3 Application Development

Oracle Database provides the most comprehensive platform with both application and data services to make development and deployment of enterprise applications simpler.

JSON

JSON-Relational Duality

JSON Relational Duality Views are fully updatable JSON views over relational data. Data is still stored in relational tables in a highly efficient normalized format but can be accessed by applications in the form of JSON documents.

Duality views provide you with game-changing flexibility and simplicity by overcoming the historical challenges developers have faced when building applications using relational or document models.

View Documentation

JSON Schema

JSON Schema-based validation is allowed with the SQL condition `IS JSON` and with a PL/SQL utility function. A JSON schema is a JSON document that specifies allowed properties (field names) and the corresponding allowed data types, and whether they are optional or mandatory.

By default, JSON data is schemaless, providing flexibility. However, you may want to ensure that your JSON data contains particular mandatory fixed structures and typing, besides other optional and flexible components, which can be done via JSON Schema validation.

View Documentation

XML and JSON Search Index Enhancements

The Oracle Text XML search index syntax and JSON search index syntax are now consistent. Additionally, the performance of JSON and XML search indexes has been improved.

Using the same syntax for XML or JSON search indexes and better performance increase productivity.

View Documentation

Changes for JSON Search Index and Data Guide

JSON search index and JSON data guide are enhanced in these ways. The first two represent changes in the default behavior.

1. When creating a JSON search index, by default a data guide is not created.
2. By default, `DBMS_JSON` procedures `create_view`, `get_view_sql`, and `add_virtual_columns` resolve name conflicts; that is, the default value of parameter `resolveNameConflicts` is `TRUE`, not `FALSE`. This means that if a resulting field name exists in the same data guide then it is suffixed with a new sequence number, to make it unique.
3. Function `json_dataguide` is enhanced to detect ISO 8601 date-time string values, using flag option `DBMS_JSON.detect_datetime`.

When this option is present, field values that are strings in the ISO 8601 date and time formats supported by Oracle are represented in a data guide with the value of field `type` not as `string` but as `timestamp`.

The default changes improve usability and performance for JSON data guides.

View Documentation

Comparing and Sorting JSON Data Types

JSON data type can now be used directly in a WHERE, ORDER BY, and GROUP BY clause.

The broader applicability of the JSON data type in SQL constructs simplifies your application development and improves the performance of your applications by avoiding the need for explicit casts.

View Documentation

DBMS_AQ Support for JSON Arrays

You can use a JSON data type array as the payload for Advanced Queuing (AQ) message-passing functions, which process an array of messages as a single operation. This applies to the AQ interfaces for C (Oracle Call Interface), PL/SQL, and Java (JDBC).

Advanced Queuing can directly use JSON data for its bulk message passing. With JSON being an increasingly popular format for data exchange, this functionality provides more flexible application development and improves developer productivity.

View Documentation

EMPTY STRING ON NULL for JSON Generation

When generating JSON data from relational data, a SQL `NULL` input value results in a JSON `null` value by default.

In Oracle SQL, a SQL `NULL` value cannot be distinguished from an empty string value (`''`). This means that an empty SQL string input is treated the same as SQL `NULL`. This behavior can sometimes confuse users.

When using a SQL/JSON generation function such as `json_object`, for `NULL` input values of a SQL character data type, such as `CLOB` and `VARCHAR2`, a user can specify that an empty JSON string (`""`) be created. The same is true for function json_scalar.

With this feature, generating a JSON empty string (`""`) from an empty SQL string is easy and efficient. Without this feature, a user needs to use a complex `CASE` statement to do the same.

View Documentation

Enhancement to JSON_TRANSFORM

`JSON_TRANSFORM` is extended to support right-hand-side path expressions, nested paths, and arithmetic operations. A SORT operator is supported which allows sorting the elements in an array.

`JSON_TRANSFORM` is the main SQL operator for modifying JSON data, both for on-disk updates and transient changes in the `SELECT` clause of a query. This enhancement increases update capabilities, such as arithmetic calculations and operations on nested arrays and raises developer productivity.

View Documentation

JSON Data Guide Format FORMAT_SCHEMA

Format `FORMAT_SCHEMA` produces a data guide that you can use to validate JSON documents.

You can produce JSON data guide documents that you can use to validate JSON documents.

View Documentation

JSON Type Modifiers

A JSON type column can store any JSON, this includes JSON objects, arrays and scalars. There are cases where a user would want to make sure that a JSON type is always an object. For this, we added type modifiers, for example, `data JSON (object)`.

This feature allows the user to specify the top level type of a JSON (object, array, scalar).

View Documentation

JSON Type Support for External Tables

Support for access and direct-loading of JSON-type columns is provided for external tables. JSON data type is supported as a column type in the external table definition. Newline-delimited and JSON-array file options are supported, which facilitates importing JSON data from an external table.

This feature makes it easier to load data into a JSON-type columns.

View Documentation

JSON-to-Duality Converter

Given an existing set of JSON collections as input, the JSON-To-Duality Converter creates a set of JSON-relational duality views, based on normalized relational schemas, that support the same document collections. This creation needs no user supervision, but users can override schema recommendations.

This feature provides one part of the JSON-to-Duality Migrator, which is a set of PL/SQL procedures to move document-centric applications and their JSON documents from a document database to duality views in Oracle Database.

View Documentation

JSON-to-Duality Importer

This feature imports application data from a set of JSON collections into JSON-relational duality views that have been created using the JSON-to-Duality Converter.

This feature provides one part of the JSON-to-Duality Migrator, which is a set of PL/SQL procedures to move document-centric applications and their JSON documents from a document database to duality views in Oracle Database.

View Documentation

JSON/JSON_VALUE will Convert PL/SQL Aggregate Type to/from JSON

The PL/SQL JSON constructor is enhanced to accept an instance of a corresponding PL/SQL aggregate type, returning a JSON object or array type populated with the aggregate type data.

The PL/SQL JSON_VALUE operator is enhanced so that its returning clause can accept a type name that defines the type of the instance that the operator is to return.

JSON constructor support for aggregate data types streamlines data interchange between PL/SQL applications and languages that support JSON.

View Documentation

JSON_ARRAY Constructor by Query

A subquery can be used as an argument to SQL/JSON function JSON_ARRAY to define the array elements. This functionality is part of the SQL/JSON standard.

This feature increases your developer productivity and higher interoperability with other SQL/JSON standard-compliant solutions.

View Documentation

JSON_BEHAVIOR Initialization Parameter to Control SQL/JSON Runtime Behavior

You can use the initialization parameter `JSON_BEHAVIOR` to change the default behavior of SQL/JSON operators within a database session.

