4 Data Analytics

This section describes the new data analytics features.

General

Hybrid Partitioned Tables with Interval and Auto-List Partitioning

You can create Hybrid Partitioned Tables using single-level partitioning with interval and automatic list partitioning. This is in addition to existing support for single-level partitioning and range and list partitioning.

These extensions to Hybrid Partitioned Tables in Oracle Database provide a user-friendly partitioning strategy.

View Documentation

Data Quality Operators in Oracle Database

This release introduces the following two new string matching operators based on approximate or "fuzzy" string matching.

- `PHONIC_ENCODE` converts words or phrases into language-specific codes based on pronunciation.
- `FUZZY_MATCH`, which is language-neutral, gauges the textual similarity between two strings.

The new phonic encoding and fuzzy matching methods enable more sophisticated matching algorithms to be run directly on data in the database rather than only in external applications, providing improved matching performance and efficiency, for example in data de-duplication, linking or enhancement.

View Documentation

Automatic Data Clustering

Oracle Database automatically and transparently clusters storage-based data in response to the type of queries used by the application workload. This allows the workload to make more efficient use of data access optimizations, such as storage indexes, zone maps, and join zone maps.

This feature significantly improves performance for data warehousing workloads based on zone maps or storage indexes. Once data is clustered, the performance of data-scanning queries improves because larger contiguous areas (or zones) of storage are pruned or skipped when they do not contain the data being matched by a particular query.

View Documentation

Extended Support and Faster Performance for JSON Materialized Views

Materialized views of JSON tables have been enhanced with the ability to fast refresh more types of Materialized Views of JSON tables as well as Query Rewrite support for these Materialized Views.

The performance for JSON table Materialized Views is significantly improved through better fast refresh capabilities and better query rewrite capabilities for more workloads. You can use JSON table Materialized Views more broadly in your applications, with better performance and less resource utilization.

View Documentation

Oracle SQL Access to Kafka

Oracle SQL Access to Kafka (DBMS_KAFKA) provides efficient, reliable, and scalable access to data streams from Apache Kafka and OCI Streaming Service. Streaming data can be queried via SQL or loaded into Oracle database tables.

Oracle Database provides efficient, reliable, and scalable integration with Apache Kafka using the `DBMS_KAFKA` APIs. This API enables Oracle Database to consume data from external data streams without the need for costly, complex direct application connections using proprietary interfaces. Oracle SQL Access to Kafka enables you to use Oracle Databases rich analytic capabilities across all your data.

View Documentation

SQL

Text Indexes with Automatic Maintenance

You can specify a new automatic maintenance mode for text indexes using the `MAINTENANCE AUTO` index parameter. This method automates the `CTX_DDL.SYNC_INDEX` operation. This is now the default synchronization method for new indexes.

With this method, newly created text indexes do not require you to specify a synchronization interval or manually run a `SYNC_INDEX` operation. A background process automatically performs these tasks without user intervention. This helps in synchronizing a large number of indexes in an optimal manner, and also eliminates the manual or time-based `SYNC` operations. By using a background job rather than the database scheduler, it avoids scheduling conflicts and the risk of running out of available jobs. Overall it makes for simpler, more resilient applications and better utilization of hardware resources.

View Documentation

Transportable Binary XML

Transportable binary XML (TBX) is a new self-contained XMLType storage method. TBX supports sharding, XML search index, and Exadata pushdown operations, providing better performance and scalability than other XML storage options.

You can migrate existing XMLType storage of a different format to TBX format in any of these ways:

- Insert-as select or create-as-select
- Online Redefinition
- Data Pump

Transportable binary XML (TBX) provides better performance and scalability. With the support of more database architectures, such as sharding or Exadata, and its capability to easily migrate and exchange XML data among different servers, containers, and PDBs, TBX allows your applications to take full advantage of both this new XML storage format on more platforms and architectures.

View Documentation

Concurrent Materialized View Refresh for on-commit

Materialized view refresh provides concurrent refresh, where multiple sessions can refresh the same on-commit materialized views simultaneously without the need for serialization.

