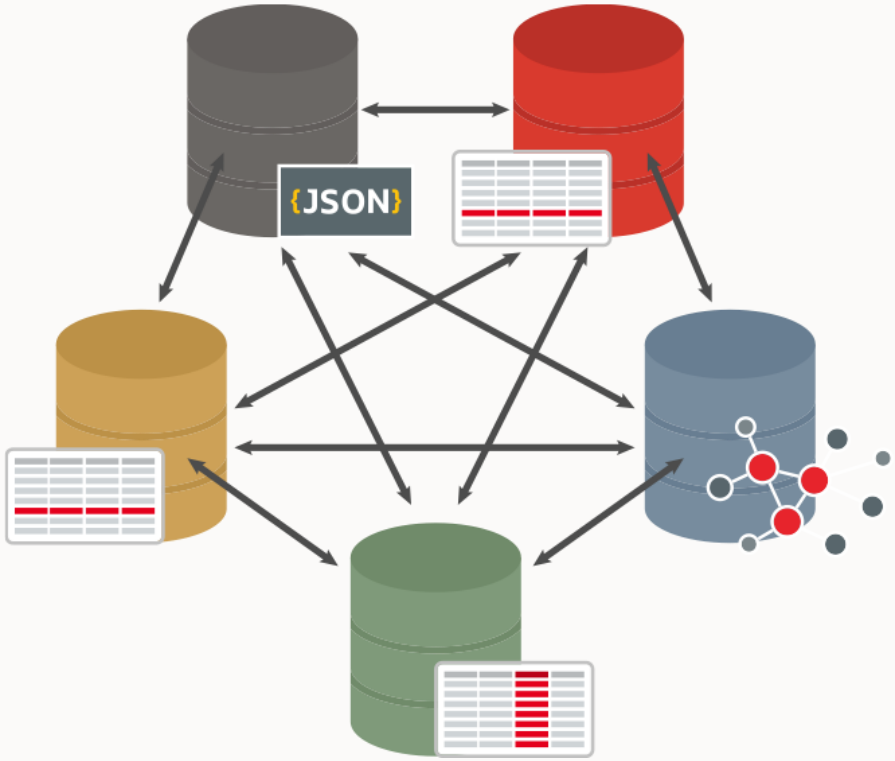
支持任何数据
Relational, JSON, graph, spatial, text, OLAP, XML, multimedia

支持任何工作负载
交易型, 分析,内存, 物联网, 流, 区块链

最适合开发人员和分析师
集成微服务，事件，REST，SaaS，ML，CI / CD，低代码

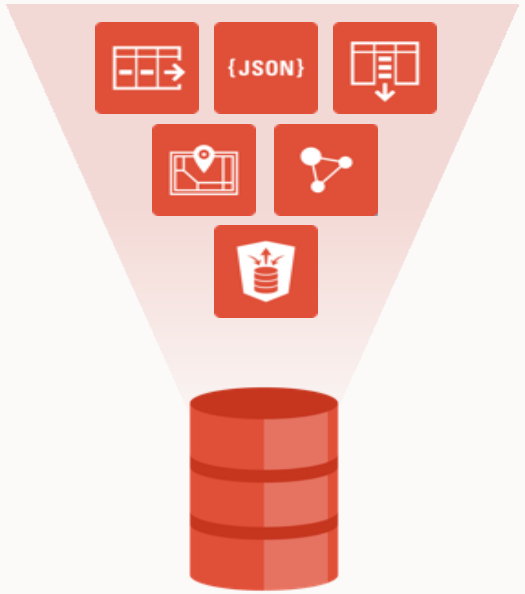
多个专用数据库导致数据碎片化

- 部署的多个单一用途数据库都会使企业数据体系结构碎片化
- 集成碎片化的数据使分析和应用程序开发极为复杂
- 多种类型的单一数据库会带来更多的运营和安全复杂性以及风险



支持融合数据库的更多创新

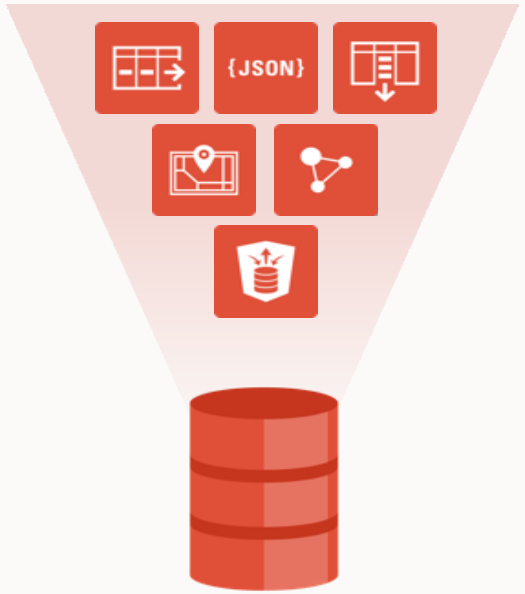
21c



- 原生JavaScript
- JSON速度和灵活性
- In-Memory 改进
- 自动区域图
- 多租户Data Guard
- SQL宏
- 自动机器学习
- 原生区块链表
- 图的性能
- 持久化内存存储
- Sharding 增强
- 多租户安全
- 安全增强

支持融合数据库的更多创新

21c



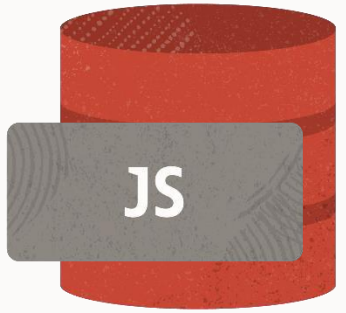
- 原生JavaScript
- JSON速度和灵活性
- In-Memory 改进
- 自动区域图
- 多租户Data Guard
- SQL宏
- 自动机器学习
- 原生区块链表
- 图的性能
- 持久化内存存储
- Sharding 增强
- 多租户安全
- 安全增强