You can change the default return data type, default type-compatibility, and default error behavior during a session for applicable SQL/JSON operators.

This lets you enforce consistent session-level JSON processing behavior, reducing the need for explicit overriding of default behaviors within individual SQL statements.

View Documentation

JSON_EXPRESSION_CHECK Parameter

A new parameter `JSON_EXPRESSION_CHECK` allows to enable/disable a JSON query check. The values are `on` and `off`. The default is `off`. For now, this parameter is limited to JSON-relational duality views. An error is raised if a JSON path expression on a duality view does not match to an underlying column, for example if the path expression has a typo. The error is raised during query compilations.

This simplifies working with JSON-relational duality views, as incorrect JSON path expressions do not need to be debugged at runtime but instead are flagged at query compilation time (by raising an error).

View Documentation

JSON_TRANSFORM Operators ADD_SET and REMOVE_SET

Oracle SQL function `JSON_TRANSFORM` operators `ADD_SET` and `REMOVE_SET` work with JSON arrays as if they are *sets*; that is, as if their elements are unordered and unique (no duplicates).

- Operator `ADD_SET` adds a value to an array only if the value is not already an element.
- Operator `REMOVE_SET` removes all occurrences of a given value from an array.

Application code can more concisely update arrays that it uses as sets.

View Documentation

LOBs Returned by SQL Functions for JSON can be Value-Based

Wherever a SQL function for JSON returns a LOB value, the returning clause can specify that the LOB be value-based. By default, a LOB reference is returned instead. For example:

```
JSON_SERIALIZE (data returning CLOB VALUE)
```

Value-based LOBs are are easier to use because they do not need to be freed explicitly. The database fully manages the lifecycle of value-based LOBs and frees them when appropriate.

View Documentation

New JSON Data Dictionary Views

New dictionary views `*_JSON_INDEXES` and `*_TABLE_VIRTUAL_COLUMNS` have been added.

These new views provide better insight into the database objects that have been created to work with JSON data.

View Documentation

ORDERED in JSON_SERIALIZE

The SQL function `JSON_SERIALIZE` has an optional keyword `ORDERED`, which reorders the key-value pairs alphabetically (ascending only). It can be combined with optional keywords `PRETTY` and `ASCII`.

Ordering the result of serialization makes it easier for both tools and humans to compare values.

View Documentation

Precheckable Constraints using JSON SCHEMA

To avoid sending invalid data to the database, an application can often precheck (validate) it. PL/SQL function `DBMS_JSON_SCHEMA.describe` provides JSON schemas that apps can use to perform validation equivalent to that performed by database column-level check constraints, and it records constraints that have no equivalent JSON schema.

Applications can also check which columns are precheckable with a JSON schema by consulting static dictionary views `ALL_CONSTRAINTS`, `DBA_CONSTRAINTS`, and `USER_CONSTRAINTS`.

When you create or alter a table you can use keyword `PRECHECK` to determine whether column check constraints can be prechecked outside the database. If no equivalent JSON schema exists for a given `PRECHECK` column check constraint then an error is raised.

Early detection of invalid data makes applications more resilient and reduces potential system downtime. All applications have access to the same information about whether data for a given column is precheckable, and if so what JSON schema validates it.

View Documentation

Predicates for JSON_VALUE and JSON_QUERY

JSON path expressions with predicates can be used in `JSON_VALUE` and `JSON_QUERY`. The functionality is part of the SQL/JSON standard.

Applying JSON path expressions more widely for querying JSON data boosts your developer's productivity and simplifies code development.

View Documentation

SCORE Ancillary Operator for JSON_TEXTCONTAINS()

This feature allows you to return a score for your `JSON_TEXTCONTAINS()` queries by using the `SCORE()` operator.

You can also order the results by the score.

JSON_TEXTCONTAINS function gains a new parameter for use with the SCORE() function allowing for an improved development experience.

View Documentation

SODA Enhancements

Various extensions are made to the SODA API:

- **Merge and patch:** New SODA operations mergeOne and mergeOneAndGet.
- **Embedded Keys:** You can now embed the key of a document in the document itself. This is used for MongoDB-compatible collections.
- **Dynamic Data Guide:** The operation to compute a data guide on the fly is extended to other SODA languages, besides PL/SQL and C.
- **Sampling operation:** The sampling operation is extended to other SODA languages, besides PL/SQL and C.
- **Flashback:** The operation to use flashback is extended to other SODA languages, besides PL/SQL and C.

- **Hints and monitoring:** Hints and SQL monitoring are extended to other SODA languages, besides PL/SQL and C.
- **Explain plan:** Obtaining a SQL execution plan is extended to other SODA languages, besides PL/SQL and C.
- **Data Guard and Golden Gate:** You can now replicate SODA collections using Oracle Data Guard and Oracle GoldenGate.
- **Index Discovery:** You can now fetch all indexes for a given SODA collection.
- **Multivalue index creation:** New SODA APIs for PL/SQL, C, and Java to create multivalue indexes.

These extensions increase the usability and capabilities of SODA in general, thus improving developer productivity.

View Documentation

Tools to Migrate JSON Text Storage to JSON Type Storages

The new PL/SQL procedure, `DBMS_JSON.json_type_convertible_check`, checks whether existing data stored as JSON text can be migrated to JSON data type. There are several alternative ways to migrate the data after this check succeeds.

Leveraging the binary JSON data type format provides the best performance for processing JSON data. Providing a simple and easy way to ensure existing data can be transformed successfully to binary JSON format helps you to adopt the preferred storage format for JSON data.

View Documentation

WHERE Clauses in JSON-Relational Duality Views

When creating a JSON-relational duality view you can use simple `WHERE` clauses to limit the rows from which to generate JSON data from underlying tables. As one kind of use case, you can create multiple duality views, whose documents contain different data depending on the values in a discriminating column. For example, with the same underlying table you can define views for data from different countries, using a `WHERE` clause that selects only table rows whose country-code column has a given value (for example, `FR` for France). The JSON documents supported by a country view reflect this requirement, and the requirement is enforced for updates.

`WHERE` clauses in view definitions allow fine-grained control of the data that is to be included in a JSON document supported by a duality view.

View Documentation

SQL

Schema Annotations

Schema annotations enable you to store and retrieve metadata about database objects. These are name-value pairs or simply a name. These are free-form text fields applications can use to customize business logic or user interfaces.

Annotations help you use database objects in the same way across all applications. This simplifies development and improves data quality.

