New Features in Oracle Database 23C

Cathye Pendley – Business Intelligence Program Manager, Rosendin

Gerald Venzl – Senior Director of Product Management, Oracle

Helpful Links –

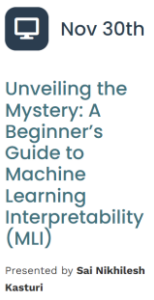
ORACLE ANALYTICS VIDEOS:
https://www.youtube.com/@OracleAnalytics/videos

OAC MAY 2023 NEW FEATURES VIDEOS BY ORACLE:
https://www.youtube.com/watch?v=cgnJeVu-plE&list=PL6gBNP-Fr8KWZkXpZnjr7lTMfDTj9-dfK&pp=iAQB

ORACLE ANALYTICS COMMUNITY:
https://community.oracle.com/products/oracleanalytics

ORACLE ANALYTICS LIVE DEMOS:
https://www.oracle.com/business-analytics/data-visualization/demos/

Future & Past TechCasts:



TechCast Archive

Click to see Live TechCast page

2023 | 2022 | 2021 | 2020 | 2019

Date	Title	Presenter(s)	Replay	Download(s)
May 4	Oracle APEX: A Swiss Army Knife Story for Your Analytics	Lucas Hirschegger & Simon Collins	Video	Slides
Apr 20	From Data to Insights with Oracle Analytics	Joel Acha	Video	Slides
Apr 6	Data Platform Migrations – Few Learnings	Sujata Balupala & Sanjay Sabnis	Video	Slides
Mar 14-16	AnDOUC Summit 2023	AnDOUC	– –	– –
Feb 16	Favorite New Features in Jan 2023 OAC	– –	– –	– –
Jan 26	Summit Preview of Presentations	Various Presenters	Video	Slides
Jan 12	Oracle Analytics & Spatial Studio	Wayne Van Sluys and David Lapp	Video	Slides

Submit a topic to share at **https://andouc.org/techcasts/**

Analytics and Data ORACLE USER COMMUNITY

www.andouc.org

Oracle Spatial & Graph SIG

Analytics and Data
ORACLE USER COMMUNITY

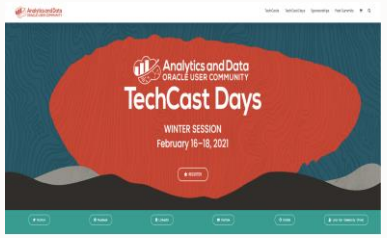
We Have Merch!

Show your “Tech Side” in everything you do!

Visit the AnDOUC Store at ANDOUC.ORG

Analytics and Data
ORACLE USER COMMUNITY

Let's Connect



Website
http://andouc.org/

slack

Chat with the Experts
https://bit.ly/Join-ANDOUC-Slack

YouTube

Watch Previous TechCasts
https://bit.ly/3qmGgHN

@AnalyticAndData

https://www.facebook.com/AnDOracleUserCommunity

https://www.linkedin.com/company/analytics-and-data-oracle-user-community

Spatial + Graph SIG
bit.ly/Spatial-Graph-LinkedIn

Analytics and Data
ORACLE USER COMMUNITY

Save the Date!

Analytics and Data Summit 2024

April 9-11, 2024
Oracle Conference Center
Redwood Shores, California

www.andouc.org/andsummit2024

Agenda

NEW IN 23c

- Oracle Database 23c New Features
  - Marque
  - Analytic
  - Nice to Know
  - Time Saver
- Oracle database 23c – Free for developers

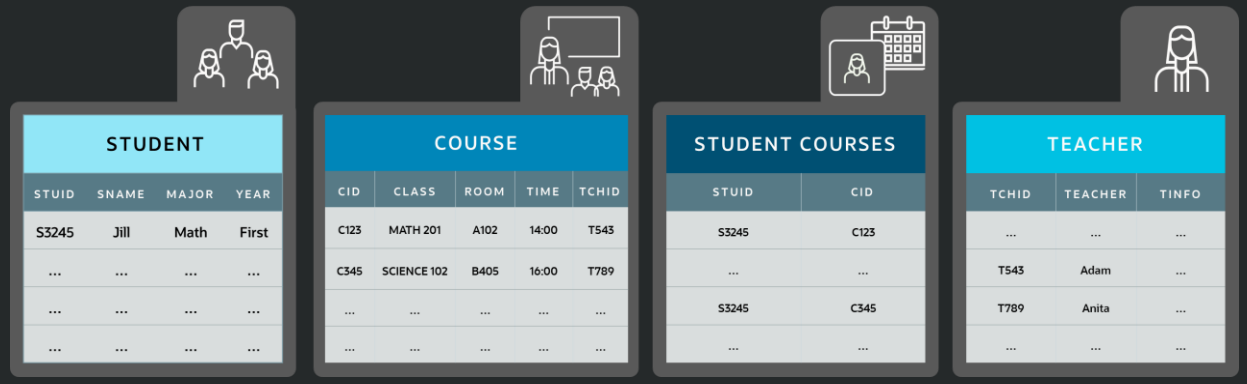
Marque Features

NEW IN 23c

- JSON Relational Duality
- JSON Schema
- Domains
- Operational Property Graphs
- SQL/PGQ

