Proposal for Adding Hologres JDBC Catalog Support

Abstract

Hologres is a real-time data warehouse engine developed by Alibaba Cloud, designed for high-concurrency and low-latency online analytical processing (OLAP) and ad-hoc analysis of petabyte-scale data. Hologres is fully compatible with the PostgreSQL protocol and deeply integrates with the big data ecosystem including MaxCompute, Flink, and DataWorks.

Apache Gravitino (incubating) is a high-performance, geo-distributed, and federated metadata lake. It manages the metadata directly in different sources, types, and regions.

Gravitino currently supports MySQL, PostgreSQL, and Apache Doris metadata management but does not support Hologres yet. This proposal is to add Hologres JDBC catalog support in Gravitino so that users can have more options for managing real-time data warehouse metadata.

Background

Hologres is fully compatible with the PostgreSQL protocol and highly compatible with PostgreSQL syntax. This means that we can connect to Hologres via the PostgreSQL JDBC Driver (version 42.3.2 or later recommended). Although Hologres is compatible with the PostgreSQL protocol, there are still some differences in data types, table properties, indexes, and partitioning strategies. Therefore, we will focus on these differences.

Refer to the following documents:

- Hologres Data Types
- Create Tables in Hologres
- Hologres Index Quick Start

About Open Source License Compatibility Issues

Hologres provides open-source connectors under the Apache-2.0 license:

- alibabacloud-hologres-connectors
- dbt-hologres
- holo-client

Since Hologres uses the PostgreSQL JDBC Driver which is licensed under the BSD-2-Clause license, and the Apache-2.0 license is compatible with BSD-2-Clause, there should be no license compatibility issues.

Additional Notes

Hologres is available as both an Alibaba Cloud managed service and has open-source components. For this proposal, we will focus on the JDBC connectivity aspect, which works with both the cloud service and any PostgreSQL-compatible Hologres deployment. The PostgreSQL JDBC Driver can be used to connect to Hologres instances, enabling complete end-to-end integration testing.

Proposals

Implementation Status

The Hologres JDBC Catalog has been implemented and is located under `catalogs-contrib/catalog-jdbc-hologres`. The implementation includes the following completed features:

Completed: Core Catalog Infrastructure

- **HologresCatalog** (`HologresCatalog.java`): Main catalog class extending `JdbcCatalog`, with `shortName()` returning `"jdbc-hologres"`. Creates all necessary Hologres-specific converters and operations via factory methods (`createExceptionConverter()`, `createJdbcTypeConverter()`, `createJdbcDatabaseOperations()`, `createJdbcTableOperations()`, `createJdbcColumnDefaultValueConverter()`).
- **HologresCatalogOperations** (`HologresCatalogOperations.java`): Extends `JdbcCatalogOperations`, handles PostgreSQL JDBC driver deregistration on `close()` to avoid memory leaks when the catalog is unloaded from IsolatedClassLoader.
- **HologresCatalogCapability** (`HologresCatalogCapability.java`): Implements `Capability`, defines naming rules following PostgreSQL conventions (regex: `^[_a-zA-Z\p{L}/][\w\p{L}-$/=]{0,62}$`) and reserves system schema names (`pg_catalog`, `information_schema`, `hologres`).

Completed: Schema (Database) Operations

- **HologresSchemaOperations** (`HologresSchemaOperations.java`): Extends `JdbcDatabaseOperations`, implements schema CRUD operations.
    - Maps Hologres Database to JDBC Catalog, Hologres Schema to JDBC Schema.
    - Filters out 10 system schemas: `pg_toast`, `pg_catalog`, `information_schema`, `hologres`, `hg_internal`, `hg_recyclebin`, `hologres_object_table`, `hologres_sample`, `hologres_streaming_mv`, `hologres_statistic`.
    - Supports schema comments via `COMMENT ON SCHEMA` syntax.

- Schema existence check via `schemaExists()` using `DatabaseMetaData.getSchemas()`.

Completed: Table Operations

- **HologresTableOperations** (`HologresTableOperations.java`): Extends `JdbcTableOperations`, implements `RequireDatabaseOperation`, provides full table lifecycle management.

    - **Table Type Support**: Lists and loads `TABLE` (regular tables and partition child tables) and `PARTITIONED TABLE` (partition parent tables) types. Excludes `VIEW` and `FOREIGN TABLE` from listing.