View Documentation

Direct Joins for UPDATE and DELETE Statements

Join the target table in `UPDATE` and `DELETE` statements to other tables using the `FROM` clause. These other tables can limit the rows changed or be the source of new values.

Direct joins make it easier to write SQL to change and delete data.

View Documentation

IF [NOT] EXISTS Syntax Support

DDL object creation, modification, and deletion now support the `IF EXISTS` and `IF NOT EXISTS` syntax modifiers. This enables you to control whether an error should be raised if a given object exists or does not exist.

The `IF [NOT] EXISTS` syntax can simplify error handling in scripts and by applications.

View Documentation

New Database Role for Application Developers

The `DB_DEVELOPER_ROLE` role provides an application developer with all the necessary privileges to design, implement, debug, and deploy applications on Oracle databases.

By using this role, administrators no longer have to guess which privileges may be necessary for application development.

View Documentation

Aggregation over INTERVAL Data Types

You can pass `INTERVAL` data types to the `SUM` and `AVG` aggregate and analytic functions.

This enhancement makes it easier for developers to calculate totals and averages over `INTERVAL` values.

View Documentation

Automatic PL/SQL to SQL Transpiler

PL/SQL functions within SQL statements are automatically converted (transpiled) into SQL expressions whenever possible.

Transpiling PL/SQL functions into SQL statements can speed up overall execution time.

View Documentation

Client Describe Call Support for Tag Options

Annotations enable you to store and retrieve metadata about database objects. These are either name-value pairs or only a name. These are free-form text fields that applications can use to customize business logic or user interfaces.

Annotations help you to use database objects in the same way, across all applications. This simplifies development and improves data quality.

View Documentation

DEFAULT ON NULL for UPDATE Statements

You can define columns as `DEFAULT ON NULL` for update operations, which was previously only possible for insert operations. Columns specified as `DEFAULT ON NULL` are automatically updated to the specific default value when an update operation tries to update a value to `NULL`.

This feature simplifies application development and removes your need for complex application code or database triggers to achieve the desired behavior. Development productivity is increased and code becomes less error-prone.

View Documentation

DESCRIBE Now Supports Column Annotations

The SQL*Plus `DESCRIBE` command can now display annotation information for columns that have associated annotations available.

Annotations help you to use database objects in the same way across all applications. This simplifies development and improves data quality.

View Documentation

Data Use Case Domain Metadata Support in OCCI

Provide access to the Data Use Case Domain metadata (domain name and domain schema) for the database columns described in OCCI (Oracle C++ Call Interface) applications.

Database adds Data Use Case Domains to columns and the column metadata need to expose the same in all the data access drivers.

View Documentation

Data Use Case Domains

A data use case domain is a dictionary object that belongs to a schema and encapsulates a set of optional properties and constraints for common values, such as credit card numbers or email addresses. After you define a use case domain, you can define table columns to be associated with that domain, thereby explicitly applying the domain's optional properties and constraints to those columns.

With use case domains, you can define how you intend to use data centrally. They make it easier to ensure you handle values consistently across applications and improve data quality.

View Documentation

Error Message Improvement

The Oracle Call Interface (OCI) `OCIError()` function has been enhanced to optionally include an Oracle URL with error messages. The URL page has additional information about the Oracle error.

This feature allows users to access information about the cause of the error and the actions that can be taken.

View Documentation

Extended CASE Controls

The `CASE` statement is extended in PL/SQL to be consistent with the updated definitions of `CASE` expressions and `CASE` statements in the SQL:2003 Standard [ISO03a, ISO03b].

Dangling predicates allow tests other than equality to be performed in simple `CASE` operations. Multiple choices in `WHEN` clauses allow `CASE` operations to be written with less duplicated code.

View Documentation

GROUP BY Column Alias or Position

You can now use column alias or `SELECT` item position in `GROUP BY`, `GROUP BY CUBE`, `GROUP BY ROLLUP`, and `GROUP BY GROUPING SETS` clauses. Additionally, the `HAVING` clause supports column aliases.

These enhancements make it easier to write `GROUP BY` and `HAVING` clauses. It can make SQL queries much more readable and maintainable while providing better SQL code portability.

View Documentation

Improved TNS Error Messages

This feature enhances common TNS error messages by providing more information, such as cause of the error and the corresponding action to troubleshoot it.

Having a better description of errors improves diagnosability.

View Documentation

Multilingual Engine Support for SQL BOOLEAN Data Type

Oracle Database features a native SQL `BOOLEAN` data type. The server-side JavaScript engine fully supports the data type on all interfaces.

When using JavaScript to write stored code in Oracle, this feature allows you to take full advantage of the capabilities offered by the new SQL `BOOLEAN` data type.

View Documentation

Oracle C++ Call Interface (OCCI) Support for SQL BOOLEAN Data Type

Oracle C++ Call Interface (OCCI) now supports querying and binding of the new SQL `BOOLEAN` data type.

Using the SQL `BOOLEAN` data type enables applications to represent state more clearly.

View Documentation

Oracle Client Driver Support for SQL BOOLEAN Data Type

Oracle client drivers support fetching and binding the new `BOOLEAN` database column.

Applications can use the native database `BOOLEAN` column data type with a native driver `BOOLEAN` data type. This enhancement makes working with `BOOLEAN` data types easier for developers.

View Documentation

SELECT Without FROM Clause

You can now run `SELECT` expression-only queries without a `FROM` clause.

This new feature improves SQL code portability and ease of use for developers.

View Documentation

SQL BOOLEAN Data Type

Oracle Database now supports the ISO SQL standard-compliant `BOOLEAN` data type. This enables you to store `TRUE` and `FALSE` values in tables and use `BOOLEAN` expressions in SQL statements.

The `BOOLEAN` data type standardizes the storage of `Yes` and `No` values and makes it easier to migrate to Oracle Database.

View Documentation

SQL UPDATE RETURN Clause Enhancements

The `RETURNING INTO` clause for `INSERT`, `UPDATE`, `DELETE` and `MERGE` statements are enhanced to report old and new values affected by the respective statement. This allows developers to use the same logic for each of these DML types to obtain values pre- and post-statement execution. Old and new values are valid only for `UPDATE` statements. `INSERT` statements do not report old values and `DELETE` statements do not report new values. `MERGE` can return both old and new values.

The ability to obtain old and new values affected by `INSERT`, `UPDATE`, `DELETE` and `MERGE` statements, as part of the SQL command's execution, offers developers a uniform approach to reading these values and reduces the amount of work the database must perform.

View Documentation

SQL*Plus Support for SQL BOOLEAN Data Type

SQL*Plus supports the new SQL `BOOLEAN` data type in SQL statements and the `DESCRIBE` command. Enhancements to the `COLUMN` and `VARIABLE` command syntax have also been made.