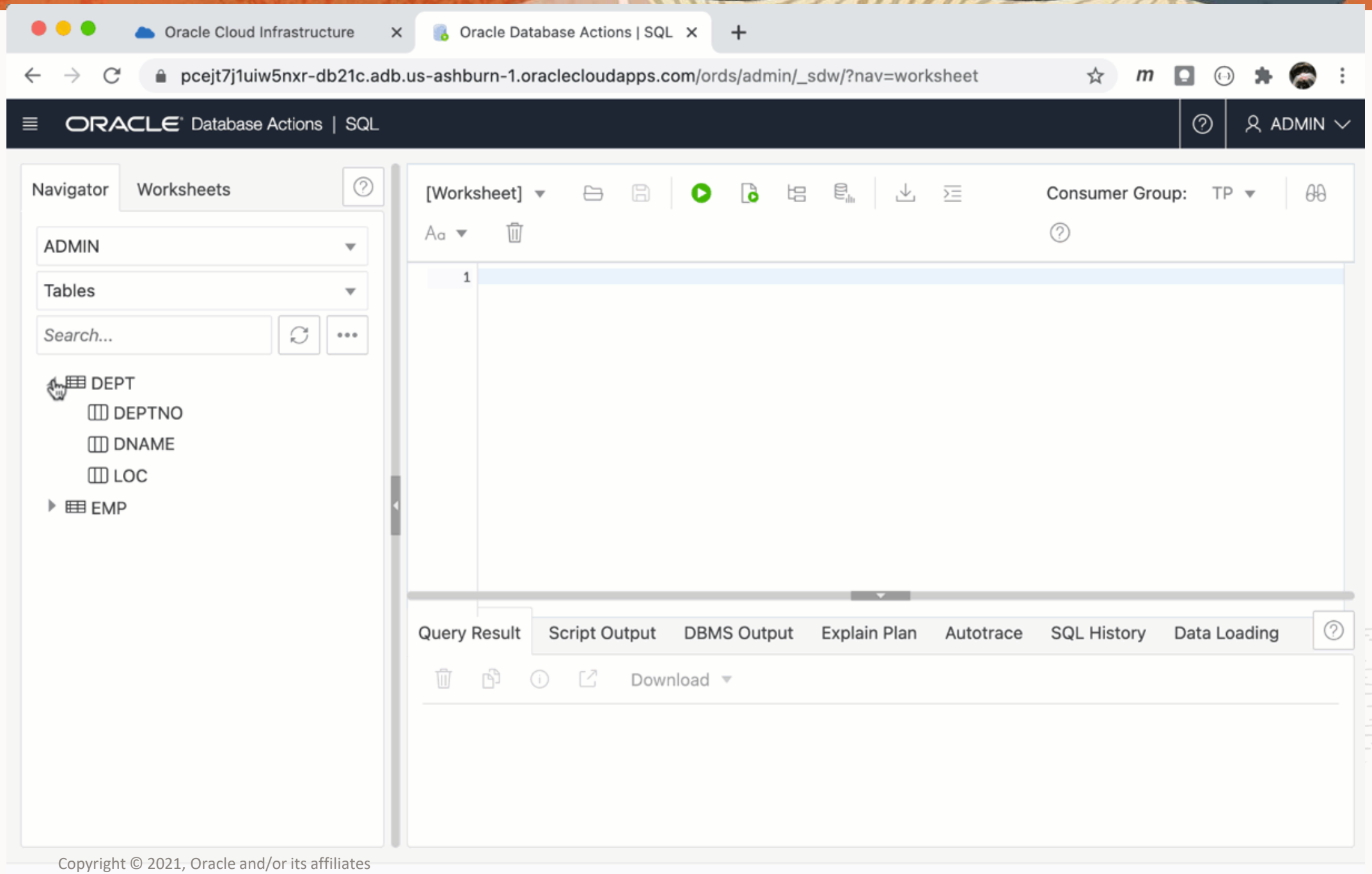
原生的 JavaScript

在数据库内部执行JavaScript

在嵌入式Graal多语言引擎（MLE）中运行数据处理逻辑

- 在数据所在的Oracle数据库内部执行JavaScript代码
- JavaScript数据类型自动映射到Oracle Database数据类型，反之亦然
- 在PL / SQL和JavaScript之间无缝交换数据
- JavaScript代码本身可以通过内置JavaScript模块执行PL / SQL和SQL
- 使开发人员能够有效地使用现代编程语言工作
- 支持在APEX内部执行JavaScript





JSON速度和灵活性

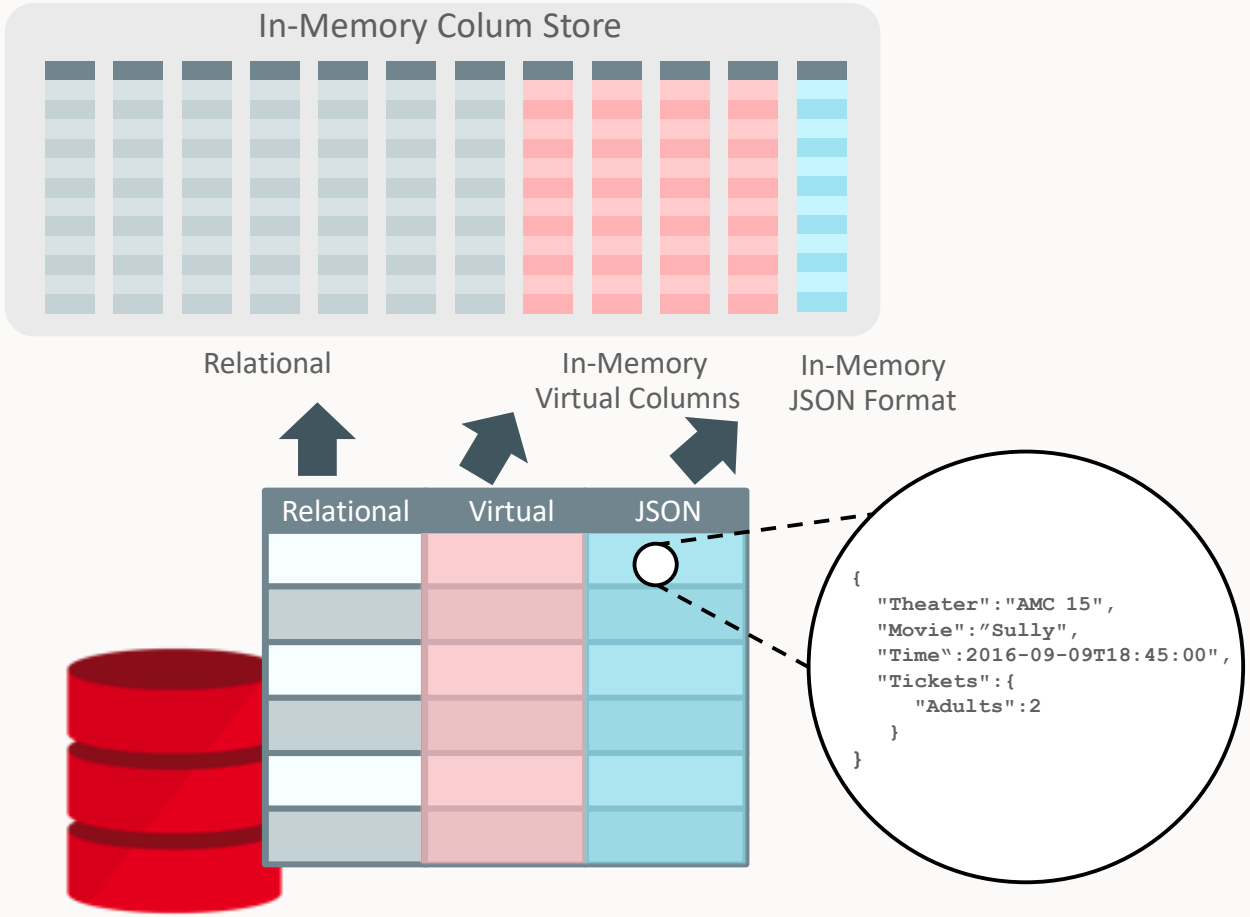
Oracle数据库中的原生JSON数据类型

- SQL和PL / SQL中使用新的高性能二进制JSON数据类型
- 简化Apps与数据库使用相同的JSON数据格式
- 简单快速的跨集合连接或聚合
- 创建索引以加速查询
- ACID和声明式并行SQL分析跨越所有数据格式
- JSON的简化SQL语法
- 基于OSON- 二进制存储
- 支持in memory


```
CREATE TABLE J_PURCHASEORDER (
   id            INTEGER PRIMARY KEY,
   po_document JSON
);
```