    - **CREATE TABLE**: Generates complete DDL with:

        - Column definitions with data type, nullable constraints, and default values
        - `WITH` clause for Hologres-specific properties (orientation, clustering_key, etc.)
        - HASH distribution via `distribution_key` in `WITH` clause
        - PRIMARY KEY constraints
        - Table and column comments via `COMMENT ON TABLE/COLUMN` statements
        - Single quote escaping in comments (e.g., `'It''s a test'`)

    - **Generated Column Support**: Detects `UnparsedExpression` default values and generates `GENERATED ALWAYS AS (expr) STORED` syntax instead of `DEFAULT`. Correctly places the `STORED` keyword before nullable constraints (`NOT NULL` / `NULL`). Distinguishes from auto-increment columns (`GENERATED BY DEFAULT AS IDENTITY`).

    - **Partition Table Support**:

        - **Logical Partition Tables** (Hologres V3.1+): Reads `is_logical_partitioned_table` and `logical_partition_columns` from `hologres.hg_table_properties`. Creates via `LOGICAL PARTITION BY LIST(col1[, col2])` syntax. Supports 1–2 partition columns.
        - **Physical Partition Tables**: Reads partition information from `pg_catalog.pg_partitioned_table` (partition strategy and attribute numbers) and `pg_catalog.pg_attribute` (attribute number to column name resolution). Creates via `PARTITION BY LIST(column)` syntax. Supports exactly 1 partition column.

- **ALTER TABLE**: Supports the following change types:

  - `UpdateComment`: Update table comment with Gravitino ID preservation
  - `AddColumn`: Add column with type and comment only. Setting NOT NULL constraint, default value, and auto-increment are not supported via ALTER TABLE (throws `IllegalArgumentException`).
  - `RenameColumn`: Rename column via `RENAME COLUMN`
  - `UpdateColumnDefaultValue`: Throws `IllegalArgumentException` (Hologres does not support altering column default value via ALTER TABLE)
  - `UpdateColumnType`: Throws `IllegalArgumentException` (Hologres does not support altering column type via ALTER TABLE)
  - `UpdateColumnComment`: Update column comment via `COMMENT ON COLUMN`
  - `DeleteColumn`: Drop column with `IF EXISTS` support
  - `UpdateColumnNullability`: Throws `IllegalArgumentException` (Hologres does not support altering column nullability via ALTER TABLE)
  - `AddIndex`: Throws `IllegalArgumentException` (Hologres does not support adding index via ALTER TABLE)
  - `DeleteIndex`: Throws `IllegalArgumentException` (Hologres does not support deleting index via ALTER TABLE)
  - `UpdateColumnAutoIncrement`: Throws `IllegalArgumentException` (Hologres does not support altering auto-increment via ALTER TABLE)
  - `UpdateColumnPosition`: Throws `IllegalArgumentException` (Hologres does not support column reordering)
  - `SetProperty` / `RemoveProperty`: Throws `IllegalArgumentException` (not yet supported)

- **Table Properties**: Reads from `hologres.hg_table_properties` system table. Handles JDBC-to-DDL property key conversion (e.g., `binlog.level` → `binlog_level`, `binlog.ttl` → `binlog_ttl`). Filters internal properties via `isUserRelevantProperty()` and only exposes user-relevant ones. During CREATE TABLE, filters `distribution_key`, `is_logical_partitioned_table`, and `primary_key` from the `WITH` clause to avoid duplication.

- **Distribution**: Reads distribution key from `hologres.hg_table_properties` and returns `Distributions.hash()`. Writes distribution key via the `WITH` clause.

Validates that only HASH distribution strategy is used and at least one column is specified.

- **Auto-increment**: Not supported. Creating auto-increment columns (e.g., `GENERATED BY DEFAULT AS IDENTITY`) is rejected with `IllegalArgumentException` in both CREATE TABLE and ALTER TABLE ADD COLUMN. Hologres SERIAL/auto-increment tables that already exist can still be loaded and displayed, but cannot be created through Gravitino.

Completed: Type Conversion

- **HologresTypeConverter** (`HologresTypeConverter.java`): Bidirectional type mapping between Hologres and Gravitino types (185 lines).
  - Handles `TIMESTAMP`/`TIMESTAMPTZ` without precision suffix (Hologres does not support precision syntax like `timestamptz(6)`).
  - Supports array types via `_prefix` to `[]` suffix conversion (e.g., `_int4` ↔ `int4[]`).
  - Maps unsupported types (`json`, `jsonb`, `uuid`, etc.) to `ExternalType`.
  - Array elements must be non-nullable (Hologres limitation).
  - Multidimensional arrays are not supported.

