Application Development with the Oracle Database

Gerald Venzl

Master Product Manager
Oracle Development
2nd of April 2019
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.
Program Agenda

1. Overview and History
2. Vision
3. Data Management Strategy
4. Oracle and Modern Development
5. Open Source initiatives
6. Developer centric functionalities
7. Q & A
Program Agenda

1. Overview and History
2. Vision
3. Data Management Strategy
4. Oracle and Modern Development
5. Open Source initiatives
6. Developer centric functionalities
7. Q & A
Continuous Oracle Database Innovations

Preserving customer’s investment though each new Computing Era

Client-Server
- Stored Procedures
- Partitioning
- Parallel Query
- Unstructured Data

Internet
- Resource Management
- Real Application Clusters
- Data Guard
- XML

Big Data & Cloud
- Big Data SQL
- Multitenant
- In-Memory
- JSON
## Application Development Over The Years

### Release 1985 – 1997: 6, 7 and 8
- Stored Procedures & Triggers
- Referential Integrity
- Distributed Transactions
- AQ
- LOBs
- Spatial

### Release 1998 – 2012: 8i, 9i, 10g, 11g
- OLTP throughput (Row Locking, MVRC)
- Parallel Query Partitioning
- Online Operations
- RAC
- Data Guard
- Flashback
- Self-Managing Database
- Enterprise Manager
- Resource Management
- Automatic Storage Mgmt
- Encryption
- Real Application Testing
- Row Compression
- Columnar Compression
- Smart Scans
- Flash Cache

### Release 2013 - 2019: 19c
- Open Source Drivers (Python, Node.js and R)
- Pattern Matching
- OpenSource Drivers
- JSON
- REST Data Services
- NoSQL Database
- Application Continuity
- Migration Framework
- HTML5 – Desktop & Browser
- Javascript
- Opensource
- Cloud

---

Copyright © 2014 Oracle and/or its affiliates. All rights reserved.
Program Agenda

1. Overview and History
2. Vision
3. Data Management Strategy
4. Oracle and Modern Development
5. Open Source initiatives
6. Developer centric functionalities
7. Q & A
Oracle Database as a Data Platform

Development Services
- Node.js, Python, .NET, Java,
- PHP, Ruby, PL/SQL, C, C++,
- Perl, Go, EBR, REST Services,
- Advanced Queuing,
- APEX, SODA, Docker

Platform Services
- Cloud to On-Premise, Clustering,
- Microservices, Sharding, Security,
- High Availability, Isolation,
- Zero Data Loss, Administration

Analytical Services
- SQL, R, Columnar In-Memory,
- Advanced Analytics,
- Machine Learning, AI

Data Support
- Relational, JSON, XML,
- Spatial, Graph, RDF,
- Text, Binary, Object
- Stores, HDFS, Kafka,
- NoSQL Stores

Infrastructure Services
- Public Cloud, Cloud at Customer,
- Exadata, BDA, ZDLRA
Program Agenda

1. Overview and History
2. Vision
3. Data Management Strategy
4. Oracle and Modern Development
5. Open Source initiatives
6. Developer centric functionalities
7. Q & A
“Polyglot persistence will occur over the enterprise as different applications use different data storage technologies. It will also occur within a single application as different parts of an application’s data store have different access characteristics.”

Martin Fowler & Pramod Sadalage, Feb. 2012
http://martinfowler.com/articles/nosql-intro-original.pdf
what might Polyglot Persistence look like?

Rapid access for reads and writes. No need to be durable
Needs transactional updates. Tabular structure fits data
Needs high availability across multiple locations. Can merge inconsistent writes
Rapidly traverse links between friends, product purchases, and ratings

Speculative Retailers Web Application

User analysis
Financial Data
Shopping Cart
Recommendations
Product Catalog
Reporting
Analytics
User activity logs

Reds
PDM
Rux
Neo4j
MongoDB
PDM
Cassandra
Cassandra

Lots of reads, infrequent writes. Products make natural aggregate
Large-scale analytics on large cluster
High volume of writes on multiple nodes
SQL interfaces well with reporting tools

This is a very hypothetical example, we would not make technology recommendations without more contextual information

Source: The future is: NoSQL Databases Polyglot Persistence
http://martinfowler.com/articles/nosql-intro-original.pdf
Two Approaches to Polyglot Persistence

Single-model
- Relational Database
- Key-value Store

Multi-model
- Graph Database
- JSON/XML Database
- Relational Data
- Graph Data
- Key-value Data
- JSON & XML Data
Considerations for Polyglot Persistence

Multi-model Polyglot:
- Benefits of consolidation and standardization
- Standardized administration
- Consistent data security policies
- Simple integration across multiple data formats
- Transactions and data consistency

Single-model Polyglot:
- Benefits of specialization
  - Specialized APIs
  - Specialized data formats
  - Specialized access methods and indexes
Evolution of data management

- Cobol
- ISAM
- 1970s
Evolution of data management

- 1970s: Cobol, ISAM
- 1980s: Relational SQL

Complexity

1970s 1980s
Evolution of data management

- **1970s**: Cobol, ISAM
- **1980s**: Relational SQL, Object DBs (e.g., ZOPE, Versant, ObjectStore)
- **1990s**: XML, XPath, JSON

**Complexity**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |
Evolution of data management

1970s
Cobol ISAM

1980s
Object DBs

1990s
Multi-Model SQL

Relational SQL

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.
Evolution of data management

1970s: Cobol ISAM
1980s: Cobol, ISAM, Object DBs, Relational SQL
1990s: ZOPE, Versant, ObjectStore, Object DBs, Relational SQL, XML, XPath
2000s: Multi-Model SQL, MarkLogic, existdb, XML Server

Complexity
Evolution of data management

1970s: Cobol ISAM

1980s: ISAM, Object DBs

1990s: ZOPE, Relational SQL, Object DBs, XML

2000s: Multi-Model SQL, JSON, XML, XPath

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.
Evolution of data management

- **1970s**: Cobol, ISAM
- **1980s**: Relational SQL, ISAM
- **1990s**: Object DBs, Zope, Versant, ObjectStore
- **2000s**: Multi-Model SQL, XML, XPath
- **2010s**: Multi-Model SQL, JSON, Couchbase, MongoDB
Multi-model prevails over time

- **1970s**: Cobol, ISAM
- **1980s**: Relational SQL, Object DBs (e.g., ZOPE, VERSANT, ObjectStore), XML
- **1990s**: Multi-Model SQL, XML, XPath
- **2000s**: Multi-Model SQL, MarkLogic, existDB, Couchbase, JSON
- **2010s**: Multi-Model SQL, MongoDB
- **20??**: Next “big” thing, Multi-Model SQL

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.
Polyglot Persistence Market Trends

• Single-model architectures are most pervasive for ‘edge’ applications
  – New business & workload requirements
• Business applications naturally converge to multi-model architectures
  – Today’s ‘edge’ applications are tomorrow’s mainstream business applications
  – Efficiencies of multi-model architecture override advantages of special-purpose systems over time
• There will always be single-model polyglot architectures
  – Because there are always new ‘edge’ applications
  – Oracle’s single-model architectures:
  • Oracle Berkeley DB, Oracle NoSQL Database, Essbase, Oracle Big Data Spatial and Graph
Oracle Product Strategy for Polyglot Persistence

Support Both – Customer chooses which one to use

**Multi-model**

- Oracle Database supports multi-model persistence
  - Relational
  - XML
  - JSON
  - Text
  - Graph & Spatial

- Oracle Database provides integrated access to all database objects

**Single-model**

- Oracle supports multiple single-model data stores
  - Relational
  - Key/Value
  - XML
  - Spatial
  - Graph
  - OLAP

- Oracle integrates single-model polyglot environments via **Big Data SQL**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.
Program Agenda