SQL*Plus scripts can take advantage of the new SQL `BOOLEAN` data type for easy development.

View Documentation

Table Value Constructor

The database's SQL engine now supports a `VALUES` clause for many types of statements. This new clause allows for materializing rows of data on the fly by specifying them using the new syntax without relying on existing tables. Oracle supports the `VALUES` clause for the `SELECT`, `INSERT`, and `MERGE` statements.

The introduction of the new `VALUES` clause allows developers to write less code for ad-hoc SQL commands, leading to better readability with less effort.

View Documentation

Unicode 15.0 Support

The National Language Support (NLS) data files for `AL32UTF8` and `AL16UTF16` character sets are updated to match version 15.0 of the Unicode Standard character database.

This enhancement enables Oracle Database to conform to the latest version of the Unicode Standard.

View Documentation

Graph

Native Representation of Graphs in Oracle Database

Oracle Database now has native support for property graph data structures and graph queries.

Property graphs provide an intuitive way to find direct or indirect dependencies in data elements and extract insights from these relationships. The enterprise-grade manageability, security features, and performance features of Oracle Database are extended to property graphs. Developers can easily build graph applications using existing tools, languages, and development frameworks. They can use graphs in conjunction with transactional data, JSON, Spatial, and other data types.

View Documentation

Support for the ISO/IEC SQL Property Graph Queries (SQL/PGQ) Standard

The ISO SQL standard has been extended to include comprehensive support for property graph queries and creating property graphs in SQL. Oracle is among the first commercial software products to support this standard.

Developers can easily build graph applications with SQL using existing SQL development tools and frameworks. Support of the ISO SQL standard allows for greater code portability and reduces the risk of application lock-in.

View Documentation

Property Graph: Native Representation of Graphs in Oracle Database

Oracle Database now has native support for property graph data structures and graph queries.

Property graphs provide an intuitive way to find direct or indirect dependencies in data elements and extract insights from these relationships. The enterprise-grade manageability, security features, and performance features of Oracle Database are extended to property graphs. Developers can easily build graph applications using existing tools, languages, and development frameworks. They can use graphs in conjunction with transactional data, JSON, Spatial, and other data types.

View Documentation

Property Graph: Support for the ISO/IEC SQL Property Graph Queries (SQL/PGQ) Standard

The ISO SQL standard has been extended to include comprehensive support for property graph queries and creating property graphs in SQL. Oracle is among the first commercial software products to support this standard.

Developers can easily build graph applications with SQL using existing SQL development tools and frameworks. Support of the ISO SQL standard allows for greater code portability and reduces the risk of application lock-in.

View Documentation

Property Graph: Use JSON Collections as a Graph Data Source

SQL/PGQ queries can be executed on graphs represented as a JSON column (SQL/PGQ is the ISO standard for property graphs).

Developers can store a graph as a schema-less object in the database. Vertices and edges in a graph can have varying number and types of properties.

View Documentation

Property Graph: Use Native Representation of Graphs in Oracle Database with Graph Tools

Developers can visualize graphs and graph query results that use the native property graph object in Oracle Database.

Developers can query, analyze, and visualize graphs created by SQL DDL statements by using built-in advanced tools in Oracle Database. Existing applications can use this native representation of graphs without changing tools and the user interface.

View Documentation

RDF Graph: Execute Graph Analytics Algorithms with RDF Graphs

Oracle Graph algorithms in Graph Server can be used with RDF graphs.

You can now benefit from popular graph analytics algorithms, such as PageRank and Community Detection, potentially enhancing strategic decision-making and enabling deeper insights.

View Documentation

Microservices

Kafka APIs for TxEventQ

Transactional Event Queues (TxEventQ) now support the KafkaProducer and KafkaConsumer classes from Apache Kafka.

Oracle Database can now be used as a source or target for applications using the Kafka APIs.

View Documentation

ODP.NET: Advanced Queuing and Transactional Event Queues

ODP.NET Core and managed ODP.NET now support Advanced Queuing (AQ) and Transactional Event Queues (TxEventQ) application programming interfaces (APIs) that can be used in modern applications, such as microservices. TxEventQ's highly optimized and partitioned implementation leverages the functions of Oracle database so that producers and consumers can exchange messages at high throughput, by storing messages persistently, and propagate messages between queues on different databases. TxEventQ are a high performance partitioned implementation with multiple event streams per queue, while AQ is a disk-based implementation for simpler workflow use cases.

ODP.NET developers can leverage the same APIs no matter if they use TxEventQ or AQ. The APIs provide access to a robust and feature-rich message queuing systems integrated with Oracle database. It can be used with web, mobile, IoT, and other data-driven and event-driven applications to stream events or communicate with each other as part of a workflow.

View Documentation

Prometheus/Grafana for Oracle

Prometheus/Grafana for Oracle will provide database metrics for developers running in a Kubernetes/Docker (K8S) environment. Database metrics are stored in Prometheus, a time-series database and metrics tailored for developers are displayed using Grafana dashboards. A database metrics exporter aids the metrics exports from database views into Prometheus time series database.

Developers of modern applications like microservices use observability at the app tier and often overlook the data tier. Data-driven applications don't get a full picture of the execution and performance of the application. Traditional database metrics are seen through AWR reports and Enterprise Manager, which are more targeted to the DBAs and less to the developers. For developers and architects, Prometheus and Grafana have become the tools for configuring metrics dashboards, setting alerts and taking remedial action. Developers can now tie in the app-tier metrics, Kubernetes container metrics, and Oracle database metrics on behalf of the application together in a single dashboard. In addition to metrics, logs and tracing is also enabled to truly get unified observability on a single pane of glass.

View Documentation

Python and REST Drivers for Transactional Event Queues (TxEventQ)

Database 23ai introduces support in new languages for Transactional Event Queues (TxEventQ). TxEventQ can now be used in Python and with REST APIs (REST APIs implemented to be like Kafka's Confluent REST APIs). TxEventQ already has support for PL/SQL, C/C++, and Java using JMS or JDBC.

This feature increases developer productivity by allowing REST APIs applications to take advantage of Transactional Event Queues (TxEventQ) to handle application and data events. With increasing popularity of Python for Machine Learning applications, TxEventQ in the Oracle Database can now be part of the Machine Learning application data and events infrastructure.

View Documentation

Saga APIs using Oracle Saga Framework

Oracle Saga APIs are implemented in the database and provide a framework to implement transactional semantics for microservices built with the Oracle Database.