- **HologresColumnDefaultValueConverter** (`HologresColumnDefaultValueConverter.java`): Handles PostgreSQL-style default value casting (133 lines).
  - `toGravitino()`: Parses literal default values with type cast patterns (e.g., `'2024-01-15'::date`), handles `NULL::type` cast expressions, returns `CURRENT_TIMESTAMP` for the corresponding PostgreSQL value, and falls back to `UnparsedExpression` for complex expressions (e.g., generated column expressions, `nextval('seq')`).
  - `fromGravitino()`: Overridden to support converting `UnparsedExpression` back to SQL strings (for generated column round-trip), delegates to super for Literal and DEFAULT_VALUE_NOT_SET.

- **HologresExceptionConverter** (`HologresExceptionConverter.java`): Maps PostgreSQL SQLSTATE error codes to Gravitino exceptions (85 lines).
  - `42P04`/`42P06` → `SchemaAlreadyExistsException`
  - `42P07` → `TableAlreadyExistsException`
  - `3D000`/`3F000` → `NoSuchSchemaException`
  - `42P01` → `NoSuchTableException`
  - `08xxx` → `ConnectionFailedException`
  - Password authentication failure → `ConnectionFailedException`

Completed: Configuration and Build

- **Build**: `build.gradle.kts` (112 lines) with dependencies on `catalog-jdbc-common`, `api`, `common`, `core`, PostgreSQL driver for testing, testcontainers support, Guava, Commons Lang3/Collections4, and Log4j.
- **Config**: `jdbc-hologres.conf` with JDBC connection configuration template.
- **Service Registration**: `META-INF/services/org.apache.gravitino.CatalogProvider` for catalog auto-discovery.
- **Distribution**: Build tasks (`copyCatalogLibs`, `copyCatalogConfig`) to package catalog into `distribution/package/catalogs/jdbc-hologres/`.

Completed: Unit Tests

5 test classes with comprehensive coverage:

- **TestHologresTypeConverter**: Covers all supported type conversions including boolean, integer types, float types, date/time types, decimal, varchar, char, text, binary, array types, external types, and error cases for nullable arrays and multidimensional arrays.

- **TestHologresColumnDefaultValueConverter**: Covers `toGravitino()` for all literal types (boolean, short, integer, long, float, double, decimal, varchar, text, bpchar, date, time, timestamp), expression fallback to `UnparsedExpression`, `NULL::type` handling, `CURRENT_TIMESTAMP`, null default value (nullable vs non-nullable), unknown type fallback, and `fromGravitino()` for `UnparsedExpression`, `Literal`, and `DEFAULT_VALUE_NOT_SET`.

- **TestHologresExceptionConverter**: Covers all SQLSTATE error code mappings (42P04, 42P06, 42P07, 3D000, 3F000, 42P01, 08xxx), null SQL state handling, and password authentication failure detection.

- **TestHologresCatalogCapability**: Covers naming validation patterns (valid names with letters, underscores, digits, hyphens, dollar signs, slashes, Unicode), invalid names (starting with digit, empty, too long), max length validation, and reserved word enforcement for SCHEMA scope.

- **TestHologresTableOperations** (830 lines, 35+ test cases): Covers:

    - **CREATE TABLE SQL generation**: Basic, with comments, column comments, distribution, properties, property filtering for `distribution_key`/`is_logical_partitioned_table`/`primary_key`, physical partition, logical partition (1 and 2 columns), auto-increment rejection, nullable columns, default values, multiple columns, single quote escaping in comments, full-featured table, generated columns (non-nullable and nullable).
    - **RENAME TABLE**, **DROP TABLE**, **PURGE TABLE** (unsupported exception).

- **Partitioning SQL**: Physical, logical (single/two columns), rejects non-list transform, rejects multiple columns for physical, rejects >2 columns for logical, rejects empty partitions, rejects multiple transforms, rejects nested field names.
- **Index field string generation**: Single column, multiple columns, rejects nested column.
- **Datetime precision calculation**: Timestamp, timestamptz, time, timetz, non-datetime type, negative scale.

Note: ALTER TABLE operations `UpdateColumnDefaultValue`, `UpdateColumnType`, `UpdateColumnNullability`, `AddIndex`, `DeleteIndex`, and `UpdateColumnAutoIncrement` are all rejected with `IllegalArgumentException` in `generateAlterTableSql()`. CREATE TABLE with indexes and constraints is still supported.

Next Phase: Implementation Roadmap

Phase 1: View Support

Support displaying and creating VIEWs in Hologres.

1a. View Display

- Include `VIEW` in `getTables()` and `getTable()` table type filters to make views visible in Gravitino.
- Display view definitions (the underlying SQL query) by querying `pg_catalog.pg_views` or `information_schema.views` to retrieve the `view_definition` SQL.
- Expose the view definition as a table property or comment so that users can inspect the view logic from Gravitino.

