OracleStore Design Proposal

Purpose

The purpose of this document is to propose a new database schema and `RawStore` implementation that is optimized for Oracle RDBMS.

Problem

At the core of the Hive Metastore is the DataNucleus ORM layer, which is an abstraction layer that hides the implementation details of the underlying storage mechanism. The inefficiencies inherent in the existing ORM-based implementation are well known and a number of attempts to remedy or work around these limitations have been made, both internally and in the community.

Internally, we only ever deploy the Hive Metastore backed by Oracle, yet incur all of the overhead of supporting multiple RDBMS imposed by the ORM-based implementation. By restructuring the schema and writing a new implementation of `RawStore` we can achieve significant gains in both query latency and storage footprint, as well as increasing the readability and maintainability of the code.

The following sections refer to `OracleStore`. `OracleStore` is an implementation of `RawStore` designed for the Oracle RDBMS and is meant as a reference implementation for this proposal.

Schema

The current ORM-based schema defines 53 tables. The `OracleStore` reference implementation defines 22 tables to implement the core set of functionality required for the Hive Metastore to support query execution from the Hive CLI. Support for ACID, bucketed and skewed tables, and some other functionality has not been implemented yet.

The ORM-based schema defines a table hierarchy analogous to the structure of the objects used by the Hive Metastore. For example, consider TBLS which defines references for the Table object as follows:

TBLS -> DBS
TBLS -> SDS -> COLUMNS_V2
TBLS -> SDS -> SERDES

Parameter tables reference the ID of the associated entity table, such as:

SD_PARAMS -> SDS
SERDE_PARAMS -> SERDES

Partition objects are stored in PARTITIONS which has the same structure and relationships as TBLS.

Duplicate Data

One side-effect of this schema is that a large amount of data is duplicated, particularly with respect to Partition object data. Inverting the relationship between the parameter table and the entity table that it references creates an opportunity to share common data across entities and reduce storage requirements by removing duplicate data. This translates into reduced query latency and opportunities to optimize retrieval in `OracleStore`.

The following table illustrates the changes required to deduplicate SD_PARAMS data:

ORM-based Schema	OracleStore Schema	Comments

`CREATE TABLE "SD_PARAMS" (`
`  "SD_ID" NUMBER`
`  "PARAM_KEY" VARCHAR`
`  "PARAM_VALUE" VARCHAR`
`)`
	`CREATE TABLE "V2_SD_PARAMS" (`
	`  "TBL_ID" NUMBER`
	`  "SD_PARAM_ID" NUMBER`
	`  "NAME" VARCHAR`
	`  "VALUE" VARCHAR`
	`)`
		`TBL_ID` references `V2_TBLS` for CASCADED drop of table data. `SD_PARAM_ID` is a unique sequence ID.

Applying similar changes to all of the parameter tables results in the following reduction in storage:

Table	ORM-based Schema	OracleStore Schema	+/-
TABLE_PARAMS	63282		
PARTITION_PARAMS	11515651	130700	-98.9%
SD_PARAMS	0	0	
SERDE_PARAMS	5414529	17922	-99.7%

The OracleStore Schema merges TABLE_PARAMS and PARTITION_PARAMS into a single table so that Tables/Partitions can share parameters wherever possible. Deduplication is only performed amongst a Table and its Partitions and never across Tables. `transient_lastDdlTime` was promoted from the TABLE_PARAMS and PARTITION_PARAMS into their respective parents, because every Table/Partition parameter map is guaranteed to contain this key-value pair. However, some parameter maps contain only this key-value pair, so reference to these parameters can be replaced by NULL.