The orchestrator Saga framework provides a way to maintain atomic data consistency across microservices.

Sagas are concurrent and execute local transactions in each participant database making it more efficient than distributed ACID transactions, thereby simplifying application code and increasing developer productivity.

View Documentation

Transactional Event Queues (TxEventQ) Propagation

Queues are used widely to send and receive events and messages between applications, increasingly being built as microservices. Transactional Event Queues (TxEventQ) are queues built into the Oracle Database. Queue propagation allows multiple databases to act as producers and consumers of events and messages. Producer applications can send events in queues in one database, set up queue propagation to a remote database, and Consumer applications can consume events in queues in the remote database.

Queue propagation is used to consolidate critical events and data from remote locations to a central location for consolidated processing. Propagation is used to operate Transactional Event Queues (TxEventQ) as a reliable and secure Event Mesh, with multiple queues across multiple databases participating to send events and messages across the enterprise reliably and to remote subscribers with permissions. TxEventQ supports exactly-once messaging which makes it simpler to build and test applications.

View Documentation

General

.NET Metrics

.NET Metrics are application numerical measurements collected at regular time intervals for the purposes of monitoring and alerting about application health. In an ODP.NET setting, metrics can monitor connection statistics, such as number of ODP.NET hard connections to the database, number of active connections, or number of free connections.

ODP.NET Core and managed ODP.NET support .NET Metrics. ODP.NET metrics can be published to and analyzed by the rich and expansive toolsets integrated with OpenTelemetry and .NET Metrics, such as Grafana and Prometheus.

View Documentation

Dynamic Performance Views for Table and Partition Access Tracking

Read access to tables and partitions is tracked with a new dynamic performance view `[G]V$TABLE_ACCESS_STATS`and exposed in a user-friendly manner as data dictionary views `[DBA | ALL | USER]_TABLE ACCESS_STATS`, providing a deeper understanding of the access frequency of tables and individual partitions.

Allowing the user to see how often tables and individual partitions are read enables you to understand the importance and frequency of your data for better assessment of your lifecycle management of your data.

View Documentation

Efficient Table DDL Change Notification

Applications can now be notified when DDLs occur on tables.

Applications that need or want to be aware of table metadata can be notified of DDL changes rather than having to continuously poll for them.

View Documentation

Enhanced Inter-Session Communication with DBMS_PIPE

DBMS_PIPE, an in-database messaging framework for inter-session communication, got enhanced to support a broader set of applications and use cases. DBMS_PIPE now can share messages across multiple database sessions with concurrent reads, provides more flexibility in managing messages overall, and supports persistence and inter-instance and inter-database communication through object store buffering.

Providing a more comprehensive in-database inter-session messaging and communication enables more applications to take advantage of DBMS_PIPE, improving the reliability and scalability of applications. It also increases developer productivity by eliminating the need for more complex application architectures.

View Documentation

GB18030-2022 Support

The implementation of the Oracle client character set ZHS32GB18030 is updated to support the latest GB18030-2022 standard.

This feature enables Oracle Database to conform to the latest edition of the GB18030 standard, which is required for all software products sold in China.

View Documentation

JDBC RSI Support for Data Load Mode

In RSI stream mode, connection and prepared statement instances are created for every batch. With the new data load mode, the instances are created once and saved in the thread's local context.

This feature brings faster data ingestion into the Oracle Database.

View Documentation

ODP.NET: Asynchronous Programming

ODP.NET supports the .NET Task Asynchronous Programming (TAP) model with the core and managed drivers.

With support for TAP and the async and await keywords, ODP.NET data access operations are more responsive and easier to develop for asynchronicity.

View Documentation

ODP.NET: OpenTelemetry

OpenTelemetry is a popular open-source observability framework for instrumenting, generating, collecting, and exporting telemetry data. It provides a common specification and protocol so that multiple services can furnish a unified version of traces, metrics, and logs.

Numerous managed ODP.NET and ODP.NET Core APIs have been instrumented to support OpenTelemetry tracing. Developers can customize the ODP.NET OpenTelemetry trace settings and use manual, dynamic, or automatic instrumentation when needed.

With OpenTelemetry support, monitoring, tracking, and analyzing how ODP.NET operations interact in cloud computing, microservices and distributed systems becomes easier using this industry standard.

View Documentation

Oracle Call Interface (OCI) Support for String Indexed PL/SQL Associative Arrays

PL/SQL string indexed associative arrays are now supported by Oracle Call Interface (OCI). Applications can natively pass these associative arrays between the database and the client application allowing for creating, binding, and manipulating of this collection type.

This feature allows for more straightforward and less error-prone code development.

View Documentation

Result Cache Integrity Mode

Oracle Result Cache allows the caching of query results in memory to improve the performance of frequently executed queries. Queries are cached optimistically based on the setting of `result_cache_mode` or explicit hinting, which considers objects that are not explicitly declared as deterministic for query caching.

Controlling the result cache integrity mode enables customers to enforce the requirement of declaring objects as deterministic before being considered for result caching.

Providing the capability to enforce the requirement of explicitly deterministic objects for query caching improves code quality and rules out the chance of accidentally caching objects that should not be cached.

View Documentation

SQL*Plus ARGUMENT Command

A new `ARGUMENT` command lets users of batch scripts control how SQL*Plus treats script argument variables for which the users have not explicitly set values. With this command, users are now able to control when to prompt for input or use a default value for each unset script argument.

This feature gives SQL script processing more resiliency and flexibility, allowing script actions to be customized by users if they want to alter parameter values.

View Documentation

SQL*Plus CONFIG Command

This command reads the default tnsnames.ora file and generates a JSON file suitable for uploading to a Centralized Configuration Provider.

This command makes it easier to migrate away from tnsnames.ora files and allows connection strings to be stored centrally.

View Documentation

SQL*Plus OERR Command and Improved HELP Syntax

A new `OERR` command in SQL*Plus allows users to see Oracle error message Cause and Action text within SQL*Plus for a user-supplied error number. The existing `HELP` command has also been enhanced to show the same text.

This feature allows developers to immediately get more information about error messages.

View Documentation

SQL*Plus PING Command and Command Line Option

A new SQL*Plus `PING` command and equivalent command line option can be used to show the round-trip time from SQL*Plus to either the network listener or to the database.

The network listener check is equivalent to the traditional `tnsping` command line utility that administrators use to check basic network connectivity. The option to check the database round-trip time is commonly used as a liveness check to ensure that the database itself is reachable.

This feature gives users of SQL*Plus power to verify basic connectivity, which is useful in many troubleshooting or post-install scenarios.