1b. View Creation

- Support creating views from Gravitino by specifying the view definition SQL.
- Generate `CREATE VIEW` / `CREATE OR REPLACE VIEW` DDL statements.

CREATE VIEW "my_view" AS

SELECT id, name, create_time

FROM "my_table"

WHERE status = 'active';

Phase 2: Physical Partition Child Table Support

Optimize display and creation of physical partition child tables.

2a. Partition Child Table Display

- Display partition child tables under their partition parent with partition value information.
- Query `pg_catalog.pg_inherits` and `pg_catalog.pg_class` system tables to resolve parent-child relationships.
- Show partition bounds (e.g., `FOR VALUES IN ('20201215')`) for each child partition.

2b. Partition Child Table Creation

- Support creating partition child tables from Gravitino by specifying the parent table and partition values.
- Generate `CREATE TABLE child_table PARTITION OF parent_table FOR VALUES IN ('value')` DDL statements.

CREATE TABLE "physical_pt_20201216" PARTITION OF "physical_parent_pt"

FOR VALUES IN ('20201216');

2c. Auto-Partitioning Configuration

- Support configuring auto-partitioning properties during partition parent table creation:
  - `auto_partitioning_enable`: Enable/disable automatic partition creation.
  - `auto_partitioning_time_unit`: Time unit for auto-partitioning (`DAY`, `MONTH`, `YEAR`).
  - `auto_partitioning_num_precreate`: Number of partitions to pre-create.
  - `auto_partitioning_num_retention`: Number of partitions to retain.

Phase 3: Dynamic Table Support

Support displaying and creating Hologres Dynamic Tables (materialized views with automatic refresh).

3a. Dynamic Table Display

- Display properties specific to Hologres dynamic tables, including:
  - `table_type = 'dynamic'` identifier.
  - Refresh interval and refresh mode (full / incremental).
  - Source query SQL definition.
  - Materialization status and last refresh time.
- Read dynamic table metadata from `hologres.hg_table_properties` and system catalogs.

3b. Dynamic Table Creation

- Support creating dynamic tables from Gravitino.
- Generate the appropriate Hologres-specific DDL.

```
CREATE DYNAMIC TABLE "my_dynamic_table"

WITH (

   refresh_interval = '60',

   refresh_mode = 'incremental'

)

AS SELECT id, name, SUM(amount) AS total_amount

FROM "source_table"

GROUP BY id, name;
```

Phase 4: Foreign Table Support

Support displaying and creating foreign tables to enable federated queries across different data sources.

4a. Foreign Table Display

- Include `FOREIGN TABLE` in `getTables()` and `getTable()` table type filters to make foreign tables visible in Gravitino.
- Display foreign table server and options information in table properties.

4b. Foreign Table Creation

Hologres supports the following foreign table types:

- **MaxCompute Foreign Table**: Query MaxCompute (ODPS) tables directly from Hologres.

```
IMPORT FOREIGN SCHEMA "project_name" LIMIT TO ("mc_table")

FROM SERVER odps_server INTO "public";
```

- **Iceberg Foreign Table**: Query Iceberg tables via Hologres foreign table interface.

```
CREATE FOREIGN TABLE "iceberg_table" (

   id int8,

   name text
```

) SERVER iceberg_server

OPTIONS (

table_path 's3://bucket/warehouse/db/table'

);

- **Paimon Foreign Table**: Query Apache Paimon tables via foreign table interface.

CREATE FOREIGN TABLE "paimon_table" (

id int8,

name text

) SERVER paimon_server

OPTIONS (

table_path 's3://bucket/warehouse/db/table'

);

- **Other Data Sources**: Support additional foreign server types as Hologres extends its foreign data wrapper (FDW) support, such as OSS, DLF, etc.

Implementation approach:

- Map foreign table properties (SERVER, OPTIONS) to Gravitino table properties.
- Support `CREATE FOREIGN TABLE` and `IMPORT FOREIGN SCHEMA` DDL generation.

Phase 5: Advanced Index Support

Support displaying and creating more index types beyond PRIMARY KEY .

5a. Full-Text Inverted Index (FULLTEXT)

- Support creating full-text inverted indexes on text columns for accelerating full-text search queries.
- Generate `CALL set_table_property('table', 'fulltext_columns', 'col1:analyzer1,col2:analyzer2')` DDL statements.

5b. Vector Index (HGraph / HNSW)

- Support creating vector indexes for approximate nearest neighbor (ANN) search.

- Generate appropriate DDL for vector index creation.

5c. Global Secondary Index (GSI)

- Support creating global secondary indexes for accelerating non-primary-key point queries.
- Generate `CALL create_global_secondary_index('index_name', 'table_name', 'columns')` DDL statements.