Concurrent refresh broadens the applicability of materialized views for your applications and helps make application development simpler. It provides faster refresh and more up-to-date materialized views.

View Documentation

Enhanced Automatic Indexing

Indexes incur a maintenance overhead during DML operations. This can work against their improvements to data access performance. The enhancements to Automatic Indexing take a broader view than in previous releases and account for index maintenance costs when deciding which indexes will benefit the workload as a whole. Columns filtered using range predicates are considered for indexes and function-based indexes are now supported. This further increases the scope of Automatic Indexing effectiveness.

Automatic Indexing better assesses the impact of DML operations in your database when choosing automatic indexing. Your performance benefits by determining the overall advantage of an index to your workload.

View Documentation

Enhanced Automatic Materialized Views

Automatic materialized views have been enhanced to include automatic partitioning. In addition, there is a more accurate internal cost model for automatic materialized view selection, which considers both access benefits and maintenance (refresh) costs, as well as the frequency of execution.

Rewrite capabilities have been broadened, including outer join queries with filter predicates.

Enhancing Automatic Materialized Views with more accurate cost-benefit analysis and broader usability optimizes the management of your materialized view eco system and improves the overall performance of your system.

View Documentation

Enhanced Automatic SQL Plan Management

Automatic SQL plan management has been enhanced to detect and repair SQL performance regressions more quickly. SQL plan changes are detected at parse-time and, after initial execution, SQL performance is compared with the performance of previous SQL execution plans. If a performance degradation is detected, the plan is repaired accordingly.

With automatic SQL Plan Management, your application service levels improves, and impacts caused by SQL performance (plan) regressions are minimized and addressed transparently and proactively.

View Documentation

Enhanced LOB Support for Distributed and Sharded Environments

Distributed LOBs are LOBs that are fetched from one server to another and may optionally be returned to the client. Shared LOBS are an extension of distributed LOBs where LOBs are transported between shards or between a shard and the shard coordinator. In previous versions, support for sharded and distributed LOBs were limited to persistent LOBs, and temporary LOBs only where they originate from JSON operations. Now all temporary LOBs (including Value LOBs) and new increased-length inline LOBs are usable as distributed and sharded LOBs.

You can now work with inline LOBs, value LOBs, and all temporary LOBs in distributed and sharded environments.

You experience improved performance, scalability, and garbage collection when you work with temporary LOBs, thus improving your developer productivity and application resilience.

View Documentation

Enhanced Parallel Processing Resources Management

Parallel processes are released pro-actively before individual statements using parallelism are finished. For example, an uncommitted parallel DML operation or a partially fetched parallel `SELECT` statement with 2 Parallel Server Sets (Producer-Consumer) will release one of the Parallel Server Sets as soon as it has finished working, freeing half of the parallel process for use of other statements.

Releasing parallel processes as early as possible and making them usable for other statements optimizes the utilization of your available resources, improving the overall performance of your systems and applications.

View Documentation

Increased Maximum Size of Inline LOBs of 8000 Bytes

LOB values are stored either in the table row (inline) or outside of the table row (out-of-line). The maximum size of the inline LOB is increased to 8000 bytes, allowing larger LOB values being stored inside a row. Earlier, the maximum size was 4000.

This provides better input-output performance while processing LOB columns. You can experience the improved performance while running operations, such as full table scans, range scans, and DML.

View Documentation

Materialized View Support for ANSI Joins

Materialized Views in Oracle Database support full rewrite capabilities for SQL statements using ANSI join syntax and for Materialized View definitions using ANSI join syntax.

Full support of ANSI joins with materialized view rewrite provides a significant performance improvement. Many queries, particularly ones generated by SQL Tools and Reports, often use ANSI join syntax. This enhancement allows such tools to benefit from materialized views for query rewrite regardless of the syntax used by joins.

View Documentation

Read-Only Value LOBs

Value LOBs, a read-only subset of Temporary LOBs, are valid for a SQL fetch duration and optimize the reading of LOB values in the context of a SQL query. Many applications use LOBs to store medium-sized objects, about a few megabytes in size, and you want to read the LOB value in the context of a SQL query.