A similar treatment to deduplicate COLUMNS_V2 can be performed so that Tables/Partitions can share common column sets wherever possible. This requires the CD_ID reference to be

promoted from SDS to TBLS/PARTITIONS. Again, deduplication is only performed amongst a Table and its Partitions and never across Tables. The reduction in storage is as follows:

Table	ORM-based Schema	OracleStore Schema	+/-
COLUMNS_V2	24977286	696361	-97.2%

Finally, the same can be done for SDS and SERDES, if the tables are restructured slightly. The previous step promoted CD_ID from SDS into TBLS/PARTITIONS. Promoting LOCATION in a similar fashion removes the final piece of unique data from SDS, which allows deduplication of SDS. Also, SERDES can be merged with SDS, because, after removing CD_ID and LOCATION, the resulting cardinality is very low. The reduction in storage is as follows:

Table	ORM-based Schema	OracleStore Schema	+/-
SDS	3350275		
SERDES	3350807		
SDS JOIN SERDES	3350191	15	-100.0%

Refer to Appendix A for a comparison of the ORM-based Schema and the OracleStore Schema.

OracleStore Implementation

We created an implementation of `RawStore` designed to work with the proposed schema, called `OracleStore`. `OracleStore` does not use DataNucleus to manage connections to the database, instead it initializes and manages its own JDBC connection via a DBCP connection pool. This decision makes the code very straight-forward to understand and maintain, and gives complete control of the SQL queries to the implementation.

By definition, the ORM-based `ObjectStore` hides all of the low-level implementation details with the database, meaning that there is limited control over the SQL queries. The exception is `MetaStoreDirectSql`, which is an attempt to work around the limitations of DataNucleus. However, `MetaStoreDirectSql` optimizes for Partition-related queries only, and is still constrained by the schema imposed by DataNucleus, so optimizations can only go so far.

`OracleStore` borrows from `MetaStoreDirectSql` by preferring to executing batch queries wherever possible, thereby reducing the amount of back-and-forth traffic with the database. Executing batch queries in conjunction with the aforementioned schema changes result in significant performance gains in the form of lower query latency. And lower query latency means more queries can be executed in the same time period, increasing the overall system throughput.

One of the schema changes mentioned above was to invert the relationship between entities and their parameters, which has the benefit of making it possible for `OracleStore` to conditionally query the parameters based on whether the ID is null or not. In contrast, `ObjectStore` ends up executing one more query than is necessary in order to determine the existence of parameter data before fetching it. In fact, it is interesting to note that DataNucleus generates 19 different queries on calls to `get_table` to generate a single Table object. By contrast, `OracleStore` can generate the same object with 6 queries (or potentially 4 queries depending on the existence of entity parameters). Also, consider that much of the data has been deduplicated, so requests for lists of objects are guaranteed to share common data, which means significantly fewer calls to the database need to be made.

`OracleStore` implements a lazy caching policy when retrieving entity parameters and `StorageDescriptor/Serdes`, which seems to work well because the cardinality for a list of Partitions, for example, was found to be small for the production data sets we used. However, this could easily be changed to pre-fetch all (15 in our case) of the `StorageDescriptor/Serdes` and fetch all parameters as a batch. Despite the differences in how data is stored and retrieved, `OracleStore` constructs objects that look no different than those constructed by `ObjectStore`.

Finally, `OracleStore` ensures that no duplicates are inserted into V2_TBL_COLS, V2_SDS, or any of the entity parameter tables.

Migration

The migration process from the ORM-based Schema to the OracleStore Schema consists of a two step process.

The first step involves running a JDBC-based Java utility which deduplicates the tables mentioned above and produces a bridging table between the two schemas. The second step involves performing the ETL by running a SQL script to migrate all of the data from the ORM-based Schema to the OracleStore Schema.