ROSENDIN

JSON Relational Duality Views new in Oracle Database 23c enable databases to generate JSON format and APIs from relational tables



SCHEDULE FOR: JILL

```
{
 "student"     : "S3245",
 "name"        : "Jill",
 "major"       : "Math",
 "schedule"    :
   [  {
        "time"    : "14:00",
        "course"  : "Math 201",
        "room"    : "A102",
        "teacher" : "Adam"
      },
      {
        "time"    : "16:00",
        "course"  : "Science 102",
        "room"    : "B405",
        "teacher" : "Anita"
      }
   ]
}
```

The structure of the Duality view mirrors the structure of the desired JSON, making it easy to define

```
CREATE JSON DUALITY VIEW student_schedule
AS student
{{
   student    : stuid
   name       : sname
   major      : major
   schedule   : student_courses
   [ {
    course
      {
         time      : time
         course    : cname
         courseId  : cid
         room      : room
         teacher @unnest
         {
           teacher : tname
         }
      }
    } ]
};
```

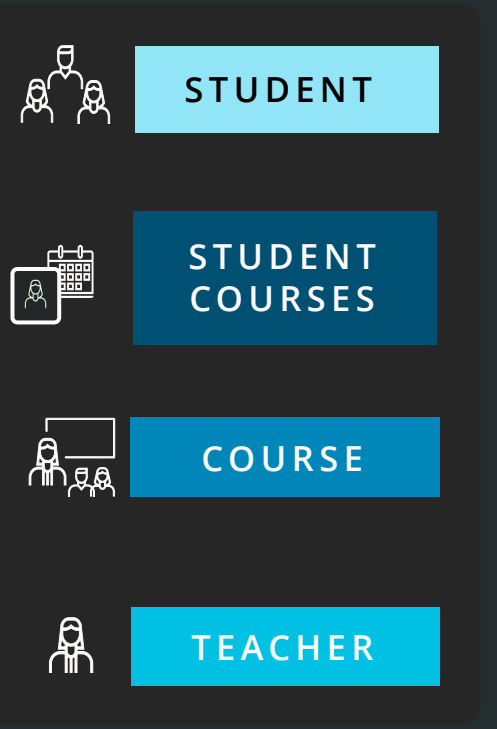
Uses familiar GraphQL syntax

STUDENT SCHEDULE FOR: JILL

```
{
 "student"     : "S3245",
 "name"        : "Jill",
 "major"       : "Math",
 "schedule"    :
   [  {
        "time"    : "14:00",
        "course"  : "Math 201",
        "room"    : "A102",
        "teacher" : "Adam"
      },
     …
   ]
}
```

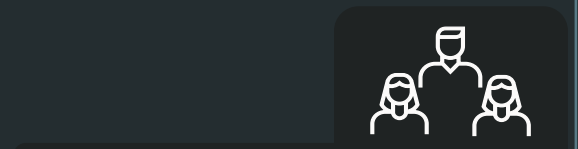
The view simply specifies the tables that contain the data to include in the JSON document

```
CREATE JSON DUALITY VIEW student_schedule
AS student
{
   student     : stuid
   name        : sname
   major       : major
   schedule    : student_courses
   [ {
    course
      {
         time       : time
         course     : cname
         courseId   : cid
         room       : room
         teacher @unnest
         {
           teacher : tname
         }
      }
    } ]
};
```



And specifies the table columns that hold the values

```
CREATE JSON DUALITY VIEW student_schedule
AS student
{
   student    : stuid
   name       : sname
   major      : major
   schedule   : student_courses
   [ {
    course
      {
         time      : time
         course    : cname
         courseId  : cid
         room      : room
         teacher @unnest
         {
           teacher : tname
         }
      }
    } ]
};
```



STUDENT

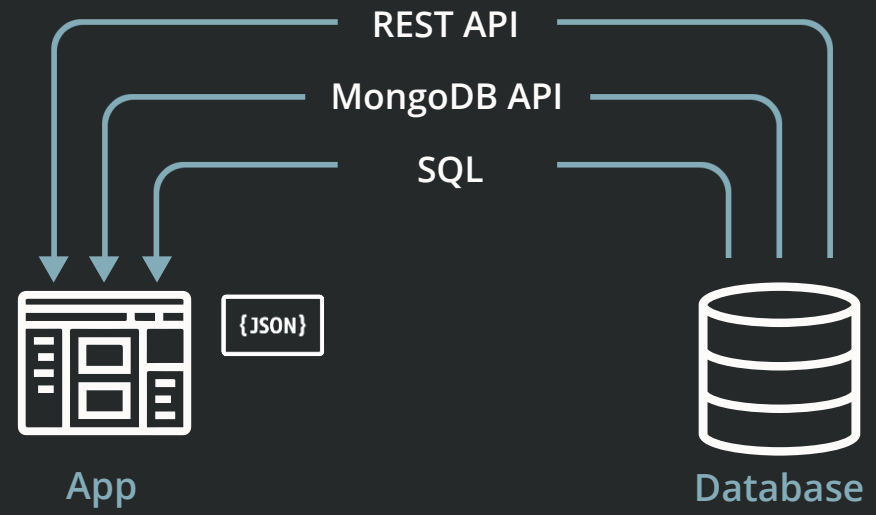
STUID	SNAME	MAJOR	YEAR
S3245	Jill	Math	First
...	...	...	...
...	...	...	...
...	...	...	...