For more information about Hologres indexes, refer to the following links:

- Primary Key
- Clustering Key
- Bitmap Index
- Full-Text Inverted Index
- Vector Index

Phase 6: Table and Index Modification

Support modifying table properties, column attributes, and indexes on existing tables. Currently the following ALTER TABLE operations throw `IllegalArgumentException`: `UpdateColumnDefaultValue`, `UpdateColumnType`, `UpdateColumnNullability`, `AddIndex`, `DeleteIndex`, `UpdateColumnAutoIncrement`, `SetProperty`, `RemoveProperty`.

6a. SET/REMOVE Table Properties

- Implement `TableChange.SetProperty` and `TableChange.RemoveProperty` handlers in `generateAlterTableSql()` (currently throws `IllegalArgumentException`).
- Generate `CALL set_table_property('table', 'key', 'value')` DDL statements for setting properties.

6b. Column Attribute Modification

- Implement `UpdateColumnDefaultValue` to support `ALTER COLUMN SET DEFAULT`.
- Implement `UpdateColumnType` to support `ALTER COLUMN SET DATA TYPE`.
- Implement `UpdateColumnNullability` to support `SET NOT NULL` / `DROP NOT NULL`.

6c. Advanced ALTER TABLE Operations

- Support modifying table-level properties like `orientation`, `clustering_key`, `segment_key`, `bitmap_columns`, `dictionary_encoding_columns`, `time_to_live_in_seconds`.

- Support modifying binlog configuration (`binlog_level`, `binlog_ttl`).

6d. Index Modification

- Implement `AddIndex` and `DeleteIndex` to support adding/removing PRIMARY KEY constraints via ALTER TABLE.

Phase Summary

Phase	Feature	Priority
Phase 1	View display and creation	High
Phase 2	Physical partition child table display and creation	High
Phase 3	Dynamic table display and creation	Medium
Phase 4	Foreign table display and creation (MaxCompute, Iceberg, Paimon, etc.)	Medium
Phase 5	Advanced index support (Full-text, Vector, Global Secondary Index)	Medium
Phase 6	Table, column, and index modification (SET/REMOVE properties, column default/type/nullability, index add/delete)	Medium

Additional future work beyond the above phases:

- Integration tests against Hologres cloud service or PostgreSQL testcontainers.

Technical Approach

Gravitino Support

Core Module

Gravitino provides a set of interfaces in package `org.apache.gravitino.catalog.jdbc`. The Hologres catalog is built by extending these interfaces:

- **HologresCatalog** extends `JdbcCatalog`: Entry point of the catalog, creates all Hologres-specific converters and operations. Registered as `"jdbc-hologres"` via `shortName()`.

- **HologresSchemaOperations** extends `JdbcDatabaseOperations`: Manages Hologres schemas (PostgreSQL Schema = JDBC Schema, PostgreSQL Database = JDBC Catalog). Filters 10 system schemas and supports schema comments.

- **HologresTableOperations** extends `JdbcTableOperations`, implements `RequireDatabaseOperation`: Implements full table lifecycle including CREATE, ALTER, DROP, LIST, and LOAD. Overrides `getTables()` and `getTable()` to support `TABLE` and `PARTITIONED TABLE` types. Overrides `getTableProperties()` to read from `hologres.hg_table_properties` system table with JDBC-to-DDL property key conversion. Overrides `getTablePartitioning()` to detect both logical partitions (via `hg_table_properties`) and physical LIST partitions (via `pg_partitioned_table` system table). Overrides `getDistributionInfo()` to read distribution keys from `hg_table_properties`. Supports generating CREATE TABLE SQL with `WITH` clause, partitioning (`PARTITION BY LIST` / `LOGICAL PARTITION BY LIST`), indexes, distribution, generated columns (`GENERATED ALWAYS AS ... STORED`), auto-increment (`GENERATED BY DEFAULT AS IDENTITY`), and comments.

- **HologresTypeConverter** extends `JdbcTypeConverter`: Bidirectional mapping between Hologres PostgreSQL types and Gravitino types. Supports array types via `_prefix` to `[]` suffix conversion. Handles `TIMESTAMP`/`TIMESTAMPTZ` without precision suffix.

- **HologresExceptionConverter** extends `JdbcExceptionConverter`: Maps PostgreSQL SQLSTATE error codes (42P04, 42P06, 42P07, 3D000, 3F000, 42P01, 08xxx) to Gravitino exceptions.