扫描速度提升 10X | 更新速度提升 4X

In-Memory JSON 数据类型



- 使用优化的二进制格式，新的JSON数据类型列内联填充为8KB
- 对JSON_TABLE，JSON_VALUE，JSON_EXISTS和JSON_TEXTCONTAINS支持的JSON内容进行In-Memory查询
- 可以在JSON列上创建表达式（例如JSON_VALUE），并保存在“内存”列存储中
- 2到30x的性能提升

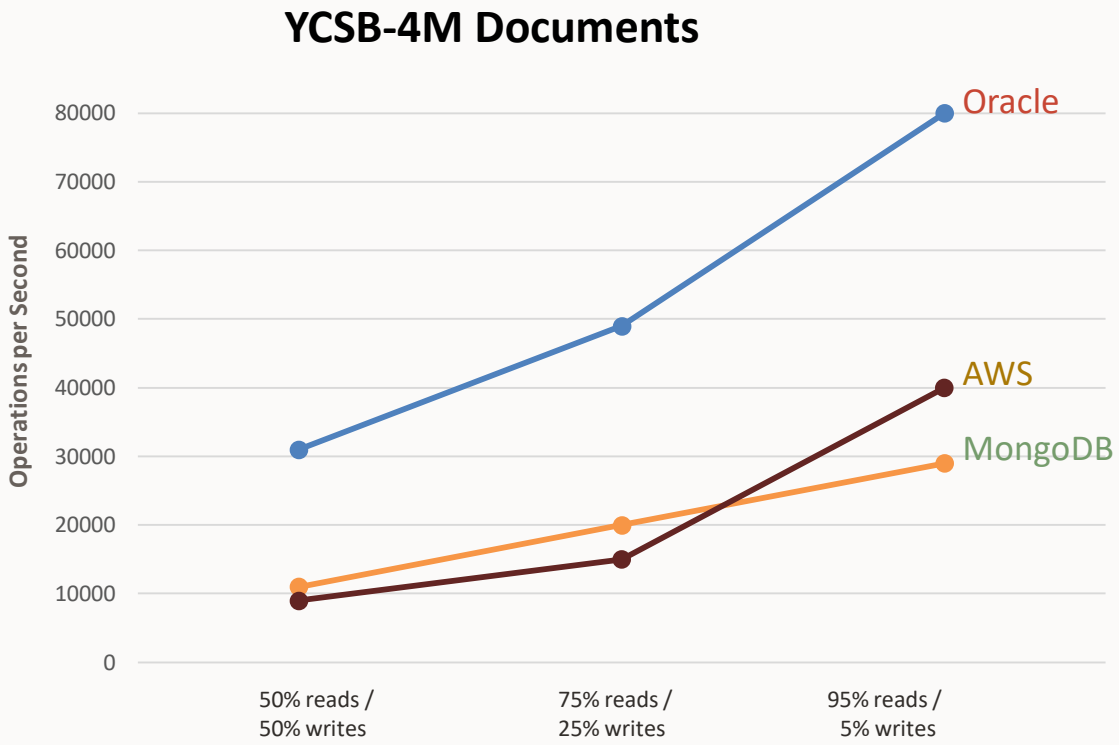
JSON数据查询Demo

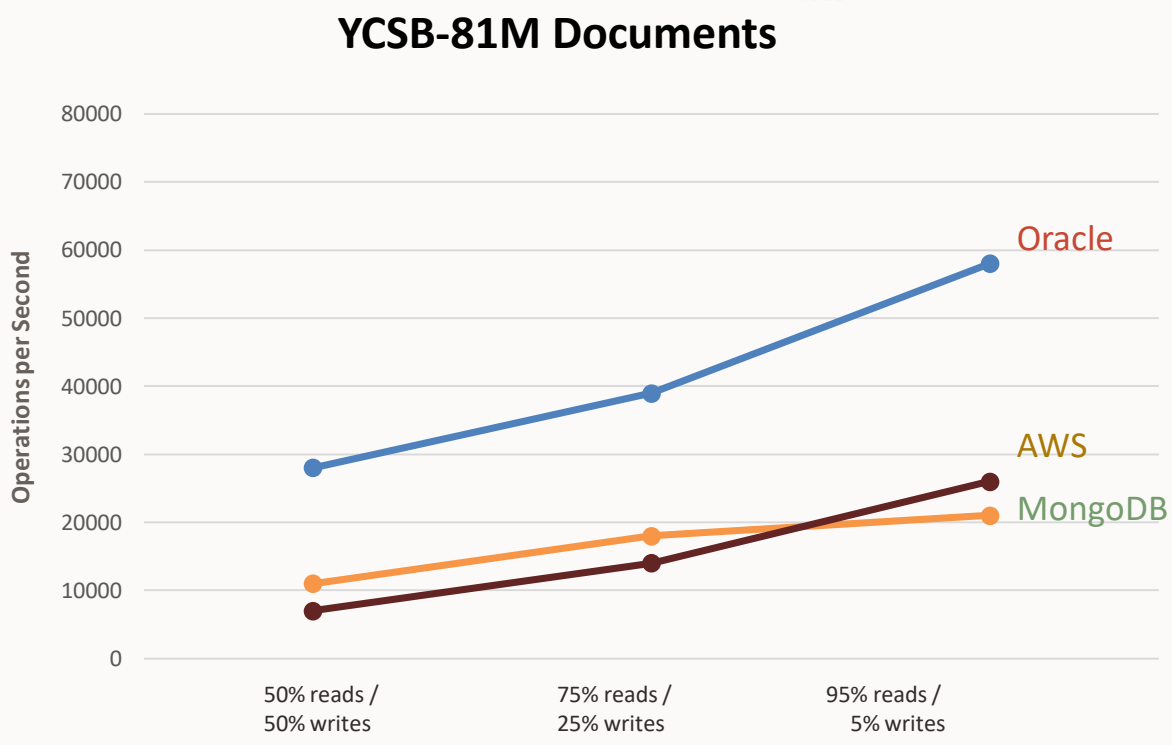
```
SQL> -- Create a table using the JSON data type.
SQL> create table t1 (
  2    id         number generated always as identity,
  3    json_data  json,
  4    constraint ta_pk primary key (id)
  5  );

Table T1 created.

SQL>
SQL> pause
```

比MongoDB & AWS DocumentDB 性能更快





Industry-standard Yahoo Cloud Serving Benchmark (**YCSB**)
Autonomous JSON Database with 8 OCPUs compared to: MongoDB Atlas on M60, AWS DocumentDB on R4.4xlarge
Source: https://www.mongodb.com/atlas-vs-amazon-documentdb/performance as of 8/12/2020

JSON多值索引

JSON Document

```
{ "title":"mybook", "author":"ROGER" }
```

创建功能索引

```
create index authindex1
   on mytable m (m.jsontext.author.string())
```

如果JSON数据中的author是数组

```
{ "title":"mybook", "author": [ "ROGER", "JOHN" ] }
```

如何创建个一个多值索引?