JSON Duality Views are simple to query using document APIs

Apps use standard REST APIs to GET a document from the View

```
GET school.edu/student_sched?q={"student":{"$eq":"Jill"}}
```

Views can also be accessed by any app using the MongoDB compatible API and SQL



JSON Duality Views are also simple to update

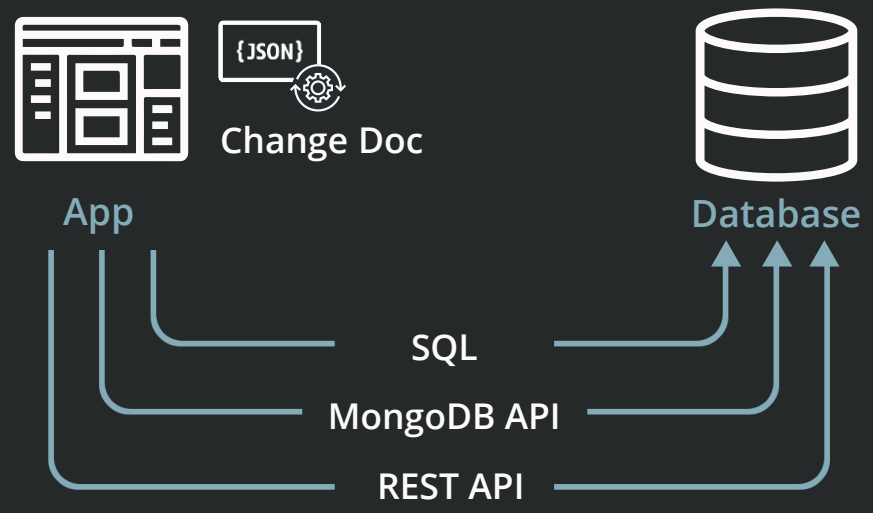
Apps edit the document they previously got

Then PUT the document back into the View

- Or write it with the MongoDB API or SQL

```
PUT school.edu/student_schedule/:stuid
```

As part of the update, the database detects the changes made to the document and only modifies the underlying table rows that have changed



Duality allows JSON documents to include any data that is convenient for the app

Duality views never duplicate data because the data is stored as normalized rows

Huge benefit for other apps using the same data!

STUDENT SCHEDULE FOR: JILL

```
{
 "student"     : "S3245",
 "name"        : "Jill",
 "major"       : "Math",
 "schedule"    :
   [  {
        "time"    : "14:00",
        "course"  : "Math 201",
        "room"    : "A102",
        "teacher" : "Adam"
      },
      {
        "time"    : "16:00",
        "course"  : "Science 102",
        "room"    : "B405",
        "teacher" : "Anita"
      }
   ]
}
```