1. Overview and History
2. Vision
3. Data Management Strategy
4. Oracle and Modern Development
5. Open Source initiatives
6. Developer centric functionalities
7. Q & A
## 2019 most popular development languages

### Stackoverflow

**Programming, Scripting, and Markup Languages**

<table>
<thead>
<tr>
<th>Language</th>
<th>All Respondents</th>
<th>Professional Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>69.8%</td>
<td></td>
</tr>
<tr>
<td>HTML</td>
<td>68.5%</td>
<td></td>
</tr>
<tr>
<td>CSS</td>
<td>65.1%</td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>57.0%</td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td>45.3%</td>
<td></td>
</tr>
<tr>
<td>Bash/Goshell</td>
<td>39.3%</td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td>38.3%</td>
<td></td>
</tr>
<tr>
<td>C#</td>
<td>34.4%</td>
<td></td>
</tr>
<tr>
<td>PHP</td>
<td>30.7%</td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td>25.4%</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>23.0%</td>
<td></td>
</tr>
<tr>
<td>TypeScript</td>
<td>17.4%</td>
<td></td>
</tr>
<tr>
<td>Ruby</td>
<td>10.1%</td>
<td></td>
</tr>
<tr>
<td>Swift</td>
<td>8.1%</td>
<td></td>
</tr>
</tbody>
</table>

1. JavaScript
2. Java
3. Python
4. PHP
5. C#
6. C++
### Oracle Database for the Developer

**Supporting all major development environments and APIs**

<table>
<thead>
<tr>
<th>LANGUAGE</th>
<th>DRIVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>OCI, ODPI-C</td>
</tr>
<tr>
<td>C++</td>
<td>OCCI</td>
</tr>
<tr>
<td>Java</td>
<td>JDBC</td>
</tr>
<tr>
<td>.NET</td>
<td>ODP.NET</td>
</tr>
<tr>
<td>Node.js</td>
<td>node-oracledb</td>
</tr>
<tr>
<td>Python</td>
<td>cx_Oracle</td>
</tr>
<tr>
<td>PHP</td>
<td>OCI8, PDO_OCI</td>
</tr>
<tr>
<td>R</td>
<td>ROracle</td>
</tr>
<tr>
<td>Go</td>
<td>goracle, rana, mattn</td>
</tr>
<tr>
<td>Rust</td>
<td>mimir</td>
</tr>
<tr>
<td>Ruby</td>
<td>ruby-oci8</td>
</tr>
<tr>
<td>Perl</td>
<td>DBD::Oracle</td>
</tr>
</tbody>
</table>

- **Red** for Oracle provided Drivers
- **Blue** for Open Source Drivers (Oracle contributions)
- **Gray** for Open Source Drivers (Third-party maintainers)

... and ODBC, OLE DB, Pro*C, Pro*COBOL, Pro*Fortran, SQLJ
JSON Support in Oracle Database

Powerful SQL Analytics

Data accessed via RESTful service or native API’s

Data persisted in database
In JSON

Oracle Database

Data analyzed via SQL
JSON Support in Oracle Database
Fast Application Development + Powerful SQL Access

Application developers:
Access JSON documents using REST API

POST /my_database/my_schema/customers HTTP/1.0
Content-Type: application/json
Body:
{
"firstName": "John",
"lastName": "Smith",
"age": 25,
"address": {
  "streetAddress": "21 2nd Street",
  "city": "New York",
  "state": "NY",
  "postalCode": "10021",
  "isBusiness": false
},
"phoneNumbers": [
  {"type": "home",
   "number": "212 555-1234" },
  {"type": "fax",
   "number": "646 555-4567" }
]
}

Analytical tools and business users:
Query JSON using SQL

SELECT
  c.json_document.firstName,
  c.json_document.lastName,
  c.json_document.address.city
FROM customers c;

firstName    lastName    address.city
"John"        "Smith"       "New York"
## Oracle vs Mongo DB

<table>
<thead>
<tr>
<th>Feature</th>
<th>Oracle</th>
<th>Mongo DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document-store : Store, index, and query JSON documents</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Simple Document-Centric API’s and REST support</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Query by Example (QBE) capability</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Joins within Documents, Within Collections and Across Collections</td>
<td>✓</td>
<td>Within Documents Only</td>
</tr>
<tr>
<td>Joins with Relational, XML, Spatial and Text Content</td>
<td>✓</td>
<td>Limited Support for Text</td>
</tr>
<tr>
<td>Standardized Query Language</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Integration with Industry Leading BI, Analytical and Reporting tools</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Concurrency Control, ACID Transactions, Read Consistency</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Enterprise Backup/Recovery and Disaster Recovery</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Architected for consolidation and multitenancy</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Oracle REST Data Services

REST-enable your data

Oracle Database (Relational)

Oracle Database (Document Store)

Oracle NoSQL Database
**Oracle REST Data Services**

**HTTP(s) API App-Dev with Relational Tables in Oracle Database**

---

**JSON**

```json
{
  "custno": 1001,
  "name": "Scott King",
  "address": "500 Main street, Innovation CA",
  "orders": [
    {
      "orderno": 404,
      "orderdate": "Feb 27, 2014",
      "status": "in process"
    },
    {
      "orderno": 303,
      "orderdate": "Feb 26, 2014",
      "status": "in process"
    }
  ]
}
```

---

**ORDS** maps standard URI requests to corresponding relational SQL (not schemaless): e.g., SQL SELECT from customers and orders table. ORDS also transforms the SQL results into the highly popular JavaScript Object Notation (JSON), other formats include HTML, binary and CSV. Fully committed to supporting any and all standards required by Fusion / SaaS / FMW; we are actively engaged in the ongoing dialog.

---

Copyright © 2015 Oracle and/or its affiliates. All rights reserved.
Oracle REST Data Services
Example: Query returning JSON for customer 1001

http://myhost/myapplication/custorders/simplequery { custno: 1001}
  • Map URI request to data access template
  • Bind custno (1001) to bindcustno
  • Execute select statement below
  • Note embedded cursor expression
  • Set format to JSON

Data Access Template

```
select c.*, 
    cursor(
        select *
        from orders o 
        where o.custno = c.custno) 
    orders 
from customers c 
where c.custno = :bindcustno
```

JSON Result

```
{ "custno": 1001,
  "name": "Scott King",
  "address": "500 Oracle Parkway, Redwood Shores, CA, 94065",
  "country": "USA",
  "class": "A",
  "orders": [ { "orderno": 303,
                "orderdate": "Feb 26, 2014",
                "status": "in process"},
             { "orderno": 202,
               "orderdate": "Jan 16, 2014",
               "status": "processed"},
             { "orderno": 101,
               "orderdate": "Dec 2, 2013",
               "status": "processed"} ] }
```
Oracle’s commitment to Single-model Polyglot

Oracle NoSQL Database

• **Developer Focus**
  - BASE & ACID txns
  - Tables / JSON / Binary
  - C, Java, Python & Node.js APIs
  - Secondary Indexes

• **Operations Focus**
  - Elastic Configuration
  - Secure Access
  - Data Center Support
  - Online management

• **Differentiating Features**
  - ACID transactions
  - Online rolling upgrades
  - Streaming large object support
  - **Strong Oracle technology Integration**
  - **Engineered Systems** and Commodity HW

[Diagram showing Oracle NoSQL Database architecture]


Copyright © 2015 Oracle and/or its affiliates. All rights reserved.
Oracle’s commitment to Single-model Polyglot

Oracle Big Data Spatial & Graph

• Massively-Scalable Graph Database
  – Scales to **trillions** of edges
  – Apache HBase
  – Oracle NoSQL Database

• In-Memory Graph Analytics
  – More than 30 graph analysis algorithms

• Simple, standard interfaces
  – Java
  – Tinkerpop: Blueprints, Gremlin, Rexster
  – Python