多值索引- 示例

```
create table mytable (jsoncol json);
insert into mytable values (
  '{ "title":"mybook", "author": [ "ROGER", "JOHN" ] }');
create multivalue index authindex2
  on mytable m(m.jsoncol.author.string());
select * from mytable m
  where json_exists(jsoncol, '$?(@.author == "JOHN")');
```

- 仅当数据类型匹配时才使用索引
- 索引可以使用string(), number(), stringOnly() 或numberOnly()
- 自动转换前两个，而不是后两个

多值索引– 执行计划输出

```
-------------------------------------------------------------------------------------------
| Id  | Operation                                | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |            |     1 |  6104 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| MYTABLE    |     1 |  6104 |     2   (0)| 00:00:01 |
|   2 |   HASH UNIQUE                            |            |     1 |  6104 |            |          |
|*  3 |    INDEX RANGE SCAN (MULTI VALUE)   | AUTHINDEX2 |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

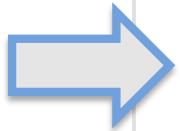
- 仅当数据类型匹配时才使用索引
- 索引可以使用string(), number(), stringOnly() 或numberOnly()
- 自动转换前两个，而不是后两个

SQL/JSON增强

SQL 中类似JSON的构造语法

使用类似于JSON文本的语法在SQL中创建JSON类型值。

```
SELECT JSON {
  'empno' : empno,
  'name' : ename,
  'salary' : sal,
  'department' : (
    SELECT JSON {
      'name' : dname,
      'loc' : loc
    }
    FROM dept d
    WHERE d.deptno = e.deptno
  )
}
FROM emp e
WHERE e.empno = 7839;
```

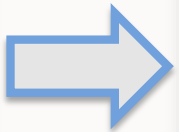


```
{
  "empno" : 7839,
  "name" : "KING",
  "salary" : 5000,
  "department" : {
    "name" : "ACCOUNTING",
    "loc" : "NEW YORK"
  }
}
```

JSON_TRANSFORM 示例

```
UPDATE employees e
SET e.jcol =  JSON_TRANSFORM (e.jcol,
              SET    '$.job' = 'Surfing Instructor',
              SET    '$.salary' = (e.jcol.salary / 3),
              REMOVE '$.phones[*]?(@.type == "work")',
              APPEND '$.children' = 'Fiona’)
WHERE e.jcol.id = 123;
```

```
{
 "id” : 123,
 ”name” : "John Smith",
 "age” : 25,
 "job” : "Programmer",
 "salary” : 60000,
 "phones” :
  [ {"type” : "mobile",
     "number” : "555-123-4567"},
      {"type” : "work",
    "number” : "555-999-1111"} ],
 "children” :["Sydney","Sierra"]
}
```



```
{
 "id” :123,
 "name" : "John Smith",
 "age " : 25,
 "job " : "Surfing Instructor",
 "salary” : 20000,
 "phones” :
  [ {"type” : "mobile",
    "number” : "555-123-4567"} ],
 "children” : ["Sydney","Sierra","Fiona"]
}
```

21c中的其他JSON改进

- SQL / JSON语法改进
  - 新的JSON路径语言项目方法支持JSON_SCALAR: float(), double(), binary(), ymInterval(), and dsInterval()
  - JSON路径语言和点符号语法支持新的聚合项方法: avg(), count(), minNumber(), maxNumber(), minString(), maxString(), sum()
- GoldenGate和XStream对原生JSON数据类型的支持
- Advance Queue对JSON数据类型的支持
- DBMS_JSON.CREATE_VIEW物化视图与增加虚拟列的支持
- JSON Object Mapping
- JSON Data Upload
- Data Pump support JSON数据类型
- JSON数据表分片

……

图的性能

In Memory Spatial

新添加In-Memory Spatial Summary列

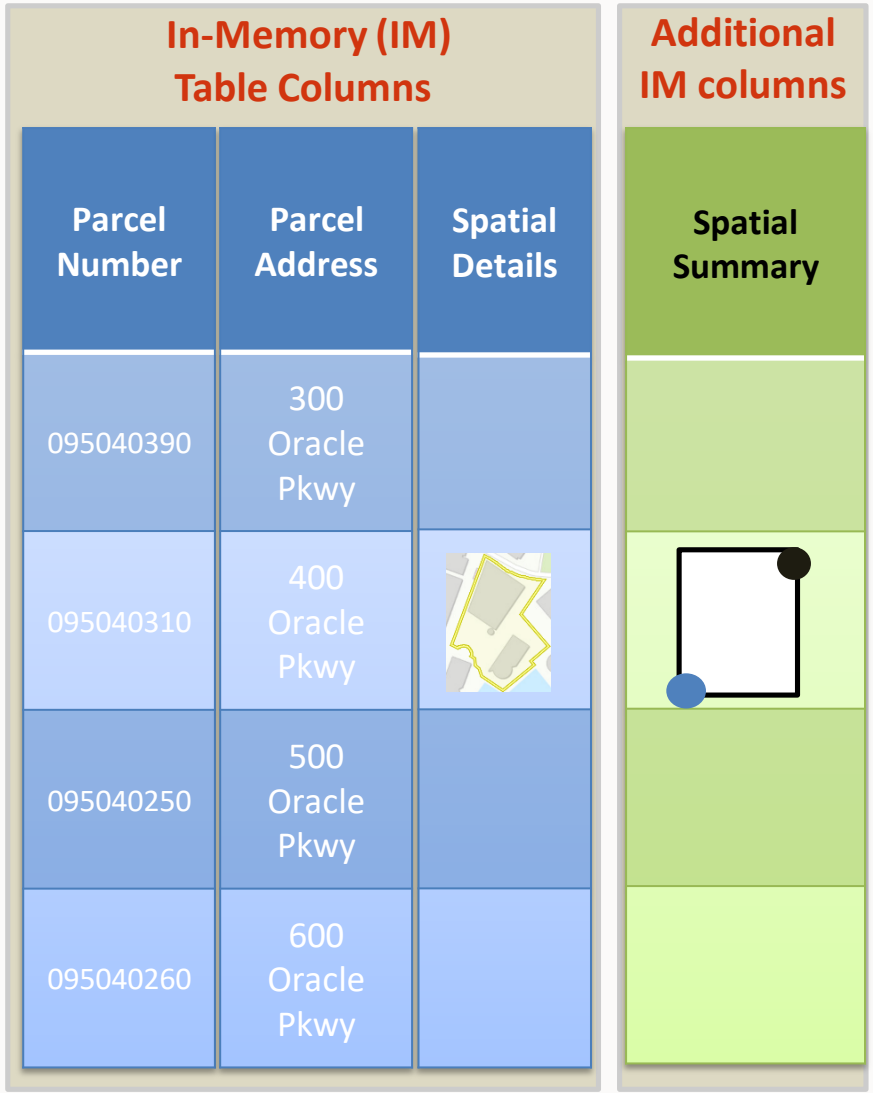
- Spatial Summary – 近似Spatial Details

以In-Memory的格式存储Spatial Summary

- 使用SIMD矢量扫描过滤值
- 替换R-Tree索引进行搜索

查询速度提高10倍

- 无需分析与维护R-Tree索引



启用In-Memory Spatial进行查询

ALTER TABLE city_points INMEMORY PRIORITY high **INMEMORY SPATIAL** (shape);

TABLE_NAME	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_DEFAULT
CITY_POINTS	SYS_IME_SDO_3ACD632341514FA8BFD1244C9A4C375C	BINARY_DOUBLE	8	SDO_GEOM_MIN_X(SYS_OP_NOEXPAND("SHAPE"))
CITY_POINTS	SYS_IME_SDO_E484EF9DF2DB4FB4BFF7AC9AF6081A95	BINARY_DOUBLE	8	SDO_GEOM_MAX_X(SYS_OP_NOEXPAND("SHAPE"))
CITY_POINTS	SYS_IME_SDO_DFF187A1C0C14F44BFF52AE162B78676	BINARY_DOUBLE	8	SDO_GEOM_MIN_Y(SYS_OP_NOEXPAND("SHAPE"))
CITY_POINTS	SYS_IME_SDO_31CF528563884F3BBFAEF9C1A68F56A5	BINARY_DOUBLE	8	SDO_GEOM_MAX_Y(SYS_OP_NOEXPAND("SHAPE"))
CITY_POINTS	SYS_IME_SDO_416B0F8BBC6A4F24BFBE1720E0E03AA3	BINARY_DOUBLE	8	SDO_GEOM_MIN_Z(SYS_OP_NOEXPAND("SHAPE"))
CITY_POINTS	SYS_IME_SDO_5F0B63CD88274F0ABFAD935DA59FA9DD	BINARY_DOUBLE	8	SDO_GEOM_MAX_Z(SYS_OP_NOEXPAND("SHAPE"))

启用In-Memory Spatial进行查询