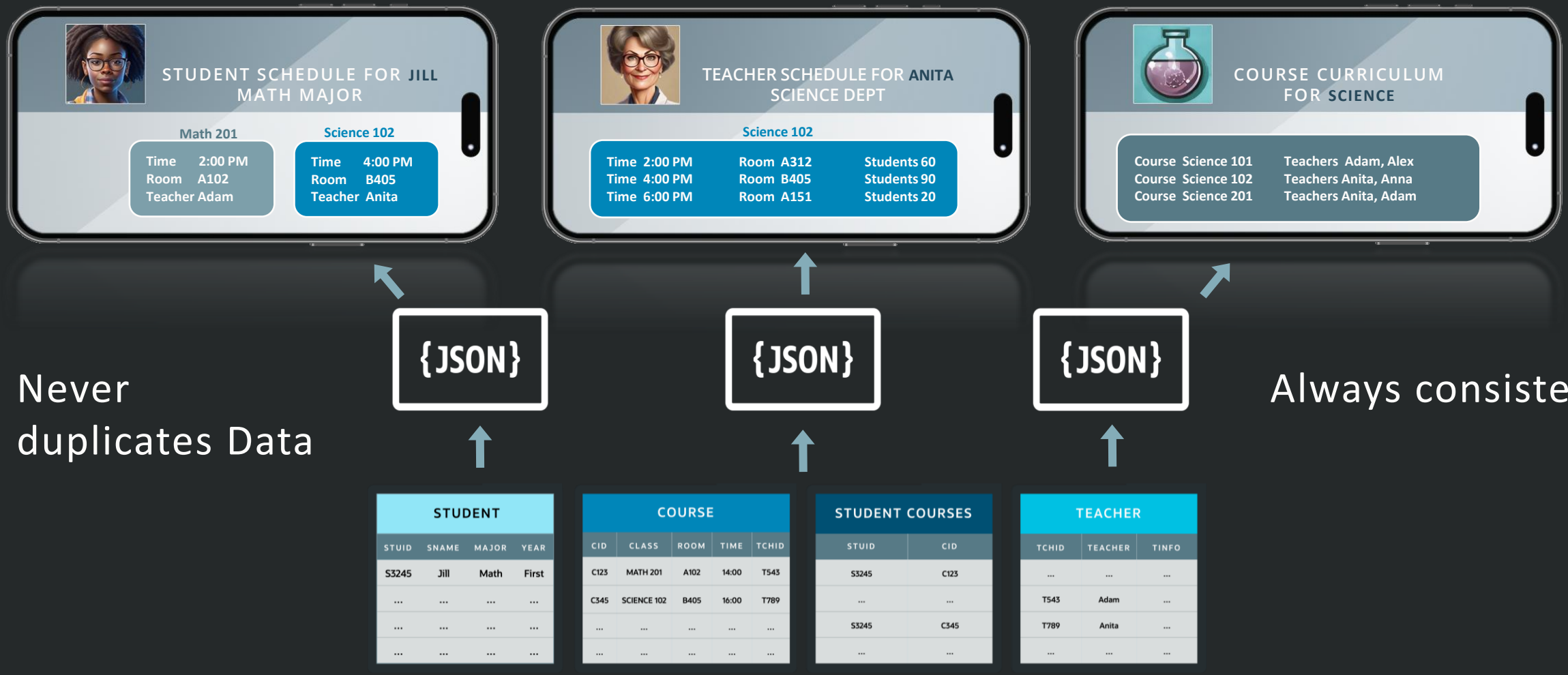
Seems duplicated but is not

STUDENT SCHEDULE FOR: LUCAS

```
{
 "student"     : "S4356",
 "name"        : "Lucas",
 "major"       : "Engineering",
 "schedule"    :
   [  {
        "time"    : "14:00",
        "course"  : "Math 201",
        "room"    : "A102",
        "teacher" : "Adam"
      },
      {
        "time"    : "18:00",
        "course"  : "Physics",
        "room"    : "A115",
        "teacher" : "Alex"
      }
   ]
}
```

Because **any** data can be included in documents,
Duality provides better JSON to Apps than JSON Databases

JSON Duality views allow the same underlying data to be customized to match the needs of each app use case



Never duplicates Data

Always consistent

Huge benefits for App Dev!

NEW IN 23c

JSON Schema

Validate JSON documents

- Validation on storage
- Validation on query
- Validation reports

```
CREATE TABLE jdocs (
 doc JSON VALIDATE
 '{
   "type": "object",
   "properties": {
    "id":
     {"type": "number"}
   }
  }'
);
```

```
SELECT * FROM staging
 WHERE doc IS JSON
VALIDATE
 '{
   "type": "object",
   "properties": {
     "id":
      {"type": "number"}
    }
  }';
```

```
SELECT
 DBMS_JSON_SCHEMA
 .VALIDATE_REPORT(doc, schema)
FROM jdocs;

REPORT
----------------------------
{
  "valid" : false,
  "errors" :
  [
    {
      "schemaPath" : "$.id",
      "instancePath" : "$",
      "code" : "JZN-00503",
      "error" : "invalid type
found, actual: string,
expected: number"
    }
  ]
}
```


ROSENDIN

Domains

Abstract domain specific knowledge into reusable objects

NEW IN 23c

```
CREATE DOMAIN DomainName AS <Data Type>
[ DEFAULT <expression> [ ON NULL ] ] [ NOT NULL ]
[ CONSTRAINT [ name ] CHECK (<expression>) [ ENABLE | DISABLE ] ]
[ COLLATE collation ]
[ DISPLAY <expression> ]
[ ORDER <expression> ]
[ ANNOTATIONS ( annotations ) ]
```

```
CREATE DOMAIN email AS VARCHAR2(255) NOT NULL
CONSTRAINT email_c CHECK
   (REGEXP_LIKE (email, '^(\S+)\@(\S+)\.(\S+)$'))
DISPLAY '---' || SUBSTR(email, INSTR(email, '@'))
ORDER SUBSTR(email, INSTR(email, '@')+1) ||
      SUBSTR(email, 1, INSTR(email, '@'));
```


ROSENDIN

NEW IN
23c

Domains

Abstract domain specific knowledge into reusable objects

```
CREATE TABLE customers (
  cust_id        NUMBER          NOT NULL PRIMARY KEY,
  name           VARCHAR2(4000)  NOT NULL,
  contact_email  VARCHAR2(1000)  DOMAIN email,
  invoice_email  email
);
```

```
INSERT INTO customers values (1, 'TEST', 'abc', 'abc');

ORA-02290: check constraint (EMAIL_C) violated
```