Oracle on Docker

• Oracle Database is fully supported on Docker
  – Oracle Linux 7
  – Red Hat Enterprise Linux 7

• Oracle image on Docker Store

• Docker build files on GitHub
Oracle on Docker

• Docker container contains single-PDB CDB
• PDB can be plugged, unplugged, etc.
• PDB can move bi-directional
Docker Store

- Oracle 12.1 & 12.2 images are available on Docker Store Registry
  - https://store.docker.com
Docker build files available on GitHub

- Repository: https://github.com/oracle/docker-images
- Build files for 18c, 12.2, 12.1, 11.2.0.2 XE
LiveSQL.oracle.com
The full power of Oracle SQL in your browser
Program Agenda

1. Overview and History
2. Vision
3. Data Management Strategy
4. Oracle and Modern Development
5. Open Source initiatives
6. Developer centric functionalities
7. Q & A
Oracle On GitHub

www.github.com/oracle

• Official Oracle representation on GitHub
• Examples and tools for Docker, Java, SQL, Python, Node.js, PL/SQL
• Repos regularly added
• Main source for Open Source components
Introducing Simple Oracle Document Access (SODA)

• An abstract API definition for
  – Collection Management: Ability to create and drop collections
  – Create, Retrieve, Update and Delete (CRUD) operations on documents
  – List operations on collections
  – Query-by-Example (QBE) for searching collections
  – Utility and control functions
    • Create and Drop Indexes
    • Bulk Insert

• Implementations currently available for JAVA and REST

• Support for NODE.js and other languages forthcoming
SODA for Java

• SODA implementation for the Java Developer
• Developers can store JSON documents in the Oracle Database without learning JDBC or SQL
• Uses a standard JDBC connection to talk to the database
• Supports transactions
• Enables hybrid application development
  – Mix and Match SODA and JDBC based operations in a single application
node-oracledb

• A simple, stable Oracle Database driver with out-of-the-box performance
• Ongoing contributions from Oracle
  – Support for latest Oracle Database features
  – 26 releases since January 2015
• Modular design
  – Underlying, simple DB access layer based on OCI
node-oracledb

• **Open source** development, release and support under Apache 2.0 license
  – GitHub repository ([www.github.com/oracle/node-oracledb](http://www.github.com/oracle/node-oracledb))
  – Approx. monthly release cycle

*Users can contribute under the Oracle Contributor Agreement.*

*Thanks to all who have contributed code, documentation and ideas*
cx_Oracle
for Python
cx_Oracle

• A simple, stable Oracle Database driver with out-of-the-box performance
• Ongoing contributions from Oracle
  – Support for latest Oracle Database features
• Modular design
  – Underlying, simple DB access layer based on OCI
cx_Oracle

• **Open source** development, release and support under BSD license
  – GitHub repository ([https://github.com/oracle/python-cx_Oracle](https://github.com/oracle/python-cx_Oracle))
  – Approx. monthly release cycle

*Users can contribute under the Oracle Contributor Agreement.*

*Thanks to all who have contributed code, documentation and ideas*
Program Agenda

1. Overview and History
2. Vision
3. Data Management Strategy
4. Oracle and Modern Development
5. Open Source initiatives
6. Developer centric functionalities
7. Q & A
Oracle Database 12c “Under the Radar” Features

- JSON
- Online Table Move
- Property Graph
- Invisible Columns
- Real Time Materialized Views
- Long Identifiers
- Online Tablespace Encryption
- Security Assessment Tool
- Application Continuity
- Longer Varchars
- SQL Plan Management Enhancements
- Auto Generated Sequences
- Auto List Partitioning
- Index Compression
- Index Usage Stats
128-byte identifiers for objects

```
CREATE TABLE VERY_VERY_LONG_TABLE_NAME_IDENTIFIER_THAT_IS_58_BYTES_LONG
(
    VERY_VERY_LONG_TEXT_COLUMN_WITH_DATA_TYPE_VARCHAR2_THAT_IS_72_BYTES_LONG VARCHAR2(25)
);

Table VERY_VERY_LONG_TABLE_NAME_IDENTIFIER_THAT_IS_58_BYTES_LONG created.

INSERT INTO VERY_VERY_LONG_TABLE_NAME_IDENTIFIER_THAT_IS_58_BYTES_LONG
VALUES ('Hello World!');

1 row inserted.

SELECT * FROM VERY_VERY_LONG_TABLE_NAME_IDENTIFIER_THAT_IS_58_BYTES_LONG;

VERY_VERY_LONG_TEXT_COLUMN
-------------------------
Hello World!
```
Case-insensitive Database and Column-level Collation

Greatly simplifies migration of case-insensitive functionality of 3rd-party products

- Linguistic-sensitive operations, e.g., comparison and sorting, on the column honor the declared collation

- Unspecified column collation is inherited from the default collation property of the parent table or schema

- \texttt{COLLATE} operator can be used to cast an explicit collation anywhere in an expression

```sql
CREATE TABLE product
( id NUMBER,
  name VARCHAR2(50) COLLATE BINARY_CI,
  comments VARCHAR2(500)
) DEFAULT COLLATION BINARY;

SELECT name, comments FROM product
WHERE name LIKE '%BASE%' OR comments COLLATE BINARY_CI LIKE '%REPORT%';
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Database</td>
<td></td>
</tr>
<tr>
<td>Activity-Based Management</td>
<td></td>
</tr>
<tr>
<td>Business Intelligence</td>
<td>Replaces Reports</td>
</tr>
</tbody>
</table>
PL/SQL deprecate pragma

CREATE PROCEDURE p AUTHID DEFINER IS
  PRAGMA DEPRECATE (p, 'p is deprecated. You must use p2 instead.');
BEGIN
  DBMS_Output.Put_Line('p');
END p;

PLW-06019: entity P is deprecated

CREATE PROCEDURE q authid Definer is
BEGIN
  p();
  DBMS_Output.Put_Line('q');
END q;

PLW-06020:
reference to a deprecated entity: p is deprecated. You must use p2 instead.
## New in 12.2 Approximate Query Processing

Delivers significantly **faster** analysis for **interactive** and highly **iterative** data exploration

| 98% | ± 0.0127 |

- Approximations for expensive aggregate calculations:
  - `APPROX_COUNT_DISTINCT (12.1)`
  - `APPROX_PERCENTILE`
  - `APPROX_MEDIAN`  
    - 6-13X faster, accuracy typically within < 1%
- Use with **ZERO code changes**
  - `approx_for_aggregation = TRUE`
- Accuracy and error rate provided
Top-N approximate aggregation
Interactive response times against terabytes of data

• Approximate results for common top-N queries
  – How many approximate page views did the top five blog posts get last week?
  – What were the top 50 customers in each region and their approximate spending?

• Order of magnitude faster processing with high accuracy (error rate < 0.5%)

• New approximate functions APPROX_COUNT(), APPROX_SUM(), APPROX_RANK()

Top 5 blogs with approximate hits

```sql
SELECT blog_post, APPROX_COUNT(*)
FROM weblog
GROUP BY blog_post
FETCH FIRST 5 ROWS ONLY;
```

Top 50 customers per region with approximate spending

```sql
SELECT region, customer_name,
    APPROX_RANK(PARTITION BY region
                ORDER BY APPROX_SUM(sales) DESC) appr_rank,
    APPROX_SUM(sales) appr_sales
FROM sales_transactions
GROUP BY region, customer_name
HAVING APPROX_RANK(..) <=50;
```
Inline external tables
Transparently access external data

- External table definition provided at runtime
  - Similar to inline view
- No need to pre-create external tables that are used one time only
  - Increased developer productivity

```sql
CREATE TABLE sales_xt
  (prod_id number, ... )
  TYPE ORACLE_LOADER
  ...
  LOCATION 'new_sales_kw13')
  REJECT LIMIT UNLIMITED );