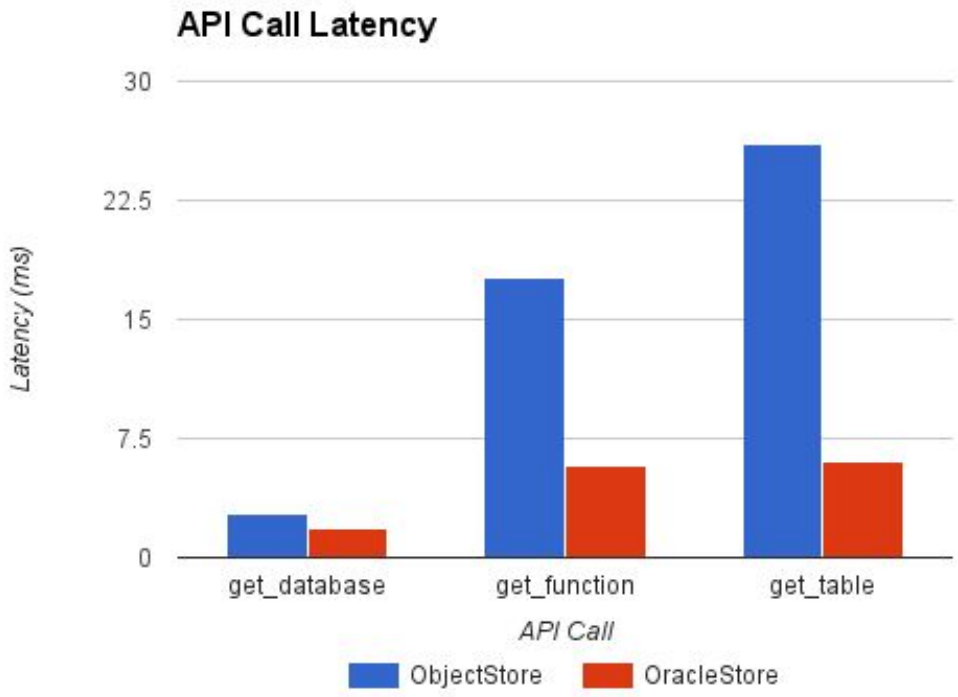
It is important to note that the OracleStore Schema entities are labelled such that they can coexist with the ORM-based Schema entities. This allows for a transition period where both schema can be maintained, using a hybrid `RawStore` implementation, for validation and debugging purposes, and failures in the `OracleStore` can fall back to `ObjectStore`.

Performance Comparison

The following section details results from two sets of performance tests. For purposes of this test, a copy of the Hive Metastore database from one of our research clusters and was loaded into a separate database. The original data consists of 643 DBS entries, 12170 TBLS entries

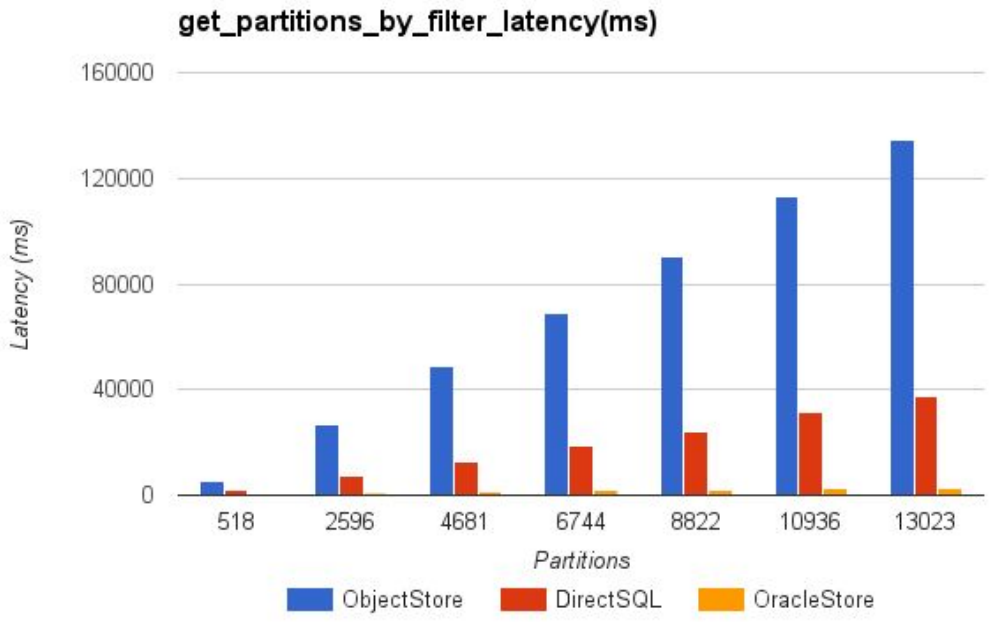
(varying in size from 1 column to 7027 columns), and 3338617 PARTITIONS entries, and 6 FUNCS entries.