ROSENDIN

NEW IN
23c

Domains

Abstract domain specific knowledge into reusable objects

- New functions DOMAIN_DISPLAY() and DOMAIN_ORDER() to retrieve display and order

```
SELECT DOMAIN_DISPLAY(invoice_email)
 AS email
  FROM customers;

EMAIL
-------------
---@aldi.com
---@swarovski.com
---@shell.com
```

```
SELECT name FROM customers
ORDER BY DOMAIN_ORDER(contact_email);


NAME
-------------
Aldi
Shell
Swarovski
```

Oracle is Constantly Innovating

In Oracle Database 23c

JSON duality views, Graph, AI, vector search, and much, much more

On YouTube

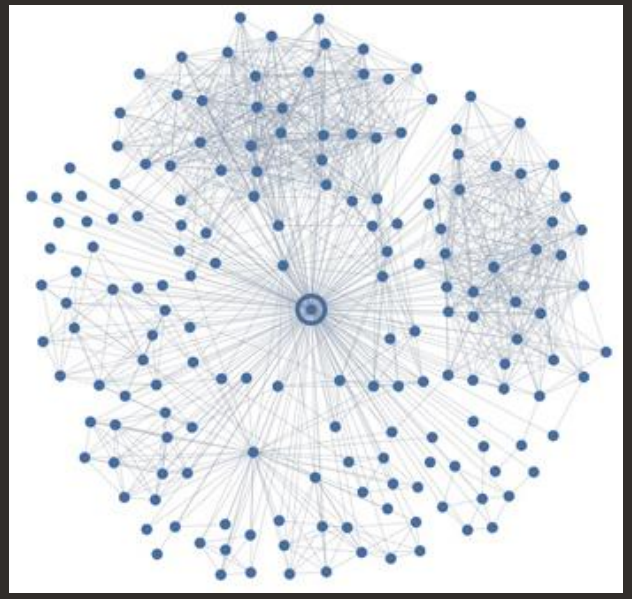
A must-watch: Juan Loaiza's keynote

"With Oracle Database 23c one part of an app can treat data as relational, while other parts treat the same data as a document, and others treat it as a graph."

Operational Property Graphs

Your data is connected

Leverage additional insights in your data by analyzing connections



Comparing SQL with and without SQL/PGQ

12 joins and 2 UNION ALLs

NEW IN 23c

```
-- transfers indirectly from <src> to <dst>

SELECT account_id
   FROM GRAPH_TABLE(bank_graph
   MATCH (src)-[is bank_transfers]     3}(dst)
   COLUMNS  src.id as account_id) );
```

```
-- transfers indirectly from <src> to <dst>

SELECT
FROM   bank_accounts  v1 ,
       bank_transfers  btx,
       bank_accounts
WHERE  (v1.id = btx.src_acct_id  AND v2.id = btx.dst_acct_id)
AND     v1.id= <src> AND  2.id= <dst>
UNION ALL
SELECT
FROM   bank_accounts  v1 ,
       bank_transfers  btx,
       bank_accounts  bc2,
       bank_transfers  btx2 ,
       bank_accounts
WHERE  (v1.id = btx.src_acct_id  AND  bc2.id = btx.dst_acct_id  AND
                                    AND
AND     v1.id= <src> AND v2.id= <dst>
UNION ALL
SELECT
FROM   bank_accounts  v1 ,
       bank_transfers  btx,
       bank_accounts  bc2,
       bank_transfers  btx2 ,
       bank_accounts  bac4,
       bank_transfers  btx5 ,
       bank_accounts
WHERE   v1.id = btx.src_acct_id  AND bc2.id = btx.dst_acct_id  AND
        bc2.id = btx2.src_acct_id  AND bac4.id = btx2.dst_acct_id  AND
        bac4.id = btx5.src_acct_id  AND  v2.id = btx5.dst_acct_id
AND     v1.id= <src> AND v2.id= <dst>
;
```


ROSENDIN

Analytics Functions

NEW IN 23c

- Aggregation over Interval Data Types
- String Matching Functions

ROSENDIN

Aggregation over Interval Data Types

NEW IN 23c

```
create table t1 (
  id          number,
  start_time  timestamp,
  end_time    timestamp,
  duration    interval day to second generated always as (end_time - start_time) virtual
);
```

```
select id,
       start_time,
       end_time,
       duration,
       avg(duration) over () as avg_duration
from t1;

        ID START_TIME           END_TIME             DURATION             AVG_DURATION
---------- -------------------- -------------------- -------------------- ------------------------------
         1 2023-04-10 08:45:00  2023-04-10 18:01:00  +00 09:16:00.000000  +000000000 09:00:15.000000000
         2 2023-04-11 09:00:00  2023-04-11 17:00:00  +00 08:00:00.000000  +000000000 09:00:15.000000000
         3 2023-04-12 08:00:00  2023-04-12 17:45:00  +00 09:45:00.000000  +000000000 09:00:15.000000000
         4 2023-04-13 07:00:00  2023-04-13 16:00:00  +00 09:00:00.000000  +000000000 09:00:15.000000000
```