View Documentation

SQL*Plus SET ERRORDETAILS Command

A new `SET ERRORDETAILS` command lets users decide whether additional information should be displayed when Oracle errors are generated in failure scenarios. Additional information that can be displayed is the error help URL and the message Cause and Action text.

This feature improves the developer experience by allowing faster troubleshooting.

View Documentation

SQL*Plus SHOW CONNECTION Command

This command can be used to show details about the current connection, list Oracle Net Service names present in the `tnsnames.ora` file, and resolve a given net service name to a connection string.

Knowing more about connection strings enables users to connect to Oracle Database more easily, and aids troubleshooting connection issues.

View Documentation

Session Exit on Invalidation

Set `SESSION_EXIT_ON_PACKAGE_STATE_ERROR` to true to force a hard session exit when a session's state has been invalidated.

Exiting sessions after state invalidation avoids errors that can occur when applications mishandle invalid state.

View Documentation

Unicode IVS (Ideographic Variation Sequence) Support

The new UCA1210_JAPANESE_IVS collation allows the processing of Unicode Ideographic Variation Sequence (IVS) in Japanese text. The SQL functions `LENGTHC()`, `SUBSTRC()`, `INSTRC()`, and `LIKEC()` are also enhanced to count IVSs as single complete characters.

This feature enables application developers to build applications supporting Unicode IVS. It is an important requirement for markets, such as Japan, where processing of

data including names such as person names, place names, and historic texts often need to support ideographic characters represented in Unicode IVS.

View Documentation

Java

Java in the Database: JDK 11 Support Including Modules

In this release, the Oracle JVM infrastructure has been re-architected to support JDK 11 capabilities including the Java module system.

This feature fosters productivity through the design or reuse and execution of code and libraries based on Java 11, inside the database.

View Documentation

Java in the Database: Web Services Callout Enhancement

This feature furnishes an enhanced implementation of the Web Services Call-Out Utility. Java, PL/SQL, and SQL can now perform a more efficient Web Services Callout.

This feature fosters extensibility and productivity by allowing the Oracle database to invoke external Web Services.

View Documentation

Java in the Database: HTTP and TCP Access While Disabling Other OS Calls

Oracle JVM now offers a more flexible Lockdown Profile configuration for on-premises and cloud database services (for example, the Autonomous Database). HTTP and TCP callouts can now be enabled separately from other OS calls to allow deployments that depend on HTTP and TCP access.

This feature couples extensibility (for example, making HTTP and TCP callouts) with enhanced security for Java code running in the database.

View Documentation

JavaScript

Multilingual Engine JavaScript Modules and Environments

Multilingual Engine (MLE) Modules and Environments allow JavaScript code to persist and be managed in the database. Call specifications provide a means to call JavaScript functions from an MLE module anywhere you can call PL/SQL functions.

The introduction of JavaScript Modules and Environments as schema objects in Oracle Database allows developers to follow established and well-known workflows used in client-side JavaScript development. Complex projects can be broken down into smaller, more manageable pieces worked on independently by team members.

View Documentation

Multilingual Engine Module Calls

Multilingual Engine (MLE) Module Calls allow developers to invoke JavaScript functions stored in modules from SQL and PL/SQL. Call Specifications written in PL/SQL link JavaScript to PL/SQL code units.

Thanks to Module Calls, developers can use JavaScript functions anywhere PL/SQL functions are called.

View Documentation

Multilingual Engine Post-Execution Debugging

Oracle Multilingual Engine (MLE) allows developers to debug their JavaScript code by conveniently and efficiently collecting runtime state while the program is being processed, a method referred to as post-execution debugging. After the code has finished running, the collected data can be used to analyze program behavior, discover, and fix bugs.

Post-execution debugging offers a convenient way to extract runtime state information from a JavaScript code unit at runtime without having to change the observed code.

View Documentation

Multilingual Engine JavaScript SODA API

Simple Oracle Document Access (SODA) is a set of NoSQL-style APIs that let you create and store collections of documents (in particular JSON) in Oracle Database, retrieve them, and query them, without needing to know SQL or how the documents are stored in the database. With the introduction of MLE, JavaScript support for SODA documents exists for client-side and server-side development.

Supporting the Simple Oracle Document Access (SODA) API in JavaScript gives developers a choice between using JSON in a relational or No-SQL way, simplifying the development process and improving the portability of code.

View Documentation

Multilingual Engine JavaScript Support for JSON Data Type

Support for JavaScript Object Notation (JSON) is an integral part of Oracle database. Oracle supports JSON natively with relational database features, including transactions, indexing, declarative querying, and views. A rich set of SQL functions is available to manipulate JSON in a relational model. Oracle Multilingual Engine (MLE) fully supports JSON: both dynamic MLE as well as MLE Module Calls support interactions with the JSON data type.

JSON and JavaScript objects are closely related, forming a natural match in such a way that makes working with JSON very easy with JavaScript code.

View Documentation

Application Connectivity

Reset Database Session State

The reset database session state feature clears the session state set by the application when the request ends. The `RESET_STATE` database service attribute cleans up dirty sessions so that the applications cannot see the state of these sessions. This feature applies to all applications that connect to the database using database services.

This feature uses the `RESET_STATE` attribute on the database service to direct the database to clean the session state at the end of each request so that developers do not have to clean the session state manually. By using this feature, you ensure that there are no data leaks from a previous session.

View Documentation

Implicit Connection Pooling for Database Resident Connection Pooling (DRCP)

This feature enables the automatic assignment of DRCP servers to and from an application connection at runtime when the application starts and finishes database operations, even if the application does not explicitly close the connection.

This feature can provide better scalability and efficient usage of database resources for applications that do not use application connection pooling.

View Documentation

Implicit Connection Pooling for Oracle Connection Manager in Traffic Director Mode (CMAN-TDM)

Client applications that do not use an application connection pool can take advantage of CMAN-TDM Proxy Resident Connection Pooling (PRCP) without making any application changes.

The new feature enables the automatic assignment of PRCP servers to and from an application connection at runtime when the application starts and finishes database operations even if the application does not explicitly close the connection.

This feature can reduce the size of PRCP pools required. It provides better scalability and efficient usage of resources for applications that do not use Oracle Session Pooling or Universal Connection Pooling (UCP).

View Documentation

Improved Oracle Connection Manager in Traffic Director Mode (CMAN-TDM) Pool Configuration Settings for Autonomous Database

Oracle Connection Manager in Traffic Director Mode (CMAN-TDM) has new Proxy Resident Connection Pooling (PRCP) configuration settings for use with Autonomous Database. Per-PDB PRCP pools can be enabled, allowing you to consolidate connection pools for each PDB and share these sessions across multiple services that belong to the same PDB. The maximum PRCP pool size can be dynamically configured based on the new `cman.ora` parameter `TDM_PERPDB_PRCP_CONNFACTOR` and the Oracle Compute Unit (OCPU) count allocated to each PDB.