The first test compares the latency of three point-lookup methods: `get_database`, `get_function`, and `get_table`. `ObjectStore` was configured with `hive.metastore.try.direct.sql=true`. Multiple runs were performed executing each method against the full set of data associated with that method. Results are shown in the following chart:

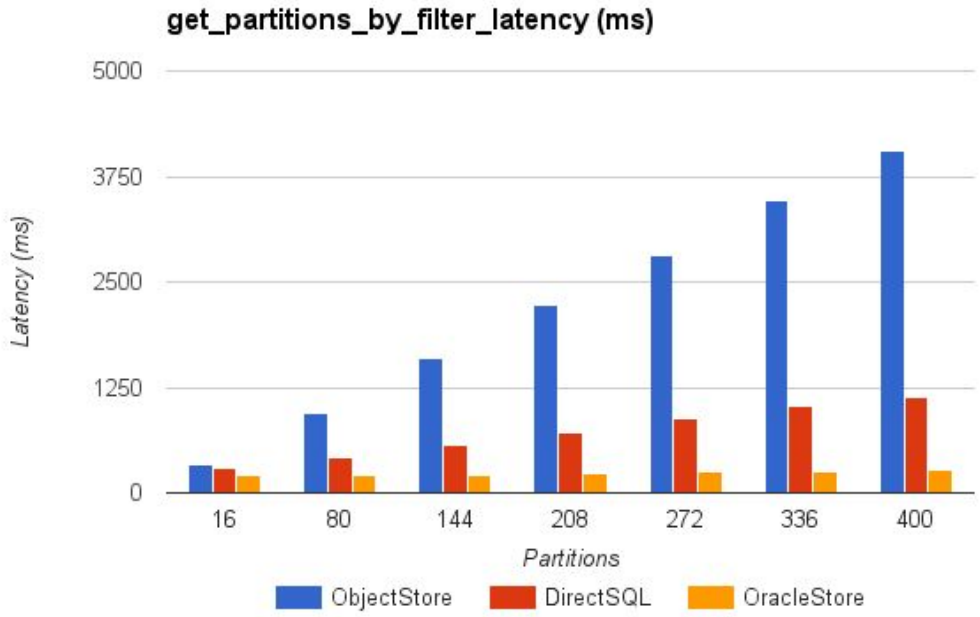


The second set of tests compares the latency of `get_partitions_by_filter` across `ObjectStore`, DirectSQL, and `OracleStore`. Here `ObjectStore` is configured with `hive.metastore.try.direct.sql=false` and DirectSQL is configured with `hive.metastore.try.direct.sql=true`. The table used in these tests has 6 levels of partitions, including a date string field, 81 columns, and has 88261 partitions. Test buckets are defined as 1, 4, 8, 12, 16, 20, and 24 hours of table data as restricted by a range filter on the date string partition field. Multiple runs were performed.