String Matching SQL Functions

NEW IN 23c

1. PHONIC_ENCODE
    - Converts words or phrases into codes based on their pronunciation.
    - Algorithms:
        - Double Metaphone (DM)
        - Double Metaphone Alternative: uses alternative codes to accommodate some ambiguous cases

2. FUZZY_MATCH
    - Gives a gauge of how *textually* similar two strings are.
    - Algorithms:
        - Levenshtein: corresponds to UTL_MATCH.EDIT_SIMILARITY/EDIT_DISTANCE
        - JARO_WINKLER: corresponds to UTL_MATCH.JARO_WINKLER/JARO_WINKLER_SIMILARITY
        - BIGRAM
        - TRIGRAM
        - WHOLE_WORD_MATCH
        - LONGEST_COMMON_SUBSTRING

String Matching SQL Functions – PHONIC_ENCODE

NEW IN 23c

```
SELECT TEXT_VALUES,
phonic_encode(DOUBLE_METAPHONE, TEXT_VALUES, 12) AS DM12,
phonic_encode(DOUBLE_METAPHONE_ALT, TEXT_VALUES, 12) AS DMA12,
phonic_encode(DOUBLE_METAPHONE, TEXT_VALUES, 3) AS DM3,
phonic_encode(DOUBLE_METAPHONE_ALT, TEXT_VALUES, 3) AS DMA3
FROM PHONIC_TEST ;
```

TEXT_VALUES	DM12	DMA12	DM3	DMA3
Knight	NT	NT	NT	NT
Night	NT	NT	NT	NT
Peter Pan	PTRPN	PTRPN	PTR	PTR
The Hulk	0LK	TLK	0LK	TLK
Barbie	PRP	PRP	PRP	PRP

NEW IN
23c

String Matching SQL Functions FUZZY_MATCH

1. LEVENSHTEIN corresponds to UTL_MATCH.EDIT_DISTANCE or UTL_MATCH.EDIT_SIMILARITY and gives a measure of character edit distance or similarity.

2. JARO_WINKLER corresponds to UTL_MATCH.JARO_WINKLER (a percentage between 0-1) or UTL_MATCH.JARO_WINKLER_SIMILARITY (the same but scaled from 0-100).

3. BIGRAM and TRIGRAM are instances of the N-gram matching technique, which counts the number of common contiguous sub-strings (grams) between the two strings.

4. WHOLE_WORD_MATCH corresponds to Word Match Percentage or Count comparison in Oracle Enterprise Data Quality. It calculates the LEVENSHTEIN or edit distance of two phrases with words (instead of letters) as matching units.

5. LONGEST_COMMON_SUBSTRING finds the longest common substring between the two strings.

String Matching SQL Functions FUZZY_MATCH

NEW IN 23c

```
SELECT
   text1, text2,
   fuzzy_match(LEVENSHTEIN,  text1, text2)             AS LEV,
   fuzzy_match(LEVENSHTEIN,  text1, text2, UNSCALED)   AS ULEV,
   fuzzy_match(JARO_WINKLER, text1, text2)             AS JW,
   fuzzy_match(BIGRAM,       text1, text2)             AS BIG,
   fuzzy_match(BIGRAM,       text1, text2)             AS UBIG,
   fuzzy_match(TRIGRAM,      text1, text2)             AS TRIG,
   fuzzy_match(LONGEST_COMMON_SUBSTRING, text1, text2) AS LCS
FROM (
  VALUES ('kitten', 'sitten'),
         ('Apco Oil Lube 170’,       'Apco Oil Lube 347'),
         ('Apco Oii  2  l Lube 170', 'Apco Oil Lube 347')
) t (text1, text2);
```

TEXT1	TEXT2	LEV	ULEV	JW	BIG	UBIG	TRIG	LCS
kitten	sitten	84	1	88	80	80	75	83
Apco Oil Lube 170	Apco Oil Lube 347	83	3	95	81	81	80	82
Apco Oiil Lube 170	Apco Oil Lube 347	78	4	94	76	76	68	44

Nice to Know

NEW IN
23c

- Schema Level Privileges
- Table Value Constructor
- Annotations
- Boolean Data Type
- Select without From
- 4096 Columns
- Developer Role
- Better Return Clause

ROSENDIN

NEW IN
23c

SCHEMA level privileges

Ability to grant privileges for objects in an entire schema

- Prior had to grant access for ANY object in the entire DB or for every object explicitly
- Now can grant access to ANY object in the **entire schema** instead

```
GRANT SELECT ANY TABLE
 TO HR;

GRANT SELECT ON
  PROD.CUSTOMERS,
  PROD.SALES,
  PROD.ADDRESSES,
  PROD.STOCK,
  PROD.PAYMENTS
  …
 TO HR;
```

```
GRANT SELECT ANY TABLE
   ON SCHEMA PROD
 TO HR;
```

Table Value Constructor (ISO SQL Standard)