```
SELECT city_name FROM city_points c where sdo_filter(c.shape,
  sdo_geometry(2001,8307,sdo_point_type(-122.453613,37.661791,null),null,null)) = 'TRUE';
```

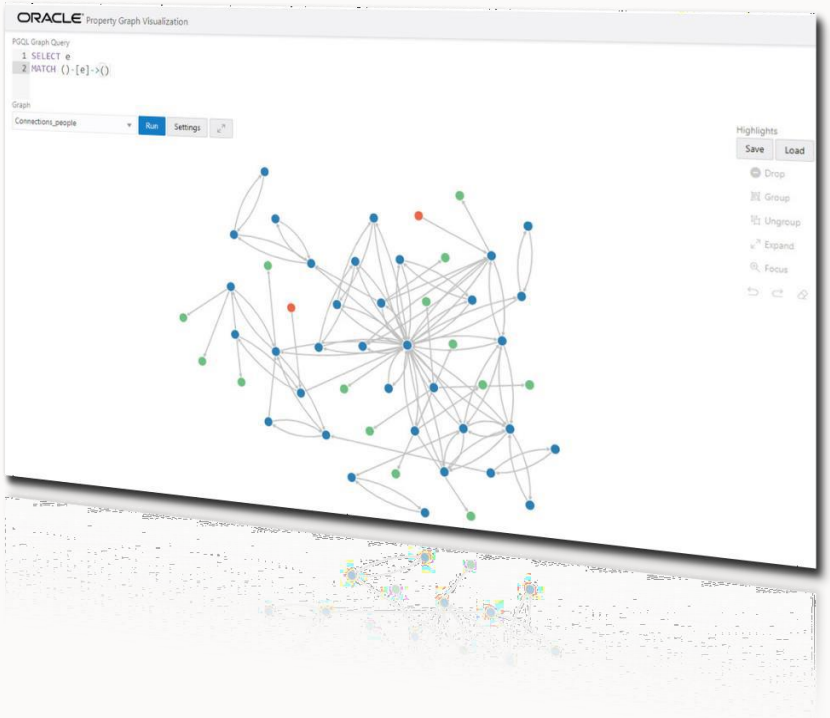
```
-------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |       |       |   334 (100)|          |
|*  1 |  TABLE ACCESS INMEMORY FULL| CITY_POINTS |     1 |  3849 |   334 (100)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter((SDO_GEOM_MAX_X("SHAPE")>=SDO_GEOM_MIN_X("MDSYS"."SDO_GEOMETRY"(2001
              ,8307,"SDO_POINT_TYPE"((-122.453613),37.661791,NULL),NULL,NULL))-7.848052667402416
              6E-008D AND SDO_GEOM_MIN_X("SHAPE")<=SDO_GEOM_MAX_X("MDSYS"."SDO_GEOMETRY"(2001,83
              07,"SDO_POINT_TYPE"((-122.453613),37.661791,NULL),NULL,NULL))+7.8480526674024166E-
              008D AND SDO_GEOM_MAX_Y("SHAPE")>=SDO_GEOM_MIN_Y. <== Rest deleted for space
```

Graph 增强

- 优化展示
  - 使用比以前所需的更少的内存来分析较大的图； 现有应用程序将运行得更快，而且不会发生任何变化。
- 用户定义的算法
  - 使用Java语法创建或扩展图算法
  - 执行速度与产品中的原生算法一样快，因为它们是使用相同的优化器编译
- 属性图可视化
- PGQL的增强



Graph 增强（Graph Studio)