The per-PDB PRCP mode provides efficient usage of database resources by reducing the number of pools in a CMAN-TDM gateway. Pool sizing can also now be more autonomous, reducing the need for manual re-configuration.

View Documentation

JDBC Enhancements to Transparent Application Continuity

This feature allows, when the RDBMS server supports it, templates (for example, stable restorable attributes), which are cross-session (one template might be used by multiple sessions). This feature also brings the ability to avoid the combinatorial explosion of templates by quarantining session states that are different in most sessions and therefore cannot be shared.

This feature simplifies high availability by moving most application continuity configurations to the server-side. Java applications inherit transparently (that is, no code required) the latest server-side enhancements.

View Documentation

JDBC Extensions for Apps Configuration Providers

JDBC instrumentalization for securely pulling Java Apps configuration from central stores such as Azure Config Store or OCI Object store or any JSON file accessible from generic web servers.

This feature simplifies Java application configuration in multi-Cloud environments.

View Documentation

JDBC Support for Kerberos Authentication using JAAS Configuration

By default, the Oracle JDBC Thin driver uses the default Kerberos login module, bundled with Oracle JDK (`com.sun.security.auth.module.Krb5LoginModule`). This feature enables applications that want to override the default behavior to specify a JAAS configuration file through the connection properties.

This feature provides flexibility with Kerberos Authentication configuration.

View Documentation

JDBC Support for Kerberos Authentication using User and Password Properties

This feature enables the users to configure Kerberos Principal and Password through the User and Password properties. The JDBC Thin driver takes care of initializing the `KerberosLoginModule` on behalf of the applications.

This feature simplifies Kerberos Authentication configuration.

View Documentation

JDBC Support for OAuth 2.0 Including OCI IAM and Azure AD

The Oracle JDBC driver provides support for OAuth 2.0 authentication for Oracle Cloud Infrastructure (OCI) Identity and Access Management (IAM) Cloud Service or the Azure Active Directory.

This feature simplifies Java application authentication to the Oracle Autonomous Database using OAuth 2.0 in multi-Cloud environments (OCI, Azure) in lieu of traditional credentials mechanisms such as username/password or strong authentication mechanisms, such as Kerberos or Radius.

View Documentation

Java Support for True Cache

The `Connection.setReadOnly` and `Connection.isReadyOnly` methods have been enhanced to transparently support True Cache. Developers simply need to set the new connection and system property `oracle.jdbc.useADCDriverConnection` to `true`.

JDBC support for True Cache furnishes mission-critical availability of Oracle database to Java applications. It eliminates single points of failure and prevents data loss and downtime.

View Documentation

Multiple Named Pools for Database Resident Connection Pooling (DRCP)

Database Resident Connection Pooling (DRCP) now supports multiple named pools. New `DBMS_CONNECTION_POOL.ADD_POOL()` and `DBMS_CONNECTION_POOL.REMOVE_POOL()` procedures are added. Oracle Net connection string syntax is enhanced so a pool name can be specified for each connection. Existing procedures can be used to start,

stop, or configure the named pools. Existing `GV$` and `V$` views show the appropriate pool name(s) in use.

Having multiple pools allows finer control on the DRCP pool usage. It helps prevent situations where some applications dominate the use of a single pool.

View Documentation

ODP.NET Transparent Application Failover

Oracle Transparent Application Failover (TAF) is a high availability feature that enables client apps to automatically reconnect to a secondary database instance if the connected primary instance fails or shuts down. ODP.NET Core and managed ODP.NET now support connection and basic session state TAF.

ODP.NET TAF enables apps to recover and continue operating when database downtime occurs. It requires no changes to .NET application code to use.

View Documentation

ODP.NET: Application Continuity

ODP.NET Core and managed drivers now support Application Continuity (AC) and Transparent Application Continuity (TAC). AC and TAC mask outages from end users and applications by recovering the in-flight database sessions following recoverable outages, including transactions. The recovery is transparent such that the end user merely experiences a slightly delayed execution, but no perceptible outage nor error.

AC and TAC improve the user experience for both unplanned outages and planned maintenance. They enhance the fault tolerance of systems and .NET applications that use an Oracle database. Developers can use AC and TAC with existing .NET apps without making any code changes.

View Documentation

ODP.NET: Pipelining

ODP.NET core and managed drivers support pipelining for its database communication. It allows subsequent database requests to be sent and queued transparently even while ODP.NET awaits a database response.

Pipelining improves overall app performance and allows database resources to be used more effectively. ODP.NET does not need to wait for the database to respond from previous requests before submitting subsequent requests.

View Documentation

Oracle Call Interface (OCI) Pipelined Operations

Oracle Call Interface (OCI) has been enhanced to support pipelining of operations. Pipelining enables applications to submit multiple database operations without waiting for a response from the server. The application has control over when the responses to the pipelined operations are harvested. This allows applications to continue work without being blocked while the database is generating results.

This feature is used to increase the overall throughput and responsiveness of applications and languages that use OCI. Pipelining reduces the server and client idle times in comparison with the traditional request-response model.

View Documentation

Oracle Call Interface (OCI) Session Pool Statistics

The Oracle Call Interface (OCI) session pool usage statistics can be viewed.

The statistics help in tuning pool sizes for better performance, and aid in understanding the life cycle of connections.

View Documentation

Oracle Connection Manager in Traffic Director Mode (CMAN-TDM) Support for Direct Path Applications

Oracle Database's Set Current Schema and Direct Path API features are now supported by Oracle Connection Manager in Traffic Director Mode (CMAN-TDM).

This feature enables more client applications to leverage CMAN-TDM connection multiplexing capabilities.

View Documentation

Oracle Connection Manager in Traffic Director Mode (CMAN-TDM) Usage Statistics

A new `V$TDM_STATS` view can be used to query usage statistics for CMAN-TDM per-PDB Proxy Resident Connection Pools (PRCP), such as the number of active client connections in the connection pool, the number of busy and free server connections, the maximum number of connections reached, and more.

Providing usage statistics helps to improve the monitoring and tuning of CMAN-TDM.

View Documentation

Resumable Cursors

Resumable cursors, those that span transactions, will be replayable with Transparent Application Continuity. Such cursors are common in batch processing (such as loading sets of records) and require special handling to reposition those cursors during replay with Transparent Application Continuity.