NEW IN 23c

- Generate multiple rows at once

```
INSERT INTO bookings
 VALUES (12113, 'Vienna', '2022-09-21'),
        (62361, 'San Francisco', '2022-10-12'),
        (38172, 'Berlin', '2022-12-15');
```

```
SELECT *
FROM (VALUES (1,'Scott'),
             (2,'James'),
             (3,'John')
) t1 (employee_id, first_name);

EMPLOYEE_ID FIRST
----------- -----
          1 Scott
          2 James
          3 John
```


ROSENDIN

Table Value Constructor (ISO SQL Standard)

NEW IN 23c

- Generate multiple rows at once

```
WITH X (c1, c2, c3) AS (
VALUES (0, 1, 2),
       (3, 4, 5),
       (6, 7, 8)
   ) SELECT * FROM X;

        C1          C2          C3
---------- ---------- ----------
         0          1          2
         3          4          5
         6          7          8
```

```
Execution Plan
----------------------------
Plan hash value: 2575724336

----------------------------
| Id  | Operation | Name  |
----------------------------
|   0 | SELECT STATEMENT  |
|   1 |  VIEW             |
|   2 |   VALUES SCAN     |
----------------------------

Table Value Constructor
-----------------------------------------------

 2 - #tuples:3, #elems:3
      values:(0, 1, 2), (3, 4, 5), (6, 7, 8)
```

NEW IN
23c

Annotations

- Provide metadata for your data and data model
- Supported: tables, views, table/view columns, materialized views, indexes, domains and more

```
annotations
        ::= 'ANNOTATIONS' ( annotations_list )

annotations_list
        ::= { 'ADD' | 'DROP' } annotation ( ',' { 'ADD' | 'DROP' } annotation )

annotation
        ::= annotation_name annotation_value
```

Annotations

NEW IN 23c

- Define annotations as free-text keys or key/value pairs
- Add annotations to an object

```
CREATE TABLE customers (...)
  ANNOTATIONS (Sensitivity 'High',
               Departments 'Sales, Delivery',
               FrontOffice);
```


ROSENDIN

Annotations

NEW IN 23c

- Define annotations as free-text keys or key/value pairs
- Add annotations to an attribute like a table column

```
CREATE TABLE employee (
  id       NUMBER(5)
              ANNOTATIONS (Identity, Display 'Employee ID', Group 'Emp_Info'),
  name     VARCHAR2(50)
              ANNOTATIONS (Display 'Employee Name',  Group 'Emp_Info'),
  salary   NUMBER
              ANNOTATIONS (Display 'Employee Salary', UI_Hidden)
)
ANNOTATIONS (Display 'Employee Table');
```

BOOLEAN data type (ISO SQL Standard)

NEW IN 23c

```
CREATE TABLE emails (address VARCHAR2(1000), active BOOLEAN);

INSERT INTO emails VALUES ('joe.doe@gmail.com', TRUE);
INSERT INTO emails VALUES ('jame.doe@yahoo.com', FALSE);
INSERT INTO emails VALUES ('mary.smith@yahoo.com', 'YES');
INSERT INTO emails VALUES ('jim.watson@bt.co.uk', 0);

SELECT address FROM emails WHERE active;

ADDRESS
--------------------
joe.doe@gmail.com
mary.smith@yahoo.com
```


ROSENDIN

SELECT without FROM

NEW IN 23c

- SELECT on expressions no longer require FROM dual
  - DUAL table remains and can still be used

```
SELECT SYSDATE;

SYSDATE
--------------------
2022-09-21 22:18:52
```

```
SELECT 2*3 AS result;

RESULT
------
     6
```

```
SELECT my_func();

MY_FUNC
------------
Hello World!
```

4096 columns

NEW IN 23c

- 23c can support up to 4096 columns per table
- **COMPATIBILITY** needs to be set to 23.0.0

```
ALTER SYSTEM SET MAX_COLUMNS=EXTENDED;
```

Developer role

NEW IN 23c

- Grant/revoke developer privileges with just one command:

```
GRANT  DB_DEVELOPER_ROLE TO   dev_user;
REVOKE DB_DEVELOPER_ROLE FROM dev_user;
```

- Includes:
  - System privileges required to build a data model
  - Object privileges required to monitor and debug applications

Developer role

NEW IN 23c

- System privileges
  - ADMINISTER SQL TUNING SET
  - CREATE ANALYTIC VIEW
  - CREATE ATTRIBUTE DIMENSION
  - CREATE CUBE
  - CREATE CUBE BUILD PROCESS
  - CREATE CUBE DIMENSION
  - CREATE DIMENSION
  - CREATE DOMAIN
  - CREATE HIERARCHY
  - CREATE JOB
  - CREATE MATERIALIZED VIEW
  - CREATE MINING MODEL
  - CREATE MLE
  - CREATE PROCEDURE
  - CREATE SEQUENCE
  - CREATE SESSION
  - CREATE SYNONYM
  - CREATE TABLE
  - CREATE TRIGGER
  - CREATE TYPE
  - CREATE VIEW
  - DEBUG CONNECT SESSION
  - EXECUTE DYNAMIC MLE
  - EXECUTE ON JAVASCRIPT
  - FORCE TRANSACTION
  - ON COMMIT REFRESH