自助Graph数据库和分析环境

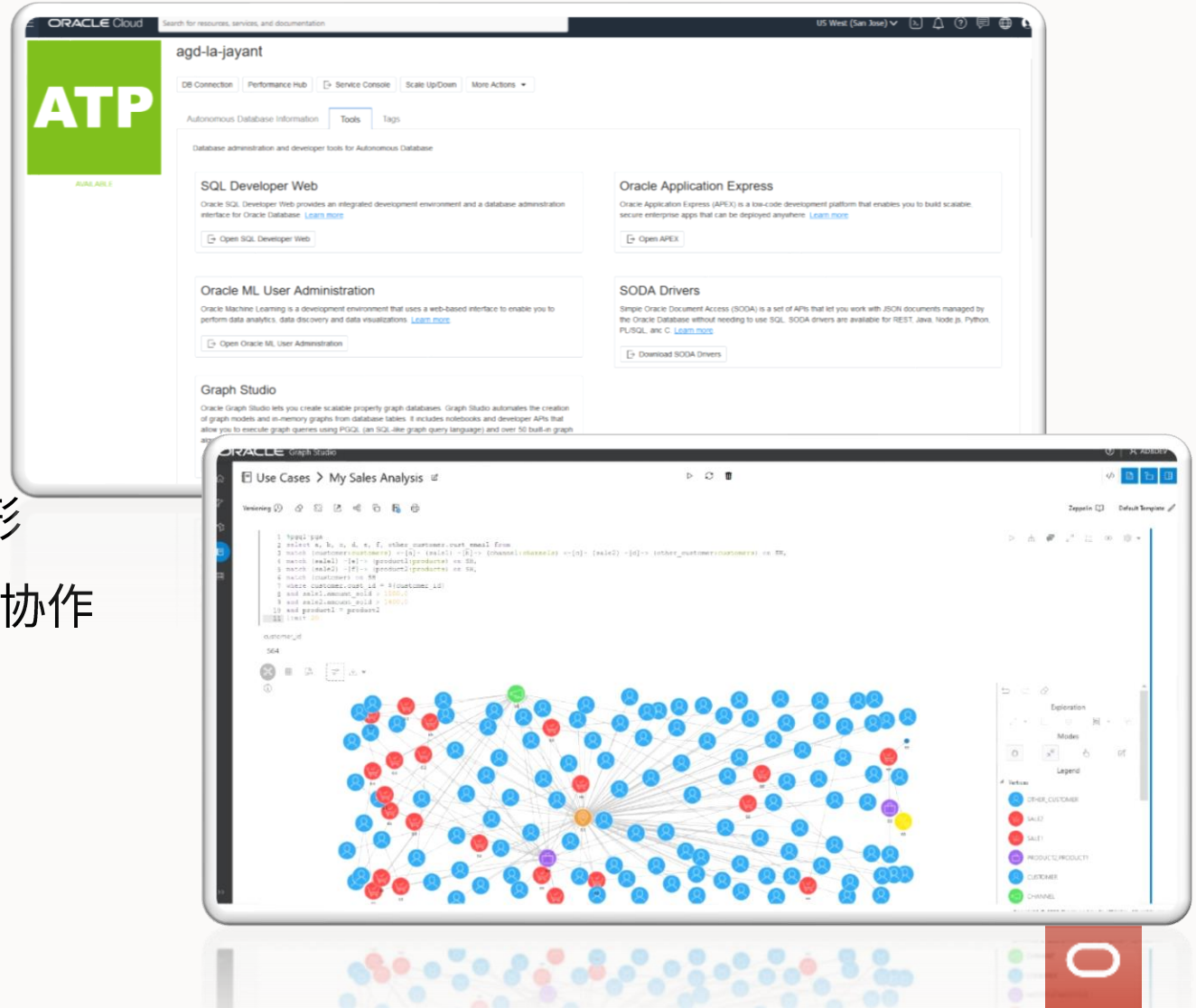
- 自治数据库的功能
- 自动驾驶，自我保护，自我修复

附带全面的工具

- 图形建模工具，用于将关系数据映射到图形
- 基于浏览器的notebook进行交互式分析和协作

集成Graph可视化

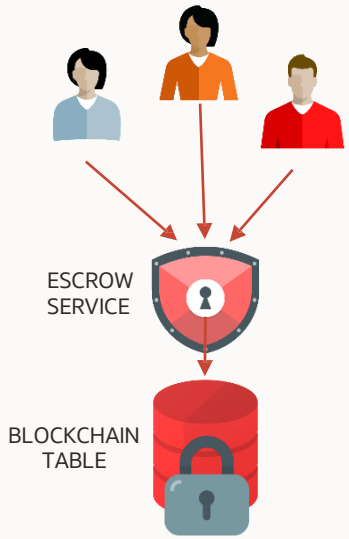
使用Graph分析引擎的成熟功能



原生态区块链表

原生的区块链表

由受信任的提供商管理的安全分类帐表，以防止欺诈



- 使用简单–允许正常SQL插入和查询的专用表
  - CREATE BLOCKCHAIN TABLE LEDGER_OF_TRADES
  - 区块链表通过创建时的Blockchain 关键字指定
  - 数据保护通过 NO DELETE 参数指定
  - 表保护通过 NO DROP 参数指定
  - 只能插入
  - 行被加密链接，参与者可以验证链接
- 区块链表中的行是防篡改的
  - 每行包含一个加密哈希值，该值基于该行中的数据和链中上一行的哈希值
  - 如果某行被篡改，则该行的哈希值会更改，这会导致链中下一行的哈希值发生更改
- 用于实施集中式区块链应用程序
  - 参与者是不同的数据库用户,信任Oracle
  - 数据库来维护交易的防篡改区块链
  - 所有参与者都必须具有将数据插入到区
  - 块链表中的特权
  - 区块链的内容由应用定义和管理
  - 集中式区块链在更高吞吐量和更低交易延迟的场景中很有用

区块链表

TRADE LEDGER

ID	User	Value
1	Tom	500
2	Carol	176
3	Stev	00
4	n	17
5	Mike	332
6	Sarah	632
7	Eve	25
8	Prisha	850

ADSJS
%SHS
SH@1
DHD3
*EGG
AH11
LIO$
SHS4

BLOCKCHAIN TABLE

- 多模：Oracle 数据库秉持多模化发展的理念，引入原生的区块链表支持
- 原生：通过 CREATE Blockchain Table 创建
- 简化：Blockchain 表只能INSERT插入记录

  行被加密链接

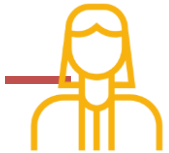
  链是参与者可验证的
- 融合：Blockchain 表可以和其他表关联进行事务或查询及数据分析

  与分布式区块链相比容易得多