INSERT INTO sales
SELECT * FROM sales_xt;

DROP TABLE sales_xt;
```
Private temporary tables
 transient tables useful for reporting applications

Global temporary tables
• Persistent, shared (global) table definition
• Temporary, private (session-based) data content
  – Data physically exists for a transaction or session
  – Session-private statistics

Private temporary tables (18c)
• Temporary, private (session-based) table definition
  – Private table name and shape
• Temporary, private (session-based) data content
  – Session or transaction duration
## Oracle Database 12 Temporal Support

<table>
<thead>
<tr>
<th>Transaction Time Temporal (Flashback Data Archive)</th>
<th>Valid Time Temporal</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Tracks transactional changes to a table over its lifetime</td>
<td>- Enables user to model &amp; query data for “real world validity”</td>
</tr>
<tr>
<td>- Typically used for compliance and auditing</td>
<td>- Typically used for insurance policies, financial markets, trade data &amp; future changes</td>
</tr>
<tr>
<td>- Enables the users to see the data as it was at a point in time in the past</td>
<td>- Users can model concepts such as the “Life time of an insurance policy”</td>
</tr>
</tbody>
</table>
Valid Time Temporal
Example

```sql
CREATE TABLE customers(
    custid NUMBER,
    custname VARCHAR2(30),
    custaddr1 VARCHAR2(50),
    custaddr2 VARCHAR2(50),
    custcity VARCHAR2(50),
    custstate VARCHAR2(2),
    custzip VARCHAR2(20),
    start_time TIMESTAMP,
    end_time TIMESTAMP,
    PERIOD FOR cust_valid_time (start_time, end_time));
```
### Valid Time Temporal Example

<table>
<thead>
<tr>
<th>custid</th>
<th>custname</th>
<th>custaddr1</th>
<th>custaddr2</th>
<th>custcity</th>
<th>custstate</th>
<th>custzip</th>
<th>start_time</th>
<th>end_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Acme Inc</td>
<td>123 Any Street</td>
<td>Suite 17</td>
<td>Anytown</td>
<td>CA</td>
<td>99999</td>
<td>01-JAN-15</td>
<td></td>
</tr>
</tbody>
</table>

**INSERT INTO** CUSTOMERS **VALUES** (1,'Acme Inc.','123 Any Street','Suite 17','Anytown','AS','99999', TO_TIMESTAMP('01-JAN-15') ,NULL);
Valid Time Temporal Example

<table>
<thead>
<tr>
<th>custid</th>
<th>custname</th>
<th>custaddr1</th>
<th>custaddr2</th>
<th>custcity</th>
<th>custstate</th>
<th>custzip</th>
<th>start_time</th>
<th>end_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Acme Inc</td>
<td>123 Any Street</td>
<td>Suite 17</td>
<td>Anytown</td>
<td>CA</td>
<td>99999</td>
<td>01-JAN-15</td>
<td>31-MAY-15</td>
</tr>
</tbody>
</table>

**UPDATE** customers

**SET** end\_time = **TO\_TIMESTAMP** ('31-MAY-15')

**WHERE** custid = 1;
### Valid Time Temporal Example

<table>
<thead>
<tr>
<th>custid</th>
<th>custname</th>
<th>custaddr1</th>
<th>custaddr2</th>
<th>custcity</th>
<th>custstate</th>
<th>custzip</th>
<th>start_time</th>
<th>end_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Acme Inc</td>
<td>123 Any Street</td>
<td>Suite 17</td>
<td>Anytown</td>
<td>CA</td>
<td>99999</td>
<td>01-JAN-15</td>
<td>31-MAY-15</td>
</tr>
<tr>
<td>1</td>
<td>Acme Inc</td>
<td>456 Another Street</td>
<td></td>
<td>Anytown</td>
<td>CA</td>
<td>99998</td>
<td>01-JUN-15</td>
<td></td>
</tr>
</tbody>
</table>

**INSERT INTO** CUSTOMERS **VALUES** (1, 'Acme Inc.', '456 Another Street', NULL, 'Anytown', 'AS', '99998', TO_TIMESTAMP('01-JUN-15'), NULL);
Valid Time Temporal Example

<table>
<thead>
<tr>
<th>custid</th>
<th>custname</th>
<th>custaddr1</th>
<th>custaddr2</th>
<th>custcity</th>
<th>custstate</th>
<th>custzip</th>
<th>start_time</th>
<th>end_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Acme Inc</td>
<td>123 Any Street</td>
<td>Suite 17</td>
<td>Anytown</td>
<td>CA</td>
<td>99999</td>
<td>01-JAN-15</td>
<td>31-MAY-15</td>
</tr>
<tr>
<td>1</td>
<td>Acme Inc</td>
<td>456 Another Street</td>
<td></td>
<td>Anytown</td>
<td>CA</td>
<td>99998</td>
<td>01-JUN-15</td>
<td></td>
</tr>
</tbody>
</table>

**SELECT** custaddr1, custaddr2, custcity, custstate, custzip  
**FROM** customers WHERE custid = 1;
Valid Time Temporal Example

<table>
<thead>
<tr>
<th>custid</th>
<th>custname</th>
<th>custaddr1</th>
<th>custaddr2</th>
<th>custcity</th>
<th>custstate</th>
<th>custzip</th>
<th>start_time</th>
<th>end_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Acme Inc</td>
<td>123 Any Street</td>
<td>Suite 17</td>
<td>Anytown</td>
<td>CA</td>
<td>99999</td>
<td>01-JAN-15</td>
<td>31-MAY-15</td>
</tr>
<tr>
<td>1</td>
<td>Acme Inc</td>
<td>456 Another Street</td>
<td></td>
<td>Anytown</td>
<td>CA</td>
<td>99998</td>
<td>01-JUN-15</td>
<td></td>
</tr>
</tbody>
</table>

EXEC DBMS_FLASHBACK_ARCHIVE.ENABLE_AT_VALID_TIME('CURRENT');

SELECT custid, start_time, end_time
FROM customers WHERE custid=1;
Valid Time Temporal Example

<table>
<thead>
<tr>
<th>custid</th>
<th>custname</th>
<th>custaddr1</th>
<th>custaddr2</th>
<th>custcity</th>
<th>custstate</th>
<th>custzip</th>
<th>start_time</th>
<th>end_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Acme Inc</td>
<td>123 Any Street</td>
<td>Suite 17</td>
<td>Anytown</td>
<td>CA</td>
<td>99999</td>
<td>01-JAN-15</td>
<td>31-MAY-15</td>
</tr>
<tr>
<td>1</td>
<td>Acme Inc</td>
<td>456 Another Street</td>
<td></td>
<td>Anytown</td>
<td>CA</td>
<td>99998</td>
<td>01-JUN-15</td>
<td></td>
</tr>
</tbody>
</table>

```
SELECT custid, start_time, end_time
FROM customers
AS OF PERIOD FOR cust_valid_time TO_TIMESTAMP('03-JUN-15');
```
SQL Pattern Matching
Simplified Analysis of Data

- Scalable discovery of business event sequences
  - Clickstream logs: sessionization, search behaviour
  - Financial transactions: fraud detection, double bottom (“W”) stock analysis
  - Telco: dropped calls
  - Medical sensors: automated medical observations and detections

Patterns are defined using regular expressions

```
Select * from Employees MATCH_RECOGNIZE (
  ... PATTERN(X+ Z{2})
  ...
)
```

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.
SQL Pattern Matching
Example: Find Double Bottom (W)

- Find double bottom (W) patterns and report:
- Beginning and ending date of the pattern
- Average Price Increase in the second ascent
- Modify the search to find only patterns that lasted less than a week