Value LOBs provide faster read performance and get automatically freed when the next fetch for a cursor is performed, preventing the accumulation of temporary LOBs and simplifying the LOB management within your application.

Value LOBs provide faster read performance than classical reference LOBs for your workload and don't need specific LOB management in your application. Using Value LOBs improves your application performance and makes implementing applications with LOBs simpler and more manageable.

View Documentation

Semi-Join Materialized Views

Semi-Join Materialized View Rewrite is a unique form of query rewrite. A single, large unified dimension table in the query is replaced with one or more join-specific materialized views. In a unified dimension data model, where multiple dimension tables are merged into a single large dimension table, semi-join Materialized Views materialize one or more of the joins of such a single, large unified dimension table with the fact table.

This new type of Materialized View significantly improves the runtime and resource consumption for complex analytical operations. Semi-join Materialized Views are especially beneficial when the number of applicable dimension keys derived from the large unified dimension table (through semi-join) is small.

View Documentation

Ubiquitous Search With DBMS_SEARCH Packages

The new `DBMS_SEARCH PL/SQL` package allows the indexing of multiple schema objects in a single index. You can add a set of tables, external tables, or views as data sources into this index. All the columns in the specified sources are indexed and available for a full-text search.

With a simplified set of `DBMS_SEARCH` APIs, you can create indexes across multiple objects, add or remove data sources, and perform a full-text search within a single data source or across multiple sources using the same index.

This simplifies indexing tasks that were previously performed using the `USER_DATASTORE` procedures, thus enhancing developer productivity.

View Documentation

In-Memory

Automatic In-Memory Enhancements for Improving Column Store Performance

Automatic In-Memory (AIM) has been enhanced to automatically enable creation and removal of Database In-Memory performance features based on an enhanced workload analysis algorithm. These features include Join Groups, caching of hashed dictionary values for join key columns and In-Memory Optimized Arithmetic.

Automatic In-Memory (AIM) has been enhanced to identify and enable or disable Database In-Memory features that can improve performance. It enables features either selectively or globally, depending on which adds the most benefit. This improves application performance and also conserves space in the In-Memory column store without requiring manual intervention.

View Documentation

Automatic In-Memory Sizing for Autonomous Databases

The In-Memory column store will now automatically grow and shrink dynamically based on workload. This allows the In-Memory column store to be available on Autonomous Database. Exadata scan performance is further improved for objects that are partially populated.

With Automatic In-Memory sizing, there is no longer a need to manually resize the In-Memory column store to accommodate different database workloads. This reduces the administrative effort of enabling Database In-Memory. Automatic In-Memory sizing also allows the In-Memory column store to be enabled on Autonomous Database (ADB), enabling applications running on ADB to also take advantage of faster analytic query performance.

View Documentation

In-Memory Optimized Dates

To enhance the performance of DATE-based queries DATE components (i.e. DAY, MONTH, YEAR) can be extracted and populated in the IM column store leveraging the In-Memory Expressions framework.

This enhancement enables faster query processing on DATE columns which can significantly improve the performance of date based analytic queries.

View Documentation

In-Memory RAC-Level Global Dictionary

Database In-Memory Join Groups now support Global Dictionaries across RAC nodes. With Join Groups a common dictionary is shared by columns that are joined together. In-Memory RAC-level global dictionaries now synchronize these common dictionaries across nodes within a RAC database.

In a RAC environment, this feature further improves database In-Memory performance for distributed hash joins.

View Documentation

Selective In-Memory Columns

With Selective In-Memory columns, it is now easier to add or exclude columns for in-memory. An `ALL` sub-clause has been added so that all columns can be either enabled or disabled from in-memory. This reduces the need to have very long strings of included or excluded columns.

With Selective In-Memory columns, the ability to specify `ALL` columns to be enabled or disabled from in-memory reduces the need for very long strings of columns, which reduces the chance for errors and makes configuring in-memory enabled tables easier.