- **HologresColumnDefaultValueConverter** extends `JdbcColumnDefaultValueConverter`: Handles PostgreSQL-style type casting in default values (e.g., `'value'::type`). Overrides `fromGravitino()` to support `UnparsedExpression` for generated column round-trip.

- **HologresCatalogOperations** extends `JdbcCatalogOperations`: Handles PostgreSQL JDBC driver deregistration on catalog close.

- **HologresCatalogCapability** implements `Capability`: Defines PostgreSQL-compatible naming rules and reserved words.

Description of Key Features

Table Type Support

Hologres (PostgreSQL-compatible) has multiple table types. The current implementation supports listing and loading the following types:

PostgreSQL TABLE_TYPE	Description	Implementation
`TABLE`	Regular tables and partition child tables	Supported via `getTables()` / `getTable()`
`PARTITIONED TABLE`	Partitioned parent tables	Supported via `getTables()` / `getTable()`
`VIEW`	Views	Planned (Phase 1)
`FOREIGN TABLE`	Foreign tables	Planned (Phase 4)

Partition

Hologres supports LIST partitioning with two variants:

Logical Partition Tables (Hologres V3.1+):

- Detected via `hologres.hg_table_properties` where `is_logical_partitioned_table = 'true'` and partition columns are stored in `logical_partition_columns`.
- Supports 1–2 partition columns.
- Created via `LOGICAL PARTITION BY LIST(col1[, col2])` syntax.

Physical Partition Tables:

- Detected via `pg_catalog.pg_partitioned_table` (partition strategy and `partattrs`) and `pg_catalog.pg_attribute` (attribute number to column name resolution) system tables.
- Supports exactly 1 partition column.
- Created via `PARTITION BY LIST(column)` syntax.

Hologres partition SQL syntax:

-- Physical partition parent table

CREATE TABLE <schema_name>.<table_name> (

<column_name> <column_type> [<column_constraints>, ...]

) PARTITION BY LIST(<column_name>);

-- Logical partition table (V3.1+)

CREATE TABLE <schema_name>.<table_name> (

<column_name> <column_type> [<column_constraints>, ...]

) LOGICAL PARTITION BY LIST(<column_name1>[, <column_name2>]);

Current Limitation: Creating partition child tables (e.g., `CREATE TABLE child PARTITION OF parent FOR VALUES IN ('value')`) is not yet supported. This is planned for Phase 2.

For more information about Hologres partitions, please refer to: https://help.aliyun.com/zh/hologres/developer-reference/create-partition-table

Table Properties

Hologres-specific table properties are read from the `hologres.hg_table_properties` system table, which stores each property as a separate row with `property_key` and `property_value`. The following user-relevant properties are exposed (controlled by `isUserRelevantProperty()`):

Property Key	Description	Example Value
`orientation`	Storage format	`column`, `row`, `row,column`
`clustering_key`	Clustering key columns	`id:asc`
`segment_key`	Event time column (segment key)	`create_time`
`bitmap_columns`	Bitmap index columns	`status,category`
`dictionary_encoding_columns`	Dictionary encoding columns	`city,province`
`time_to_live_in_seconds`	Data TTL setting	`2592000`
`table_group`	Table group name	`my_table_group`
`storage_format`	Internal storage format	`orc`, `sst`

Property Key	Description	Example Value
`binlog_level`	Binlog level	`replica`, `none`
`binlog_ttl`	Binlog TTL	`86400`
`is_logical_partitioned_table`	Whether logical partitioned	`true`
`partition_expiration_time`	Partition expiration time	`604800`
`partition_keep_hot_window`	Partition keep hot window	`172800`
`partition_require_filter`	Whether partition filter required	`true`
`partition_generate_binlog_window`	Partition binlog generation window	`86400`

Notes:

- The JDBC system table stores some property keys with dots (e.g., `binlog.level`, `binlog.ttl`), but the CREATE TABLE `WITH` clause uses underscores (e.g., `binlog_level`, `binlog_ttl`). The implementation handles this conversion automatically via `convertFromJdbcPropertyKey()`.
- During CREATE TABLE, the properties `distribution_key`, `is_logical_partitioned_table`, and `primary_key` are filtered from the `WITH` clause to avoid duplication with their dedicated parameters.
- Distribution key is managed separately via the `Distribution` parameter and mapped to `distribution_key` in the `WITH` clause.

Current Limitation: Modifying table properties via `SET/REMOVE` is not yet supported (planned for Phase 6).

Generated Column Support

Hologres supports generated (stored computed) columns using `GENERATED ALWAYS AS (expression) STORED` syntax. The implementation in `appendColumnDefinition()`:

- Detects `UnparsedExpression` default values during CREATE TABLE and generates `GENERATED ALWAYS AS (expr) STORED` instead of `DEFAULT expr`.
- Correctly places the `STORED` keyword before nullable constraints (`NOT NULL` / `NULL`).