```
PATTERN (X+ Y+ W+ Z+)
DEFINE X AS (price < PREV(price))
```
SQL Pattern Matching

Example: Find Double Bottom (W)

- Find double bottom (W) patterns and report:
- Beginning and ending date of the pattern
- Average Price Increase in the second ascent
- Modify the search to find only patterns that lasted less than a week

```
PATTERN (X+ Y+ W+ Z+)
DEFINE X AS (price < PREV(price))
  Y AS (price > PREV(price))
```
SQL Pattern Matching
Example: Find Double Bottom (W)

• Find double bottom (W) patterns and report:
• Beginning and ending date of the pattern
• Average Price Increase in the second ascent
• Modify the search to find only patterns that lasted less than a week

Stock price

```
SELECT first_x, last_z
FROM ticker MATCH_RECOGNIZE (
    PARTITION BY name ORDER BY time
    MEASURES FIRST(x.time) AS first_x
    LAST(z.time) AS last_z
    ONE ROW PER MATCH
    PATTERN (X+ Y+ W+ Z+)
    DEFINE X AS (price < PREV(price))
    Y AS (price > PREV(price))
    W AS (price < PREV(price))
    Z AS (price > PREV(price))
```

Example: Find Double Bottom (W)
SQL Pattern Matching

Example: Find Double Bottom (W)

• Find double bottom (W) patterns and report:
  • Beginning and ending date of the pattern
  • Average Price Increase in the second ascent
  • Modify the search to find only patterns that lasted less than a week

```
SELECT first_x, last_z
FROM ticker MATCH_RECOGNIZE (
    PARTITION BY name ORDER BY time
    MEASURES FIRST(x.time) AS first_x,
    LAST(z.time)  AS last_z
    ONE ROW PER MATCH
    PATTERN (X+ Y+ W+ Z+)
    DEFINE X AS (price < PREV(price)),
               Y AS (price > PREV(price)),
               W AS (price < PREV(price)),
               Z AS (price > PREV(price))
)
```
SQL Pattern Matching
More power to directly applied to your data

```
if (q.isEmpty() && !next.isEmpty()) {
    state = "F";
} else {
    next = lineNext.getQuantity();
    if (!q.isEmpty() && (prev.isEmpty() || (eq(q, prev) && gt(q, next)))) {
        state = "S";
        return state;
    }
    if (gt(q, prev) && gt(q, next)) {
        state = "T";
        return state;
    }
    if (lt(q, prev) && lt(q, next)) {
        state = "B";
        return state;
    }
    if (!q.isEmpty() && (next.isEmpty() || (gt(q, prev) && eq(q, next)))) {
        state = "E";
        return state;
    }
    if (q.isEmpty() || eq(q, prev)) {
        state = "F";
        return state;
    }
    return state;
}
```

```
private boolean eq(String a, String b) {
    if (a.isEmpty() || b.isEmpty()) {
        return false;
    }
    return a.equals(b);
}
```

```
private boolean gt(String a, String b) {
    if (a.isEmpty() || b.isEmpty()) {
        return false;
    }
    return Double.parseDouble(a) > Double.parseDouble(b);
}
```

```
private boolean lt(String a, String b) {
    if (a.isEmpty() || b.isEmpty()) {
        return false;
    }
    return Double.parseDouble(a) < Double.parseDouble(b);
}
```

```
public String getState() {
    return this.state;
}
```

```
SELECT first_x, last_z
FROM ticker MATCH_RECOGNIZE ()
PARTITION BY name ORDER BY time
MEASURES FIRST(x.time) AS first_x,
LAST(z.time) AS last_z
ONE ROW PER MATCH
PATTERN (X+ Y+ W+ Z+)
DEFINE X AS (price < PREV(price)),
Y AS (price > PREV(price)),
W AS (price < PREV(price) AND
    z.time - FIRST(x.time) <= 7 ),
Z AS (price > PREV(price) AND
    z.time - FIRST(x.time) <= 7 )
```

250+ Lines of Java
12 Lines of SQL

Less code, easier to maintain, faster to write
PL/SQL in SQL
PL/SQL functions embedded in “with” clause

WITH
  FUNCTION get_domain(url VARCHAR2) RETURN VARCHAR2 IS
    pos BINARY_INTEGER;
    len BINARY_INTEGER;
  BEGIN
    pos := INSTR(url, 'www.');
    len := INSTR(SUBSTR(url, pos + 4), '.') - 1;
    RETURN SUBSTR(url, pos + 4, len);
  END;
SELECT
  DISTINCT get_domain(catalog_url)
FROM
  orders;
IDENTITY
Auto increment for Oracle

• Create a table with an id column that is always populated

```sql
CREATE TABLE t1
(id NUMBER GENERATED AS IDENTITY,
 first_name VARCHAR2(30))
;
```

• Create a table with an id column that is populated if not provided

```sql
CREATE TABLE t2
(id NUMBER GENERATED BY DEFAULT AS IDENTITY
 (START WITH 100 INCREMENT BY 10),
 first_name VARCHAR2(30))
;
```
32k VARCHAR2/NVARCHAR2
Longer strings to store

• Enable 32k VARCHAR2 support

```
ALTER SYSTEM set MAX_STRING_SIZE = EXTENDED scope = SPFILE;
```

• Create a table with 32k VARCHAR2

```
CREATE TABLE Applicants
(id NUMBER GENERATED AS IDENTITY,
 first_name VARCHAR2(30),
 last_name VARCHAR2(30),
 application DATE,
 CV VARCHAR2(32767))
```
Row Limit
SQL Standard for row limiting

• Select only the first 5 rows

```
SELECT employee_id, last_name
    FROM employees
    ORDER BY employee_id
    FETCH FIRST 5 ROWS ONLY;
```

• Select only the first 5% of rows including rows that “tie”

```
SELECT employee_id, last_name, salary
    FROM employees
    ORDER BY salary
    FETCH FIRST 5 PERCENT ROWS WITH TIES;
```
Polymorphic Tables: Self-Describing, Fully Dynamic SQL

- Part of ANSI 2016
- Encapsulate sophisticated algorithms
  - Hides implementation of algorithms
  - Leverage powerful, dynamic capabilities of SQL
  - Pass in any table-columns for processing
  - Returns SQL rowset (table, JSON, XML doc etc.)
    - E.g. return credit score and associated risk level

```sql
SELECT state_id, ..., AVG(credit_score), risk
FROM CREDIT_RISK(
  tab => scott.customers,
  cols => columns(dob, zip, loan_default),
  outs => columns(credit_score, risk_level))
WHERE risk_level = 'High'
GROUP BY state_id;
```
Oracle Advanced Queuing (AQ)
Messaging and Notification in the Database

• JMS support
• PL/SQL, OCI, JDBC, .NET support
• Integrated with the Database
• Messaging Gateway
AQ-JMS Sharded Queues
New with 12.1.0.2

• A single logical queue with many “shards”
  – A “shard” is a way of obtaining higher concurrency and throughput via horizontal partitioning.

• Automatic management of session affinity to shards
• Automatic management of table partitions to avoid contention
• Automatic management of partition instance affinity
• Integrated with the database to optimize performance
AQ-JMS Sharded Queues

Key benefits

• Higher throughput
• Less system resource consumption
• Many enqueuers and dequeuers across multiple RAC instances
• Large number of subscribers
AQ Sharded Queues
Architecture for Scalability and Performance

Single logical queue model

Physical Queue: partitions mapped to shards with instance affinity on RAC

Instance 1

Instance 2

Instance 3

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.
AQ-JMS Sharded Queues

Key benefits

• Higher throughput
• Less system resource consumption
• Large number of subscribers
• Event-based listener with fewer database connections
• Many concurrent enqueuers and dequeuers across multiple RAC instances
• Backwards Compatible for Standard JMS based applications
  – just recreate the AQ in the database
Analytic Views

• Moves business logic (Aggregations, Hierarchies, Calculations) back into database

• Simple SQL for complex analytic queries
  – no joins or GROUP-BY clauses necessary

• Works on top of pre-existing tables or views
  – no persistent storage

• Built-in data visualization via APEX
Analytic Views

A new type of view in the Oracle Database

• A new type of view in the Oracle Database
  – Business model and calculation rules are embedded within the Analytic View

• Analytic Views as easily queried with simple SQL and MDX
  – With a smart Analytic View, SQL generation is easy
  – MDX Provider (OLE DB for OLAP) supports Excel PivotTable connections

• Access data from tables, views, external tables and Big Data SQL
  – Use Analytic Views to organize and present a wide variety of data
Analytic Views
Easier Access To Your Data

Your Data → Analytic View → SQL → Your Applications

Organized & Enhanced

Simple SQL
Three New Database Objects

- **Attribute Dimensions**
  - Map to data objects with dimension / attribute data
  - Identify the roles of columns

- **Hierarchies**
  - Organizes levels into aggregation and drill paths
  - A new type of view that can be queried with SQL

- **Analytic Views**
  - Maps to data objects with fact / measure data
  - A new type of view that can be queried with SQL and MDX
Three New Database Objects

- Attribute Dimensions