View Documentation

Vectorized Query Processing: Multi-Level Joins and Aggregations

This feature enhances the In-Memory Deep Vectorization framework by fully exploiting SIMD capabilities to further improve hash join and group by aggregation performance. New optimizations include incorporating multi-level hash join support, full In-Memory group by aggregation support, and support for multi-join key and additional join methods.

This feature adds improvements in the performance of joins and aggregations, which are the foundations for analytic queries. This enables faster real-time analytic performance and requires no application SQL changes. This feature is automatically used when enabled, which is the default.

View Documentation

Machine Learning - Enhancements

Automated Time Series Model Search

This feature enables the Exponential Smoothing algorithm to select the forecasting model type automatically - as well as related hyperparameters - when you do not specify the `EXSM_MODEL` setting. This can lead to more accurate forecasting models.

This feature automates the Exponential Smoothing algorithm hyperparameter search to produce better forecasting models without manual or exhaustive search. It enables non-expert users to perform time series forecasting without detailed understanding of algorithm hyperparameters while also increasing data scientist productivity.

View Documentation

Explicit Semantic Analysis Support for Dense Projection with Embeddings in OML4SQL

The unstructured text analytics algorithm Explicit Semantic Analysis (ESA) is able to output dense projections with embeddings, which are functionally equivalent to the popular doc2vec (document to vector) representation.

Producing a doc2vec representation is useful as input to other machine learning techniques, for example, classification and regression, to improve their accuracy when used solely with text or in combination with other structured data. Use cases include processing unstructured text from call center representative notes on customers or physician notes on patients along with other customer or patient structured data to improve prediction outcomes.

View Documentation

GLM Link Functions

The in-database Generalized Linear Model (GLM) algorithm now supports additional link functions for logistic regression: probit, cloglog, and cauchit.

These additional link functions expand the set available to match standard Generalized Linear Model (GLM) implementations. They enable increasing model quality, for example, accuracy, by handling a broader range of target column data distributions and expand the class of data sets handled. Specifically, the probit link function supports binary (for example, yes/no) target variables, such as when predicting win/lose, churn/no-churn, buy/no-buy. The asymmetric link function complementary-log-log (cloglog) supports binary target variables where one outcome is relatively rare, such as when predicting time-to-relapse of medical conditions. The cauchit link function supports handling data with, for example, data recording errors, more robustly.

View Documentation

Improved Data Prep for High Cardinality Categorical Features

This feature introduces the setting `ODMS_EXPLOSION_MIN_SUPP` to allow more efficient, data-driven encoding for high cardinality categorical columns. You can adjust the threshold (define the minimum support required) for the categorical values in explosion mapping or disable the feature, as needed.

This feature introduces a more efficient, data-driven encoding of high cardinality categorical columns, allowing users to build models without manual data preparation of such columns.

It efficiently addresses large datasets with millions of categorical values by recoding categorical values to include only those with sufficient support, enabling you to overcome memory limitations.

View Documentation

Lineage: Data Query Persisted with Model

This feature enables users to identify the data query that was used to provide the training data for producing a model. The `BUILD_SOURCE` column in the `ALL/DBA/USER_MINING_MODELS` view enables users to access the data query used to produce the model.

This feature records the query string that is run to specify the build data, within the model's metadata to better support the machine learning lifecycle and MLOps.

View Documentation

Multiple Time Series

The Multiple Time Series feature of the Exponential Smoothing algorithm enables conveniently constructing Time Series Regression models, which can include multivariate time series inputs and indicator data like holidays and promotion flags. It enables constructing Time Series Regression models to include multivariate time series inputs and indicator data like holidays and promotion flags.

This feature automates much of what a data scientist would perform manually by generating backcasts and forecasts on one or more input time series, where the target time series also receives confidence bounds. The result is used as input to other ML algorithms, for example, to support time series regression using XGBoost, with multivariate categorical, numeric, and time series variables.