- Distinguishes from auto-increment columns (`GENERATED BY DEFAULT AS IDENTITY`) by checking `!column.autoIncrement()` first.
- `HologresColumnDefaultValueConverter.fromGravitino()` supports round-tripping `UnparsedExpression` back to SQL.

Example:

"ds" timestamp GENERATED ALWAYS AS (date_trunc('day'::text, order_time)) STORED NOT NULL

Data Types

The following Hologres data types are supported with bidirectional mapping to Gravitino types:

Hologres Type	Gravitino Type	Notes
`bool`	`BooleanType`	
`int2` (SMALLINT)	`ShortType`	
`int4` (INTEGER)	`IntegerType`	
`int8` (BIGINT)	`LongType`	
`float4` (REAL)	`FloatType`	
`float8` (DOUBLE PRECISION)	`DoubleType`	
`numeric(p,s)`	`DecimalType(p,s)`	
`varchar(n)`	`VarCharType(n)`	`varchar` without length maps to `StringType`
`bpchar(n)` (CHAR)	`FixedCharType(n)`	
`text`	`StringType`	
`bytea`	`BinaryType`	
`date`	`DateType`	
`time`	`TimeType`	With optional precision
`timestamp`	`TimestampType` (without timezone)	Always emitted without precision suffix

Hologres Type	Gravitino Type	Notes
`timestamptz`	`TimestampType` (with timezone)	Always emitted without precision suffix
`_int4` (int4[])	`ListType(IntegerType, false)`	Array types via `_` prefix
`_int8` (int8[])	`ListType(LongType, false)`	
`_float4` (float4[])	`ListType(FloatType, false)`	
`_float8` (float8[])	`ListType(DoubleType, false)`	
`_bool` (bool[])	`ListType(BooleanType, false)`	
`_text` (text[])	`ListType(StringType, false)`	
`json`, `jsonb`, `uuid`, etc.	`ExternalType`	Unsupported types mapped as ExternalType

Notes:

- Array types are non-nullable element only (Hologres limitation).
- Multidimensional arrays are not supported (Hologres doesn't enforce dimensions internally).
- Types like `json`, `jsonb`, `uuid`, `inet`, `money`, `roaringbitmap` are mapped to `ExternalType` with the original type name preserved.
- Hologres does not support precision syntax for `TIMESTAMP`/`TIMESTAMPTZ` (e.g., `timestamptz(6)` is invalid), so the type converter always emits the base type without precision.

Index Support

The current implementation supports the following index types in CREATE TABLE only (adding/deleting indexes via ALTER TABLE is not supported):

- **PRIMARY KEY**: Supported via Gravitino's `PRIMARY_KEY` index type. Maps to PostgreSQL `PRIMARY KEY` constraint.
- Composite indexes are supported (multiple columns in a single index).

Future Work: Full-text Inverted Index (FULLTEXT), Vector Index (HGraph/HNSW), and Global Secondary Index (GSI) support is planned for Phase 5.

User Interface

For creating a Hologres JDBC catalog, these parameters are required:

- jdbc-url (format: `jdbc:postgresql://{ENDPOINT}:{PORT}/{DBNAME}`)
- jdbc-user (AccessKey ID or database username)
- jdbc-password (AccessKey or database password)
- jdbc-database (database name, mandatory for Hologres — validated in `HologresTableOperations.initialize()`)
- jdbc-driver (org.postgresql.Driver)

Optional parameters:

- jdbc-properties (additional JDBC connection properties like `reWriteBatchedInserts=true` for better batch insert performance, `ApplicationName=YourAppName` for query tracking)

Project Structure

The Hologres JDBC catalog is located under `catalogs-contrib/catalog-jdbc-hologres/`:

catalogs-contrib/catalog-jdbc-hologres/

├── build.gradle.kts

└── src/

  ├── main/

  │ ├── java/org/apache/gravitino/catalog/hologres/

  │ │ ├── HologresCatalog.java

  │ │ ├── HologresCatalogCapability.java

  │ │ ├── HologresCatalogOperations.java

  │ │ ├── converter/

  │ │ │ ├── HologresColumnDefaultValueConverter.java

  │ │ │ ├── HologresExceptionConverter.java

```
│   │   │   └── HologresTypeConverter.java
│   │   └── operation/
│   │       ├── HologresSchemaOperations.java
│   │       └── HologresTableOperations.java
│   └── resources/
│       ├── META-INF/services/org.apache.gravitino.CatalogProvider
│       └── jdbc-hologres.conf
└── test/
    └── java/org/apache/gravitino/catalog/hologres/
        ├── TestHologresCatalogCapability.java
        ├── converter/
        │   ├── TestHologresColumnDefaultValueConverter.java
        │   ├── TestHologresExceptionConverter.java
        │   └── TestHologresTypeConverter.java
        └── operation/
            └── TestHologresTableOperations.java
```