```sql
CREATE OR REPLACE ATTRIBUTE
DIMENSION time_attr_dim
USING time_dim
ATTRIBUTES
(year_id
  CLASSIFICATION caption VALUE 'YEAR_ID'
  CLASSIFICATION description VALUE 'YEAR ID',
year_name
  CLASSIFICATION caption VALUE 'YEAR_NAME'
  CLASSIFICATION description VALUE 'Year',
...)
LEVEL month
CLASSIFICATION caption VALUE 'MONTH'
CLASSIFICATION description VALUE 'Month'
KEY month_id
..."DETERMINES (month_end_date,
quarter_id,
season,
season_order,
month_of_year,
month_of_quarter) ...
```

- Hierarchies

```sql
CREATE OR REPLACE HIERARCHY time_hier
CLASSIFICATION caption VALUE 'CALENDAR'
CLASSIFICATION description VALUE 'CALENDAR'
USING time_attr_dim
(month CHILD OF quarter CHILD OF year)
```

- Analytic Views

```sql
CREATE OR REPLACE ANALYTIC VIEW sales_av
CLASSIFICATION caption VALUE 'Sales AV'
CLASSIFICATION description VALUE 'Sales Analytic View'
CLASSIFICATION created_by VALUE 'George Jones'
USING sales_fact
DIMENSION BY
(time_attr_dim
  KEY month_id REFERENCES month_id
  HIERARCHIES (time_hier DEFAULT,
    time_season_hier,
    time_year_season_hier,
    time_month_of_qtr_hier),
...)
MEASURES
(sales FACT sales
  CLASSIFICATION caption VALUE 'Sales'
  CLASSIFICATION description VALUE 'Sales'
  CLASSIFICATION format_string VALUE '$999,999,999,999.99',
units FACT units
  CLASSIFICATION caption VALUE 'Units'
  CLASSIFICATION description VALUE 'Units Sold'
  CLASSIFICATION format_string VALUE '$999,999,999,999',
...)
```
Analytic Views

Organize and Enhance Data

• Transforms data into a business model and presentation layer in the database
  – Data is organized for easy access and navigation
  – Data is easily extended with interesting calculations and aggregations
  – Data is easily queried with simple SQL

• Easily defined with SQL
  – Complete applications defined with just a few SQL statements
  – Supported by SQL Developer
Analytic Views
Better for Everyone

• For the data warehouse architect and developer
  – Easily extend star schema with aggregate data and calculations

• For the application developer
  – Simplifies metadata management and SQL generation

• For the business user
  – Built-in, browser-based data visualization via APEX application
Analytic Views

• How would you build this application?
  – Analysis of health insurance coverage rates in the United States
  – Coverage rates by time, counties and states
  – Geographic comparisons
  – Measure improvement over time
  – Interactive data visualization tools for end users
Analytic Views

• This application can be built with 5 SQL statements
  – Create 2 hierarchies (4 SQL statements)
  – Create 1 analytic view (1 SQL statement)

• Instantly accessible via APEX based data visualizer

• Entirely in the Database

Simple SQL

Analytic View

Data Tables, Views, etc.
Analytic Views

Simple SQL

```sql
SELECT time_hier.member_name AS TIME,
       geog_hier.member_name AS GEOGRAPHY,
       pct_insured
FROM insured_av HIERARCHIES(time_hier,geog_hier)
WHERE time_hier.level_name = 'YEAR'
       AND geog_hier.level_name = 'STATE'
ORDER BY time_hier.hier_order,
       geog_hier.hier_order;
```

Fact data is selected from analytic views using SQL

Analytic views are views on top of a star schema. No storage structures
The HIERARCHIES clause specifies the dimensions and hierarchies for this query.

No JOIN or GROUP BY clauses in analytic view queries.
Analytic Views

Simple SQL

SELECT time_hier.member_name AS TIME,
       geog_hier.member_name AS GEOGRAPHY,
       pct_insured
FROM insured_av HIERARCHIES(time_hier, geog_hier)
WHERE time_hier.level_name = 'YEAR'
AND   geog_hier.level_name = 'STATE'
ORDER BY time_hier.hier_order,
         geog_hier.hier_order;

Standardized columns such as ‘member_name’ are selected from the hierarchies

A typical star query would instead select a column such as ‘time.year’
Levels of aggregation are specified in the WHERE clause

When filtering on the level ‘State’ for the time hierarchy, the member named will include California, New York, etc.
Analytic Views

Simple SQL

SELECT time_hier.member_name AS TIME,
     geog_hier.member_name AS GEOGRAPHY,
     pct_insured
FROM insured_av HIERARCHIES(time_hier, geog_hier)
WHERE time_hier.level_name = 'YEAR'
AND geog_hier.level_name = 'COUNTY'
ORDER BY time_hier.hier_order,
     geog_hier.hier_order;

To drill, just update the WHERE clause. Everything else remains the same.

The calculations automatically use new hierarchy levels.
Analytic Views

Simple SQL

```
SELECT time_hier.member_name AS TIME,
geog_hier.member_name AS GEOGRAPHY,
pct_insured_diff_us_avg
FROM insured_av HIERARCHIES(time_hier,geog_hier)
WHERE time_hier.level_name = 'YEAR'
AND geog_hier.level_name = 'COUNTY'
ORDER BY time_hier.hier_order,
geog_hier.hier_order;
```

To select a calculation, just select columns.

Calculations are express in the analytic view so they can just be selected in the query.
Analytic Views
Embedded Calculations

• Easily create new measures
  – Simplified syntax based on business model
  – Includes dimensional and hierarchical functions

Add Percent Uninsured Difference from US Average with a single line of code

\[
\text{SHARE\_OF}(\text{pct\_uninsured} \text{ HIERARCHY geog\_hier} \text{ MEMBER country ['USA']} - 1)
\]
Analytic Views
Embedded Calculations

Add time series calculations with a single line of code