使企业应用程序可以轻松地使用区块链来识别和防止欺诈

AWS需要单独的QLDB服务来支持分类帐表 – 一个专业数据库

区块链表提供多层欺诈保护



用户欺诈保护

区块链表可防止任何人对SQL进行更改-无论凭据如何

区块链所有者对新数据进行加密签名可实现不可否认性

管理员欺诈保护

加密区块链哈希检测数据的任何更改

区块链哈希的独立验证和分发可防止表重写

身份欺诈保护

用户对新数据进行加密签名可防止假冒

自动机器学习

全新 AutoML

自动建立和比较机器学习模型，为数据科学家和开发人员提供更快，更轻松的机器学习

- 自动模型选择
  - 识别每个工作负载的最佳预测算法
  - 比穷举搜索更快地找到最佳模型
- 自动特征选择
  - 通过确定最具预测性的特征来减少特征数量
  - 识别最佳预测结果的数据
  - 提高性能和准确性
- 自动优化超参数
  - 大大提高模型准确性
  - 避免使用手动或详尽的搜索技术

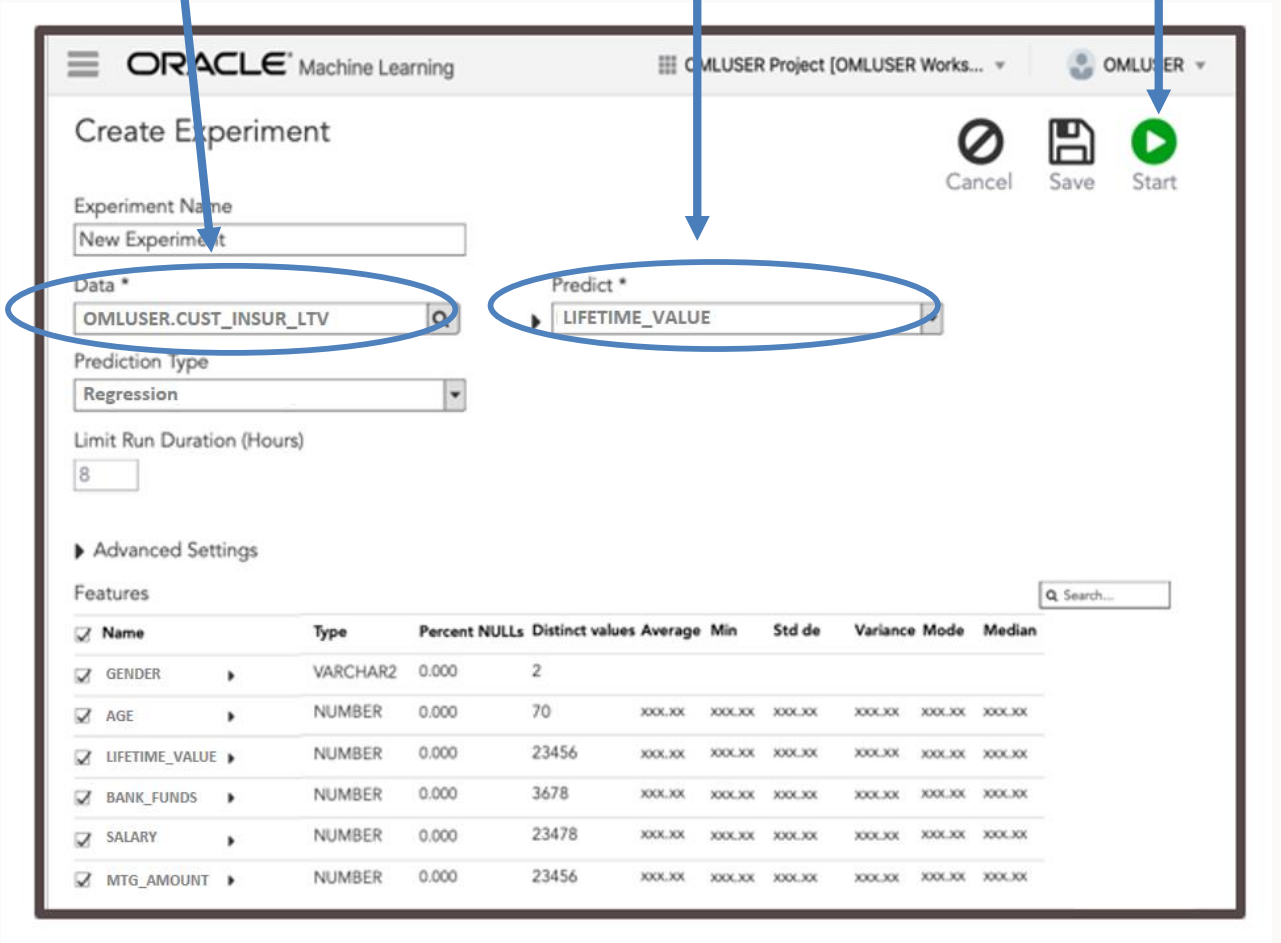
使非专业用户可以利用机器学习

自动机器学习用户界面

1.选择数据集

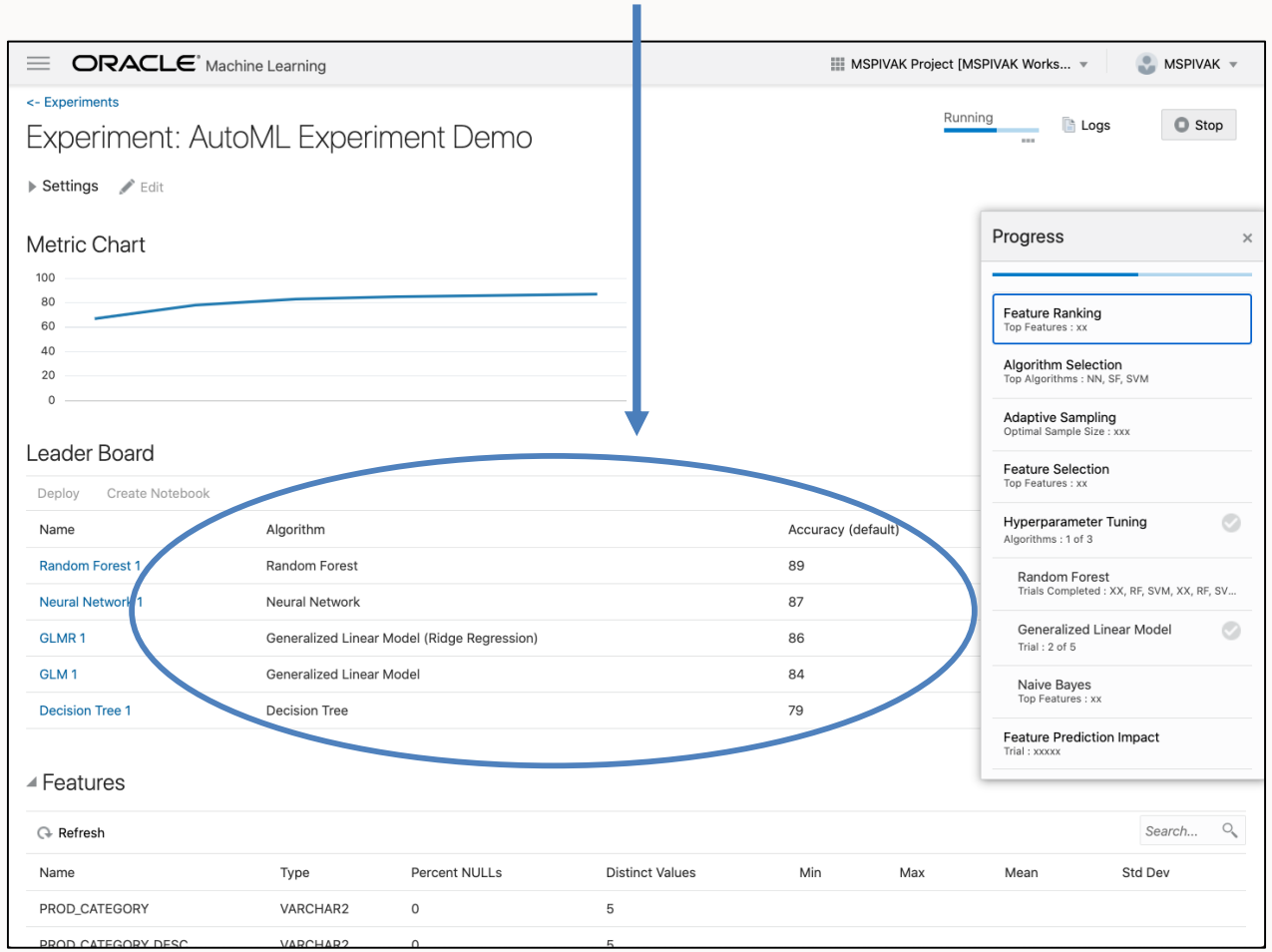
2.选择要预测的数据

3.按开始



自动机器学习用户界面

多个预测算法比较最佳选择



其他机器学习改进

- MSET-SPRT 算法的支持
  - 多元状态估计技术-顺序概率比检验（MSET-SPRT）算法是一种用于监视关键过程的非线性非参数异常检测技术
  - 该算法检测细微的异常，同时产生最少的错误警报
- XGBoost 算法的支持
  - 该算法是用于回归和分类的高效可扩展梯度树增强机器学习算法
  - 为神经网络算法增加了Adam优化求解器
  - Adam是神经网络算法的优化求解器，该算法计算效率高，需要的内存很少，非常适合处理数据或参数较大或两者兼而有之的问题。

SQL宏

SQL 宏支持

- 允许开发人员通过宏定义缩略复杂的处理逻辑
- 支持 SCALAR 和 TABLE 两种SQL宏类型:
  - **SCALAR** 表达式使用在SELECT list, WHERE/HAVING, GROUP BY/ORDER BY clauses
  - **TABLE** 表达式使用在 FROM-clause
- Scalar Macros提供了一种封装复杂SQL表达式的简单方法
- SQL Table Macro封装了FROM子句中使用的SQL