Broadened support with Transparent Application Continuity for applications that rely upon resumable cursors, those that span commits. These cursors are very common in repetitive batch operations, looping through sets of records for updates and inserts with a commit for each set of records. Now, TAC will be able to replay the transactions that were interrupted (and not yet committed).

View Documentation

Shut Down Connection Draining for Database Resident Connection Pooling (DRCP)

A new, optional `DRAINTIME` argument to `DBMS_CONNECTION_POOL.STOP_POOL()` allows active DRCP pools to be closed after a specified connection drain time, or be closed immediately without waiting for connections to be idle.

This feature gives DBAs better control over DRCP usage and configuration.

View Documentation

UCP Support for XA Transactions with Sharded Databases

This feature allows sharded database connections to participate in eXtended Architecture (XA) transactions managed by WebLogic Server Transaction Manager.

This feature allows reliable XA transactions coupled with the scalability of sharded databases.

View Documentation

Database Drivers API Enhancements

Easy Connect Plus Support for LDAPS/LDAP

Oracle supports LDAP based name look-up for retrieving Database connection strings from the directory servers. The directory used can be OID, OUD, or AD.

Now, LDAP-based name lookup is possible without having `ldap.ora` and `sqlnet.ora`. The values that are specified as part of `ldap.ora` and `sqlnet.ora` for ldap name lookup, are passed in the URL string. If `ldap.ora` or `sqlnet.ora` is present and the ldap URL is passed, then the preference is given to the URL string.

For
example: `ldap[s]://host[:port]/name[,context]?[parameter=value{&parameter=value}]`

Easy Connect Plus extends its support beyond TCP and TCPS to make it easy to use LDAP and LDAPS protocol and parameters.

View Documentation

Enhanced UCP Connection Borrow

Connection creation using the user thread, in the context of a borrow request, can take longer than the specified connectionWaitTimeout (CWT). If a connection has been released by another thread in the meantime, the connection creation request keeps waiting for the operation to complete. It is therefore more effective to borrow the released connection rather than waiting for the one being created.

This feature brings performance enhancement to Java applications during connection borrow.

View Documentation

JDBC Connection Property sendBooleanAsNativeBoolean

A new Connection property oracle.jdbc.sendBooleanAsNativeBoolean is added to restore the old behavior of the Boolean data type, which is used to take integer (0 or 1)

for a Boolean data type.
When set to false (the default is true), this property will restore the old behavior of sending integer values (0 or 1) for the boolean data type.

This feature brings compatibility to Java applications that rely on the old behavior of the boolean data type.

The feature furnishes backward compatibility with the earlier JDBC driver behavior. This feature simplifies upgrading to the latest JDBC driver without breaking the behavior of existing Java applications.

View Documentation

JDBC Support for Database Annotation

Annotation is a mechanism to store application metadata centrally in the database. Annotations can be specified at creation time (CREATE) or at modification time (ALTER). An individual annotation has a name and an optional value. Both the name and the value are freeform text fields. A schema object can have multiple annotations. JDBC furnishes the `getAnnotations()` method with two signatures (as illustrated below). It returns the annotation associated with the specified table or view. It returns `null` if there is no annotation for the given object.

```
getAnnotations?(java.lang.String objectName, java.lang.String domainName,
java.lang.String domainOwner) throws java.sql.SQLException

getAnnotations?(java.lang.String objectName, java.lang.String columnName,
java.lang.String domainName, java.lang.String domainOwner) throws
java.sql.SQLException
```

This feature enables sharing metadata across applications and microservices thereby increasing metadata management and productivity.

View Documentation

JDBC Support for Pipelined Database Operations

In the previous releases, the JDBC driver would not allow another database call to start until the current call had been completed however, with asynchronous and reactive programming, Java applications could perform non-database operations, in the meantime. In this release, the database server and the Oracle JDBC-Thin both support pipelining database operations. Java applications can now asynchronously submit several SQL requests to the server without waiting for the return of the preceding calls.

The combination of Java reactive and asynchronous programming (JDBC Reactive Extension, Reactive Streams (R2DB and Virtual Threads) with database support for pipelining fosters high throughput.

View Documentation

JDBC Support for SQL BOOLEAN Data Type

This feature exposes the Oracle RDBMS `BOOLEAN` data type to Java through a new `BOOLEAN` data type in `oracle.jdbc.OracleType Enum`, and `DatabaseMetadata`. This feature also performs the implicit conversion of character and number data types to `BOOLEAN` data types.

Java applications can take advantage of the new JDBC support for the standard JDBC `BOOLEAN` data type. The benefits include: increased portability and the ease of development fostered by the implicit conversion of character and number to `BOOLEAN`.

View Documentation

JDBC Support for Self-Driven Diagnosability

This feature eliminates the need to switch from the non-logging JAR files (for example, `ojdbcXX.jar`) to the debug JAR files (for example, `ojdbcXX_g.jar`) for logging purposes. In addition, it enables logging in the following three ways: logging per connection, logging at the tenant level, or logging globally.

This feature furnishes increased productivity and ease of use for Java applications. It greatly simplifies the debugging of Java applications by removing the need to switch from the production jars to the debug jars.

View Documentation

ODBC Support for SQL BOOLEAN Data Type

ODBC now supports the new SQL `BOOLEAN` data type.

Using the SQL `BOOLEAN` data type enables applications to represent the state more clearly.

View Documentation

Oracle Call Interface (OCI) Support for SQL BOOLEAN Data Type

Oracle Call Interface (OCI) now supports querying and binding of the new SQL `BOOLEAN` data type.

Using the SQL `BOOLEAN` data type enables applications to represent state more clearly.

View Documentation

Precompiler Support for SQL BOOLEAN Data Type

The Pro*C and Pro*COBOL precompilers now support querying and binding of the new SQL `BOOLEAN` data type.

Using the new data type makes it easier to represent boolean state in applications instead of using a character column to indicate Y or N.

View Documentation

UCP Asynchronous Extension

Universal connection pool (UCP) is extended with asynchronous (reactive) database calls.

This extension furnishes high scalability and throughput to Java applications.

View Documentation

UCP Support for Self-Driven Diagnosability

The new universal connection pool (UCP) diagnosability feature provides the following capabilities:

- When logging is enabled (it is disabled by default), log records are written into an in-memory ring buffer.
- Tracing is enabled by default. A tracing event dumps the ring buffer into either a data-source-specific buffer or a common buffer.

This feature fosters productivity (for example, real-time debugging) and ease of use for Java applications using the UCP.

View Documentation