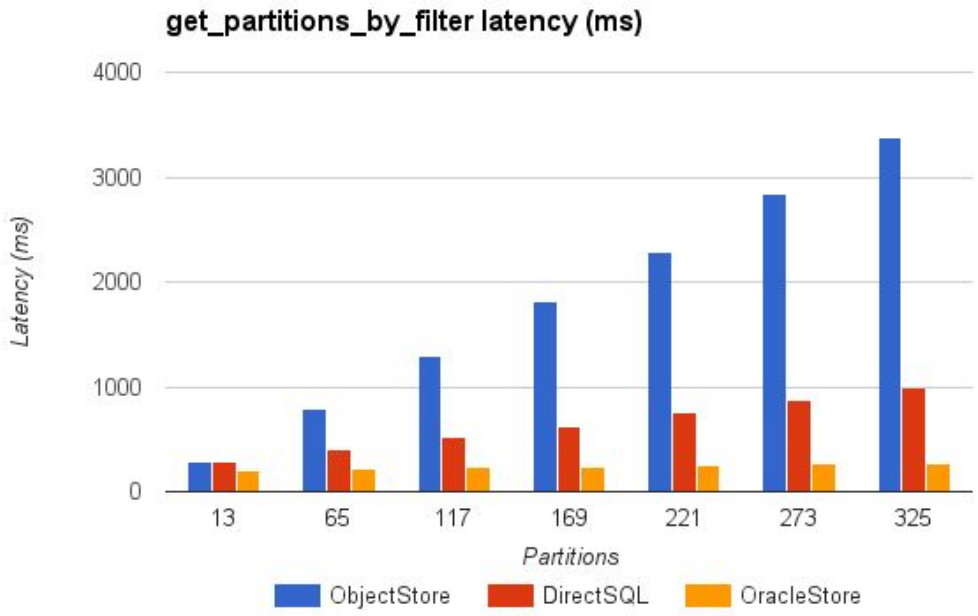
The following results were generated by using a single range filter on the first partition field to restrict partitions to the buckets defined above:



The following results were generated by adding an equality filter on the third partition field to the range filter on the first partition field:



The following results were generated by adding a second equality filter on the fifth partition to the equality filter on the third partition field and the range filter on the first partition field:



`get_database` (10.0%), `get_function` (6.9%), `get_table` (54.1%), and `get_partition_by_filter` (6.7%) are the top 4 methods invoked by the Hive Metastore, as determined from audit logs, and account for 77.7% of all requests.

Support For Other RDBMS

The `OracleStore` reference implementation was written to work specifically with Oracle, which means that it cannot be used as-is with other RDBMS. However, an effort was made to use generic SQL syntax wherever possible, so creating another reference implementation for a different database should be a relatively painless exercise. Furthermore, all of the non-SQL logic is independent of the RDBMS and could be reused as-is.

The following table lists some of the known differences that would have to be addressed when creating an implementation for MySQL or PostgreSQL.

Issue	**MySQL**	**PostgreSQL**
Data types	Replace with supported types (e.g. NUMBER(19) -> BIGINT).	Replace with supported types (e.g. NUMBER(19) -> BIGINT).
Sequence support	No. Replace with AUTO_INCREMENT field and call to LAST_INSERT_ID after insert.	Yes.
Limit query	Replace with MySQL syntax.	Replace with PostgreSQL syntax
String functions	Replace with MySQL functions.	Replace with PostgreSQL functions.

Future Work

The `OracleStore` implementation has shown very promising performance numbers, so the next step is to create a hybrid implementation of `RawStore` which inserts new data into tables of both schemas. A hybrid implementation is important because it would permit a more controlled migration path from the ORM-based Schema to the OracleStore Schema. During this transition period the hybrid implementation would ensure a fallback to known code in the case of failure caused by `OracleStore`. This insurance policy would come at the price of higher latency on data insert and delete operations and should not be looked at as a long term solution.

Calls to `get_table` are particularly prolific and it is not uncommon for the same call to be made, needlessly, multiple times in the course of generating a query plan. Implementing a caching mechanism would help to amortize the cost of such repetitive calls. HIVE-9453 already introduces support for this type of caching mechanism, in the context of `HBaseStore`, so `OracleStore` would just have to implement that functionality.

Finally, the ORM-based model objects and Thrift protocol objects are direct parallels of each other, which means the Thrift layer inherits all of the inefficiencies of the model objects. From previous sections, it is clear how much data is duplicated in the database. A similar amount of data duplication is present in the response to the client request. By breaking down the objects and packing them in a more efficient manner, it should be possible to realize similar reductions in the amount of data sent over the wire. The client would be responsible for reconstructing the objects from this compact representation.

Appendix A: Schema Comparison

ORM-based Schema	OracleStore Schema	Comments

CREATE TABLE "TABLE_PARAMS" (
  "TBL_ID" NUMBER
  "PARAM_KEY" VARCHAR2
  "PARAM_VALUE" VARCHAR2
)
		Merged into V2_TBL_PART_PARAMS.