Testing and Documentation

Current Test Coverage

5 test classes providing comprehensive coverage:

- **TestHologresTypeConverter**: Unit tests covering all supported type conversions (boolean, integer types, float types, date/time types, decimal, varchar, char, text, binary, array types, external types), including error cases for nullable arrays and multidimensional arrays.

- **TestHologresColumnDefaultValueConverter** (210 lines): Unit tests covering `toGravitino()` for all literal types (boolean, short, integer, long, float, double, decimal, varchar, text, bpchar, date, time, timestamp), expression fallback to `UnparsedExpression`, `NULL::type` handling, `CURRENT_TIMESTAMP`, null default

value handling (nullable returns `Literals.NULL`, non-nullable returns `DEFAULT_VALUE_NOT_SET`), unknown type fallback, and `fromGravitino()` for `UnparsedExpression`, `Literal`, and `DEFAULT_VALUE_NOT_SET`.

- **TestHologresExceptionConverter** (126 lines): Unit tests covering all PostgreSQL SQLSTATE error code mappings (42P04, 42P06, 42P07, 3D000, 3F000, 42P01, 08xxx), null SQL state handling, and password authentication failure detection.

- **TestHologresCatalogCapability** (147 lines): Unit tests covering naming validation patterns (valid names with letters, underscores, digits, hyphens, dollar signs, slashes, Unicode), invalid names (starting with digit, empty, too long), max length validation, and reserved word enforcement for SCHEMA scope.

- **TestHologresTableOperations** (830 lines, 35+ test cases): Unit tests covering:

  - **CREATE TABLE SQL generation**: Basic, with comments, column comments, distribution, properties, property filtering for `distribution_key`/`is_logical_partitioned_table`/`primary_key`, physical partition, logical partition (1 and 2 columns), auto-increment rejection, nullable columns, default values, multiple columns, single quote escaping in comments, full-featured table, generated columns (non-nullable and nullable).
  - **RENAME TABLE**, **DROP TABLE**, **PURGE TABLE** (unsupported exception).
  - **Partitioning SQL**: Physical, logical (single/two columns), rejects non-list transform, rejects multiple columns for physical, rejects >2 columns for logical, rejects empty partitions, rejects multiple transforms, rejects nested field names.
  - **Index field string generation**: Single column, multiple columns, rejects nested column.
  - **Datetime precision calculation**: Timestamp, timestamptz, time, timetz, non-datetime type, negative scale.

  Note: ALTER TABLE operations `UpdateColumnDefaultValue`, `UpdateColumnType`, `UpdateColumnNullability`, `AddIndex`, `DeleteIndex`, and `UpdateColumnAutoIncrement` are all rejected with `IllegalArgumentException` in `generateAlterTableSql()`. CREATE TABLE with indexes and constraints is still supported.

About Hologres Testing Environment

For the GitHub integrated test environment, we have the following options:

1. **Mock-based testing**: Use mock objects to simulate Hologres responses for unit tests.

2. **PostgreSQL-based testing**: Since Hologres is PostgreSQL-compatible, we can use a PostgreSQL container (testcontainers-postgresql is already configured in `build.gradle.kts`) for basic compatibility testing of standard SQL operations.

3. **Hologres Cloud testing**: For full integration testing including Hologres-specific features (table properties from `hg_table_properties`, system schemas, etc.), we can use Hologres cloud service with appropriate credentials (stored as GitHub secrets).

Golden Answer Testing

A metadata-driven automated testing framework (`scripts/hologres-test/`) has been developed:

- **extract_metadata.sh**: Extracts metadata from a reference Hologres catalog via Gravitino REST API, generates golden answer JSON files for schemas and tables, and produces CREATE request JSON files.
- **create_and_verify.sh**: Creates schemas and tables via Gravitino REST API using golden CREATE request files, then loads and compares the results against golden answer files to verify correctness. Supports modes: create-only, verify-only, create+verify, and drop-first for re-testing.

Documentation

The following documentation will be provided:

1. **User Guide**: How to configure and use the Hologres JDBC Catalog in Gravitino
2. **Data Type Mapping Reference**: Mapping between Hologres and Gravitino data types
3. **Configuration Reference**: All supported configuration parameters and their descriptions