View Documentation

OML4Py and OML4R Algorithm and Data Type Enhancements

The Oracle Machine Learning for Python (OML4Py) API exposes additional in-database machine learning algorithms, specifically Non-negative Matrix Factorization (NMF) for feature extraction, Exponential Smoothing Method (ESM) for time series forecasting, and Extreme Gradient Boosting (XGBoost) for classification and regression. OML4Py introduces support for date, time, and Integer datatypes.

The Oracle Machine Learning for R (OML4R) API exposes additional in-database machine algorithms, specifically Exponential Smoothing Method (ESM) for time series forecasting, Extreme Gradient Boosting (XGBoost) for classification and regression, Random Forest for classification, and Neural Network for classification and regression.

The enhancements to OML4R and OML4Py further enable Oracle Database as a platform for data science and machine learning, providing some of the most popular in-database algorithms from Python and R.

The additional in-database algorithms enable use cases such as demand forecasting using ESM, churn prediction and response modeling using Random Forest, and generating themes from document collections using NMF. As a feature extraction algorithm, NMF supports dimensionality reduction and as a data preparation step prior to modeling using other algorithms. XGBoost is a popular classification and regression algorithm due to its high predictive accuracy and also supports the machine learning technique survival analysis. Random Forest is a popular classification algorithm due to its high predictive accuracy. Neural Network is a classification and regression algorithm that is well-suited to data with noisy and complex patterns, such as found in sensor data, and provides fast scoring.

The OML4Py support for date, time, and integer data types enables operating on database tables and views that contain those data types, for example, to transform and prepare data at scale in the database.

View Documentation

Outlier Detection using Expectation Maximization (EM) Clustering

The Expectation Maximization algorithm is expanded to support distribution-based anomaly detection. The probability of anomaly is used to classify an object as normal

or anomalous. The EM algorithm estimates the probability density of a data record, which is mapped to a probability of an anomaly.

Using Expectation Maximization (EM) for anomaly detection expands the set of algorithms available to support anomaly detection use cases, like fraud detection. Since different algorithms are capable of identifying patterns in data differently, having multiple algorithms available is beneficial when addressing machine learning use cases.

View Documentation

Partitioned Model Performance Improvement

This feature improves the performance for a high number of partitions (up to 32K component models) in a partitioned model and speeds up the dropping of individual models within a partitioned model.

Machine learning use cases often require building one model per subset of data, e.g., a model per state, region, customer, or piece of equipment. The partitioned models capability already automated the building of such models - providing a single model abstraction for simplified scoring - and this enhancement improves overall performance when using larger number of partitions.

View Documentation

XGBoost Support for Constraints and for Survival Analysis in OML4SQL

The in-database XGBoost algorithm is enahnced to support the machine learning technique survival analysis, as well as feature interaction constraints and monotonic constraints. The constrainsts allow you to choose how variables are allowed to interact.

Survival analysis is an important machine learning technique for multiple industries. This enhancement enables increased model accuracy when predicting, for example, equipment failures and healthcare outcomes. Specifically, this supports data scientists with the Accelerated Failure Time (AFT) model - one of the most used models in survival analysis – to complement the Cox proportional hazards regression model.

Interaction and monotonic constraints provide for greater control over the features used to achieve better predictive accuracy by leveraging user domain knowledge when specifying interaction terms.

View Documentation

Machine Learning - Enhancements for R

Exponential Smoothing Method (ESM) for Time Series Forecasting

Exponential Smoothing is a moving average method with a single parameter which models an exponentially decreasing effect of past levels on future values. This in-database algorithm is exposed through the R API of Oracle Machine Learning for R.

Exponential Smoothing Methods have been widely used in forecasting for over half a century. It has applications at the strategic, tactical, and operation level. Being exposed as part of the R API, you have native R access to this in-database algorithm.

View Documentation

In-Database Neural Network Algorithm in OML4R

The in-database Neural Network algorithm allows you to address classification and regression use cases.

Neural networks are well-suited to data with noisy and complex patterns, such as found in sensor data, and provide fast scoring. Being exposed as part of the R API, you have native R access to this in-database algorithm.