```
create function clip(lo number, x number, hi number)
return varchar2 SQL_MACRO(SCALAR)
is begin
    return 'least(greatest(x, lo), hi)';
```

```
SELECT ename, CLIP (:lower, sal, :upper)
FROM emp;
```

标量的SQL Macros

标量宏提供了一种封装复杂SQL表达式的简单方法

- 充当SQL的预处理器
- 对SQL Optimizer透明的可重用代码
- 轻松创建可重用和可移植的代码
- 自由的上下文切换

```
create or replace function
       sales_tax(unit_cost number,
                 unit_type varchar)
return varchar2 SQL_MACRO(SCALAR) is
begin
return q'[case when unit_type = 'FOOD'
         then unit_cost
         else unit_cost * 1.2 end]';
end;
```

FUNCTION RETURNS STRING

```
SQL> select sales_tax(20,'WINE') from dual;

SALES_TAX(20,'WINE')
--------------------
                  24
```

EQUIVALENT TO

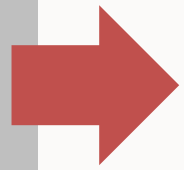
```
SQL> select case
            when 'WINE' = 'FOOD' then 20
            else 20*1.2
       end
     from dual;
```

;

表的SQL Macros

SQL表宏封装了FROM子句中使用的SQL

```
CREATE OR REPLACE FUNCTION budget
return varchar2 SQL_MACRO
IS
BEGIN
RETURN q'( select department_id,
sum(salary) budget
from hr.employees
group by department_id )';
END;
```



```
SQL> SELECT * FROM budget() WHERE
department_id IN (10,50);
```

```
DEPARTMENT_ID BUDGET
------------- ----------
50            156400
10            4400
```

总结

21c

1、原生的JavaScript：支持在数据库内部执行Javascript ，无论PL/SQL还是APEX都可以直接调用

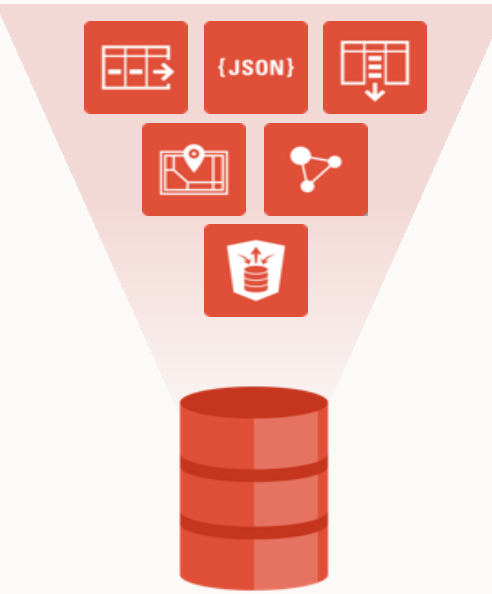
2、JSON加速与灵活性：支持二进制格式的JSON；支持In-Memory；更新密集型或扫描密集型操作速度数倍增长

3、图的性能改进：支持In-Memory Spatial，查询更快；优化property graph可视化，可自定义算法

4、原生的区块链支持 - Native Blockchain Tables，原生区块链表类似于标准表，允许SQL插入且插入的行以加密方式链接。用户可以选择对行数据进行签名，来杜绝身份欺诈。轻松集成到应用中参与其它表的事务和查询。

5、自动机器学习 - AutoML: 自动化的机器学习 - 非专家级用户也可以充分利用机器学习的优势。AutoML将推荐最合适的算法，自动选择特性并调优超参数，从而大幅提升模型准确性；以前是收费选件，现在免费。

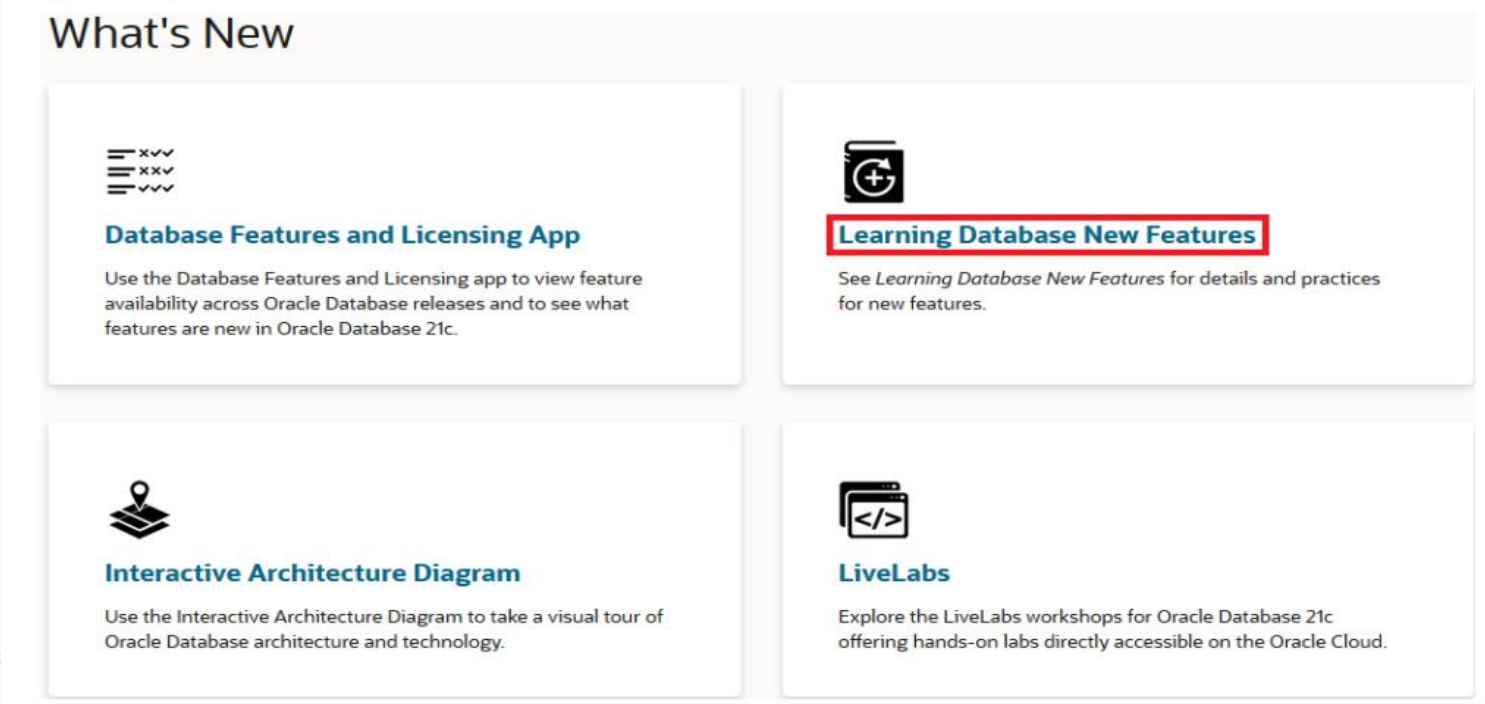
6、SQL宏的支持增加：可以允许开发者定义SQL宏缩略复杂逻辑



学习资源

https://docs.oracle.com/en/database/oracle/oracle-database/21/whats-new.html

从 21c 开始， Oracle 针对 新特性 的学习提供了一套完整的理论+练习（Learning Database New Features）的方式呈现在我们面前。



Thank you