```
LAG_DIFF_PERCENT(pct_insured)
OVER (HIERARCHY time_hier
OFFSET 1 ACROSS ANCESTOR AT LEVEL year)
```
### “Standard” and Analytic Views

<table>
<thead>
<tr>
<th>Feature</th>
<th>“Standard” View</th>
<th>Analytic View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Sources (FROM)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Joins</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Business Model-Based Calculations</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Automatic Hierarchical Columns</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Automatic Multi-Level Aggregation</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Automatic Filter Expansion</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Automatic Outer Join</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Automatic Order of Calculation</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Presentation Metadata</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Machine Learning and Advanced Analytics

• Machine Learning Algorithms available since 9i Release 2 (2002)
• New algorithms in 18
  – Random Forests for Classification
  – Neural Networks for both classification and regression
  – Explicit Semantic Analysis ML algorithm extended to support classification
  – Time Series via Exponential Smoothing
  – CUR decomposition-based algorithm for attribute and row importance
• New ability to export ML models to C and Java for applications deployment
Property Graph Improvements

- PGQL – Property Graph Query Language
  - SQL-like declarative language to query in-memory and in-database Property Graph
  - Supports graph pattern matching and recursive path queries
  - Proposing as ISO standard; Language is also licensed under Apache https://github.com/oracle/pgql-lang

- In-memory virtual columns for RDF Graph
  - Up to 100X faster queries performance

- RDF Graph networks now support list-hash composite partitioning
  - 5 to 10 times query performance improvement
JSON Support
JSON Queries using SQL

• Simple Queries

```
SELECT j.PODOCUMENT
FROM J_PURCHASEORDER j
WHERE j.PODOCUMENT.PONumber = 1600;
```

• Advanced queries using JSON path expressions

```
SELECT JSON_VALUE(PODOCUMENT, 
'$.LineItems[0].Part.UnitPrice' returning NUMBER(5,3))
FROM J_PURCHASEORDER p
WHERE JSON_VALUE(PODOCUMENT, '$.PONumber' RETURNING 
NUMBER(10)) = 1600;
```

– Complies with SQL:2016 syntax
JSON integration with PL/SQL

- New PL/SQL objects enable fine grained manipulation of JSON content
  - `JSON_OBJECT_T` : for working with JSON objects
  - `JSON_ARRAY_T` : for working with JSON Arrays
- `JSON_OBJECT_T` and `JSON_ARRAY_T` are subtypes of `JSON_ELEMENT_T`
- These objects provide a set of methods for manipulating JSON
- Piecewise updates of JSON documents now supported in PL/SQL
WITH FUNCTION updateTax(JSON_DOC in VARCHAR2) RETURN VARCHAR2 IS

    jo JSON_OBJECT_T;
    price NUMBER;
    taxRate NUMBER;

BEGIN
    jo := JSON_OBJECT_T(JSON_DOC);
    taxRate := jo.get_Number('taxRate');
    price := jo.get_Number('total');
    jo.put('totalIncludingTax', price * (1 + taxRate));
    RETURN jo.to_string();
END;

ORDERS AS ( SELECT '{"taxRate":0.175,"total":10.00}' JSON_DOCUMENT
                           FROM dual
                 )

SELECT JSON_DOCUMENT, updateTax(JSON_DOCUMENT)
  FROM ORDERS;

<table>
<thead>
<tr>
<th>JSON_DOCUMENT</th>
<th>UPDATETAX (JSON_DOCUMENT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{&quot;taxRate&quot;:0.175,&quot;total&quot;:10.00}</td>
<td>{&quot;taxRate&quot;:0.175,&quot;total&quot;:10.00,&quot;totalIncludingTax&quot;:11.75}</td>
</tr>
</tbody>
</table>
Data Guide: Understanding your JSON documents

• Metadata discovery: discovers the structure of collection of JSON documents
  – Optional: deep analysis of JSON for List of Values, ranges, sizing etc.

• Automatically Generates
  – Virtual columns
  – Relational views
    • De-normalized relational views for arrays
  – Reports/Synopsis of JSON structure
Data Guide: Automatic Schema Inference

Table containing JSON documents

```
SQL> desc MOVIE_TICKETS
NAME     TYPE
--------- -------
BOOKING_ID RAW(16)  
BOOKING_TIME TIMESTAMP(6)
BOOKING_DETAILS VARCHAR2(4000)
```

JSON DataGuide

```
JSON
{
  "Theater":"AMC 15",
  "Movie":"Jurassic World 3D",
  "Time":"2015-11-26T18:45:00",
  "Tickets":{
    "Adults":2
  }
}
```

Table enhanced with virtual columns

```
SQL> desc MOVIE_TICKETS
NAME     TYPE
--------- -------
BOOKING_ID RAW(16)  
BOOKING_TIME TIMESTAMP(6)
BOOKING_DETAILS VARCHAR2(4000)
BOOKING_DETAILS$Movie VARCHAR2 (16)
BOOKING_DETAILS$Theater VARCHAR2 (16)
BOOKING_DETAILS$Adults NUMBER
BOOKING_DETAILS$Time VARCHAR2 (32)
```

```
DBMS_JSON.AddVC( 'MOVIE_TICKETS', 'BOOKING_DETAILS' );
```
JSON Search Index : A universal index for JSON content

```
CREATE SEARCH INDEX JSON_SEARCH_INDEX
ON J_PURCHASEORDER (PO_DOCUMENT) FOR JSON;
```

- Supports searching on JSON using key, path and value
- Supports range searches on numeric values
- Supports full text searches:
  - Full boolean search capabilities (and, or, and not)
  - Phrase search, proximity search and "within field" searches.
  - Inexact queries: fuzzy match, soundex and name search.
  - Automatic linguistic stemming for 32 languages
  - A full, integrated ISO thesaurus framework
Query Optimizations for JSON

Exadata Smart Scans

- Exadata Smart Scans execute portions of SQL queries on Exadata storage cells
- JSON query operations ‘pushed down’ to Exadata storage cells
  - Massively parallel processing of JSON documents

In-Memory Columnar Store

- Virtual columns, included those generated using JSON Data Guide loaded into In-Memory Virtual Columns
- JSON documents loaded using a highly optimized In-Memory binary format
- Query operations on JSON content automatically directed to In-Memory
Native JSON Generation

```sql
SQL> SELECT JSON_OBJECT('Id' is EMPLOYEE_ID, 'FirstName' is FIRST_NAME,
                        'LastName' is LAST_NAME) JSON
       FROM HR.EMPLOYEES
       WHERE EMPLOYEE_ID = 100;

JSON

{ "Id" : 100 , "FirstName" : "Steven" , "LastName" : "King" }

SQL>
```

- JSON generation functions available:
  - JSON_OBJECT / JSON_OBJECTAGG
  - JSON_ARRAY / JSON_ARRAYAGG
JSON Enhancements

Simpler development of JSON-centric applications using Oracle Database

• Generate large JSON documents from relational data
  – JSON generation extended to supports LOB’s
• New SODA (Simple Oracle Document Access) drivers
  – OCI and PL/SQL now added, in additional to JSON and REST
  – Simple, non-relational (‘nosql-like’) API for accessing JSON data
• New TREAT (<expression> AS JSON) operator
  – Dynamically declare operands to be handled as JSON data, enabling more seamless JSON optimizations
• Extended key length for JSON search indexes
  – Raise the key length from 64 bytes to 255 bytes; enables faster search queries for JSON objects containing long key names.
Multitenant
Oracle Data and User Data
Before 12.1: Oracle and user data intermingle over time

- New database contains Oracle meta-data only
- Populate database with user data
  - Oracle and customer meta-data intermingled
  - Portability challenge!
- Multitenant fix: Horizontally-partitioned data dictionary
  - Only Oracle-supplied meta-data remains in root
Application Containers
Programs replicated across PDBs

the brooklyn bean V2
Application Containers

Root container for your applications

• Application Container comprises
  – Application Root (Master)
  – Application PDBs (for each Tenant)
  – Application Seed (for provisioning)

• PDBs share application objects
  – Code, metadata and data

• Further simplifies management
  – Apply updates to application container
  – Sync tenant PDBs from central master

• Suitable for all applications
  – SaaS, franchise, divisional, etc.
Application Containers
Share & propagate across multiple PDBs

- Bank Card
- Bank Account
- Account Type
  - Credit
  - Debit
  - Check
  - Savings

- Person

- PDB$SEED
- APP ROOT
- PDB1
- PDB2
- PDB3
What is an Application Container?

- An Application container is a collection of PDBs consisting of Application Root and all Application PDBs associated with it.
Application Containers
The future of Database Application Development

- Application Root PDB for defining application master
  - Metadata and common data shared across tenant PDBs
- Install one copy of your application
- Instant provisioning of an Application PDB/Tenant (with a seed PDB)
- Container Data views for reporting across PDBs (CONTAINERS clause based)
- Supports in-place simple patching
- Supports Unplug/Plug upgrade across Application Root
Sharding
Oracle Database Sharding

Oracle Database for web-scale applications

- RAC and Data Guard meet needs of over 99% of applications while preserving application transparency
- Some Global-Scale OLTP applications prefer to shard massive databases into a farm of smaller databases
  - Avoid scalability or availability edge cases of a single large database
  - Willing to customize data model and applications to enable transactions to be automatically routed to the right shard
- Native SQL for sharding tables across up to 1000 Shards
  - Routing of SQL based on shard key, and cross shard queries
  - Online addition and reorganization of shards
  - Linear scalability of data, workload, users with isolation
Application Suitability for Sharding

**OLTP Applications with the Following Characteristics**

- Applications for massive scale  
  - E.g. e-commerce, mobile, social etc.
- Applications must be shard-aware
- Primary usage pattern  
  - Single-shard operations based on shard key, e.g. customer_id, account_id etc.
Oracle Sharding Automated Distribution
Enhanced SQL syntax for Sharding