View Documentation

In-Database Random Forest for Classification in OML4R

The Random Forest algorithm provides an ensemble learning technique for classification.

Random Forest is a popular classification algorithm due to its high predictive accuracy. You can now use this in-database algorithm through the R API of Oracle Machine Learning for R.

View Documentation

XGBoost Support for Classification and Regression in OML4R

XGBoost is a scalable gradient tree boosting algorithm that supports both classification and regression. The in-database implementation makes available the XGBoost Gradient Boosting open source package.

XGBoost is a popular classification and regression algorithm due to its high predictive accuracy and its support for the machine learning technique survival analysis. Being exposed as part of the R API, you have native R access to this in-database algorithm.

View Documentation

Machine Learning - Enhancements for Python

Exponential Smoothing Method (ESM) for Time Series Forecasting in OML4Py

Exponential Smoothing is a moving average method with a single parameter which models an exponentially decreasing effect of past values. This in-database algorithm is exposed through the Python API of Oracle Machine Learning for Python.

Exponential Smoothing Methods have been widely used in forecasting for over half a century. It has applications at the strategic, tactical, and operational levels. Being exposed as part of the Python API, you have native Python access to this in-database algorithm.

View Documentation

Non-Negative Matrix Factorization Support for Dimensionality Reduction in OML4Py

Non-Negative Matrix Factorization (NMF) is a state-of-the-art feature extraction algorithm. You can now use this in-database algorithm through the Python API of Oracle Machine Learning for Python.

NMF is useful when there are many attributes, and those attributes are ambiguous or have weak predictability. By combining attributes through linear combinations, NMF can produce meaningful patterns, topics, or themes.

View Documentation

Support for Date, Time, and Integer Data Types in OML4Py

OML4Py introduces support for date, time, and Integer data types.

The OML4Py support for date, time, and integer data types enables you to create pandas DataFrame proxy objects and operate on database tables and views that contain those data types. This enables you to explore and prepare data at scale in the database.

View Documentation

XGBoost for Classification and Regression in OML4Py

XGBoost is a scalable gradient tree boosting algorithm that supports both classification and regression. The in-database implementation makes available the XGBoost Gradient Boosting open source package.

XGBoost is a popular classification and regression algorithm due to its high predictive accuracy. Being exposed as part of the Python API, you have native Python access to this in-database algorithm.

View Documentation

Spatial

Spatial: 3D Models and Analytics

The point cloud feature of Oracle Database supports change detection through SQL and PL/SQL APIs.

This feature automates discovery of relevant changes between two point clouds, enabling easy inclusion in applications, such as modeling changes in forest canopies, assessing damages to landscape due to fire, flood, landslides, or earthquakes, and measuring progress over time in infrastructure projects.

View Documentation

Spatial: REST APIs for GeoRaster

Oracle Database includes a comprehensive set of REST APIs for working with GeoRaster data such as satellite imagery.

In addition to existing PL/SQL and Java APIs, developers can use REST APIs to perform GeoRaster query and data manipulation operations. This feature simplifies the development of cloud applications which frequently depend on REST APIs.

View Documentation

PL/SQL API to Generate Spatial Vector Tiles for Map Visualization

Developers can use SQL to generate vector tiles, a dynamic mapping technology, to convert geometries stored in the Oracle Database into vector tiles for efficient map rendering capabilities in web applications.

The utilization of vector tiles in map visualization enables businesses to deliver customizable, efficient, and engaging map experiences.

View Documentation

Workspace Manager: Improved Security when using Oracle Workspace Manager

Database users can have workspace manager objects in their own schema instead of in the WMSYS schema.

Oracle Workspace Manager enables collaborative development, what-if analysis from data updates, and maintains a history of changes to the data. Developers can create multiple workspaces and group different versions of table row values in different workspaces. With this enhancement, developers have improved security when using this feature. All the workspace manager objects can be stored and invoked in their own user schema in Oracle Autonomous Database and user-managed databases.

View Documentation