Developer role

NEW IN 23c

- Object privileges:
  - GRANT SELECT ON SYS.DBA_PENDING_TRANSACTIONS
  - GRANT SELECT ON V$SESSION, V$SESSTAT, V$STATNAME
- Included Roles:
  - RESOURCE
  - SODA_APP
  - CTXAPP

Better RETURNING clause

NEW IN 23c

- Return values for all DML statements (INSERT/UPDATE/DELETE/MERGE)
- Return OLD and NEW values

```
RETURNING CLAUSE ::=
  { RETURN | RETURNING } { OLD | NEW } expr
      [, { OLD | NEW } expr ] ...
  INTO variable [, variable ] ...
```

```
UPDATE employees SET salary=salary*2
 WHERE country = 'Austria'
RETURNING OLD salary, NEW salary
 INTO :old_salary, :new_salary;
```

```
MERGE INTO sales s USING
   (SELECT account, sale FROM ext) e
    ON (e.account=s.account)
  WHEN MATCHED THEN
    UPDATE SET s.sale=e.sale
  WHEN NOT MATCHED THEN
    INSERT (s.account, s.sale)
       VALUES (e.account, e.sale)
RETURNING s.account, e.sale
 INTO :n1, :n2;
```


ROSENDIN

Time Savers

NEW IN 23c

- Group By Column Alias / Position
- Direct Joins for UPDATE and DELETE
- IF [NOT] EXISTS
- Seamless Concatenations
- DEFAULT ON NULL for UPDATE or Insert Statements

GROUP BY column alias / position

NEW IN 23c

- No longer need to repeat lengthy expressions in the GROUP BY clause

```
SELECT extract(year FROM hiredate) AS hired_year, COUNT(*)
FROM emp
GROUP BY extract(year FROM hiredate)
HAVING extract(year FROM hiredate) > 1985;
```

```
SELECT extract(year FROM hiredate) AS hired_year, COUNT(*)
FROM emp
GROUP BY hired_year
HAVING hired_year > 1985;
```

Direct Joins for UPDATE and DELETE (ISO SQL Standard)

NEW IN 23c

- Update a table via a condition from a join

```
UPDATE employees e SET e.salaries = e.salaries * 2
 FROM departments d
  WHERE e.dept_id = d.dept_id
   AND d.name = 'Development';
```

```
DELETE FROM employees e
 FROM departments d
  WHERE e.dept_id = d.dept_id
   AND d.name = 'Sales'
   AND e.hire_date < TO_DATE('01-JAN-16','DD-MON-YY');
```

IF [NOT] EXISTS

NEW IN 23c

- Control DDL error condition

```
CREATE TABLE test123 (id NUMBER);

ORA-00955: name is already used by
an existing object

CREATE TABLE IF NOT EXISTS
  test123(id NUMBER);

Table created.
```

```
DROP TABLE test123;

ORA-00942: table or view does not
exist

DROP TABLE IF EXISTS test123;

Table dropped.
```

Seamless Concatenations

NEW IN 23c

- Before 23c

```
SELECT CONCAT(CONCAT(CONCAT('Hello', ' '), 'World'), '!') AS string;

STRING
------------
Hello World!
```

- With 23c

```
SELECT CONCAT('Hello', ' ', 'World', '!') AS string;

STRING
------------
Hello World!
```

DEFAULT ON NULL for UPDATE or Insert Statements

NEW IN 23c

```
drop table if exists t1 purge;

create table t1 (
  id  number,
  description  varchar2(15) default on null for insert and update 'banana'
);
```

```
insert into t1 (id, description) values (1, null);
insert into t1 (id) values (2);

select * from t1;

        ID DESCRIPTION
---------- ---------------
         1 banana
         2 banana
```

NEW IN 23c

Oracle released Oracle 23c Free

Overview

Developers can download and start using the Oracle 23c Free release to get a head start on new features of the Oracle database. Oracle provides a VirtualBox download that includes:

- Oracle Linux 8.7
- Oracle Database 23.2 Free - Developer Release for Linux x86-64
- Sample Schema and Tables
- Oracle REST Data Services 23.1
- Oracle SQLcl 23.1
- Oracle APEX 22.2

Limitations

- 12 GB of User Data storage
- Maximum RAM is 2 GB

Steps

- Download Image - https://www.oracle.com/database/free/download/
- Install Virtual Box - https://www.virtualbox.org/
- Follow the instructions and you should be ready to go in less than an hour.

ROSENDIN

Free Resources for Developers

oracle.github.io/free

Free Oracle Software

oracle-sql-features.github.io

Oracle Database Feature Documentation

asktom.oracle.com

Oracle Database Q&A Forum

livesql.oracle.com

Oracle SQL scratchpad

devgym.oracle.com

Learn SQL & PL/SQL

ROSENDIN