```
CREATE SHARDED TABLE Customers
( CustId VARCHAR2(60) NOT NULL,
  FirstName VARCHAR2(60),
  LastName VARCHAR2(60),
  ...
  PRIMARY KEY(CustId),
)
PARTITION BY CONSISTENT HASH (CustId)
```

- SQL syntax for creating sharded tables
  - Not proprietary APIs as with NoSQL
- Creation of a sharded table automatically partitions data across shards
  - Transparent resharding as data grows
- Choice of sharding methods:
  - System managed - consistent hash
  - User defined - range, list
  - Composite - range-hash, list-hash
- Common reference data (e.g. Price List) is automatically duplicated on all shards
- Supports shard placement in specific geographies to satisfy government data privacy
## Sharded Schema

### Customers
<table>
<thead>
<tr>
<th>Customer</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Mary</td>
</tr>
<tr>
<td>456</td>
<td>John</td>
</tr>
<tr>
<td>999</td>
<td>Peter</td>
</tr>
</tbody>
</table>

### Orders
<table>
<thead>
<tr>
<th>Order</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>4001</td>
<td>123</td>
</tr>
<tr>
<td>4002</td>
<td>456</td>
</tr>
<tr>
<td>4003</td>
<td>999</td>
</tr>
<tr>
<td>4004</td>
<td>456</td>
</tr>
<tr>
<td>4005</td>
<td>456</td>
</tr>
</tbody>
</table>

### Line Items
<table>
<thead>
<tr>
<th>Line</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>40011</td>
<td>1001</td>
</tr>
<tr>
<td>40012</td>
<td>4003</td>
</tr>
<tr>
<td>40013</td>
<td>4001</td>
</tr>
<tr>
<td>40014</td>
<td>4004</td>
</tr>
<tr>
<td>40015</td>
<td>4003</td>
</tr>
</tbody>
</table>

### Products
<table>
<thead>
<tr>
<th>SKU</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Coil</td>
</tr>
<tr>
<td>101</td>
<td>Piston</td>
</tr>
<tr>
<td>102</td>
<td>Belt</td>
</tr>
</tbody>
</table>

### Sharded
- Customers, Orders, Line Items are sharded.
- Products are duplicated.
Sharded Table Family – Enhanced SQL DDL Syntax

CREATE SHARED TABLE Customers
( CustNo NUMBER NOT NULL,
  Name VARCHAR2(50),
  ...
  Class VARCHAR2(3),
  CONSTRAINT RootPK PRIMARY KEY(CustNo)
)
PARTITION BY CONSISTENT HASH (CustNo)
PARTITIONS AUTO
TABLESPACE SET ts1;

CREATE LOOKUP TABLE Products(
  SKU NUMBER(4) PRIMARY KEY,
  Product VARCHAR2(20),
  Price NUMBER(6,2))
)
TABLESPACE dupl;

CREATE SHARED TABLE Orders
( OrderNo NUMBER(5),
  CustNo NUMBER(3),
  OrderDate DATE,
  ...
  CONSTRAINT CustFK FOREIGN KEY (CustNo)
    REFERENCES Customers(CustNo)
)
PARTITION BY REFERENCE (CustFK);
Routing Support on Client for Highest Speed

- Clients pass **sharding key** (e.g. Customer ID) to Connection pool, connection is routed to the right shard
- **Fast**: caching key ranges on client ensures that most accesses go directly to the shard
- **Scalable**: easily scales with more clients and shards
- Supports UCP, OCI, ODP.NET, and JDBC
Non-Shard Key Access & Cross-Shard Queries

- If **client does not pass shard key** to Connection pool, the connection is made to the coordinator database
- Coordinator parses SQL and will proxy/route request to one or more shards
  - Supports shard pruning and scatter-gather
- For developer convenience and not for high performance
- Supports many but not all Queries
- No Update support
EZConnect Improvements

• Simplification of Easy Connect syntax
• Easy Connect adaptor will now accept a list of name value pairs
  – For example: SDU, RETRY_COUNT, CONNECT_TIMEOUT, etc.)
• Will now enable multiple hosts/ports in the connect string
  – Typically used in load-balancing client connections.

```
$> sqlplus soe/soe @(DESCRIPTION=
   (ADDRESS_LIST=
      (LOAD_BALANCE=ON) (ADDRESS=(PROTOCOL=tcp)(HOST=salesserver1)(PORT=1522))
      (ADDRESS=(PROTOCOL=tcp)(HOST=salesserver2)(PORT=1522))
      (ADDRESS=(PROTOCOL=tcp)(HOST=salesserver3)(PORT=1522)))
   (CONNECT_DATA=(SERVICE_NAME=sales.us.example.com)))
```
EZConnect Improvements

• Simplification of Easy Connect syntax

• Easy Connect adaptor will now accept a list of name value pairs
  – For example: SDU, RETRY_COUNT, CONNECT_TIMEOUT, etc.)

• Will now enable multiple hosts/ports in the connect string
  – Typically used in load-balancing client connections.

```sql
$> sqlplus soe/soe@//salesserver1,salesserver2,salesserver3:1522/sales.us.example.com
```
EZConnect Improvements

• Simplification of Easy Connect syntax

• Easy Connect adaptor will now accept a list of name value pairs
  – For example: SDU, RETRY_COUNT, CONNECT_TIMEOUT, etc.)

• Will now enable multiple hosts/ports in the connect string
  – Typically used in load-balancing client connections.

```sql
$> sqlplus soe/soe@(DESCRIPTION=
  (retry_count=3) (connect_timeout=60) (transport_connect_timeout=30)
  (ADDRESS=(PROTOCOL=tcp) (HOST=salesserver1) (PORT=1521))
  (CONNECT_DATA=(SERVICE_NAME=sales.us.example.com)))
```
EZConnect Improvements

• Simplification of Easy Connect syntax
• Easy Connect adaptor will now accept a list of name value pairs
  – For example: SDU, RETRY_COUNT, CONNECT_TIMEOUT, etc.)
• Will now enable multiple hosts/ports in the connect string
  – Typically used in load-balancing client connections.

$> sqlplus soe@/salesserver1:1521/sales.us.example.com?connect_timeout=60&transport_connect_timeout=30&retry_count=3
Program Agenda

1. Overview and History
2. Vision
3. Data Management Strategy
4. Oracle and Modern Development
5. Open Source initiatives
6. Developer centric functionalities
7. Q & A