CREATE TABLE
"PARTITION_PARAMS" (
  "PART_ID" NUMBER
  "PARAM_KEY" VARCHAR2
  "PARAM_VALUE" VARCHAR2
)
	CREATE TABLE
	"V2_TBL_PART_PARAMS" (
	  "TBL_ID" NUMBER
	  "TBL_PART_PARAM_ID" NUMBER
	  "NAME" VARCHAR2
	  "VALUE" VARCHAR2
	)
		Merged into V2_TBL_PART_PARAMS. TBL_PART_PARAM_ID is a unique sequence ID.

CREATE TABLE "SD_PARAMS" (
  "SD_ID" NUMBER
  "PARAM_KEY" VARCHAR
  "PARAM_VALUE" VARCHAR
)
	CREATE TABLE "V2_SD_PARAMS" (
	  "TBL_ID" NUMBER
	  "SD_PARAM_ID" NUMBER
	  "NAME" VARCHAR
	  "VALUE" VARCHAR
	)
		SD_PARAM_ID is a unique sequence ID.

CREATE TABLE "SERDE_PARAMS" (
  "SERDE_ID" NUMBER
  "PARAM_KEY" VARCHAR
  "PARAM_VALUE" VARCHAR
)
	CREATE TABLE
	"V2_SERDE_PARAMS" (
	  "TBL_ID" NUMBER
	  "SERDE_PARAM_ID" NUMBER
	  "NAME" VARCHAR
	  "VALUE" VARCHAR
	)
		SERDE_PARAM_ID is a unique sequence ID.

CREATE TABLE "SDS" (
  "SD_ID" NUMBER
  "CD_ID" NUMBER
  "INPUT_FORMAT" VARCHAR
  "IS_COMPRESSED" NUMBER
  "LOCATION" VARCHAR
  "NUM_BUCKETS" NUMBER
  "OUTPUT_FORMAT" VARCHAR
  "SERDE_ID" NUMBER
  "IS_STOREDASSUBDIRECTORIES"
NUMBER
)
	CREATE TABLE "V2_SDS" (
	  "SD_ID" NUMBER
	  "INPUT_FORMAT" VARCHAR
	  "OUTPUT_FORMAT" VARCHAR
	  "SERDE_NAME" VARCHAR
	  "SERDE_LIB" VARCHAR
	)
		Promoted CD_ID and LOCATION to V2_TBLS and V2_PARTITIONS. Demoted IS_COMPRESSED and IS_STOREDASSUBDIRECTORIES to V2_SD_PARAMS.

CREATE TABLE "SERDES" (
  "SERDE_ID" NUMBER
  "NAME" VARCHAR
  "SLIB" VARCHAR
)
		Merged into V2_SDS

CREATE TABLE "COLUMNS_V2" (
  "CD_ID" NUMBER
  "COMMENT" VARCHAR
  "COLUMN_NAME" VARCHAR
  "TYPE_NAME" VARCHAR
  "INTEGER_IDX" NUMBER
	CREATE TABLE "V2_TBL_COLS" (
	  "TBL_ID" NUMBER
	  "CD_ID" NUMBER
	  "POSITION" NUMBER
	  "NAME" VARCHAR
	  "TYPE" VARCHAR
		CD_ID is a unique sequence ID.

) 	"COMMENT" VARCHAR)	
CREATE TABLE "TBLS" ("TBL_ID" NUMBER "CREATE_TIME" NUMBER "DB_ID" NUMBER "LAST_ACCESS_TIME" NUMBER "OWNER" VARCHAR "RETENTION" NUMBER "SD_ID" NUMBER "TBL_NAME" VARCHAR "TBL_TYPE" VARCHAR "VIEW_EXPANDED_TEXT" CLOB "VIEW_ORIGINAL_TEXT" CLOB)	CREATE TABLE "V2_TBLS" ("TBL_ID" NUMBER "DB_ID" NUMBER "SD_ID" NUMBER "CD_ID" NUMBER "SD_PARAM_ID" NUMBER "SERDE_PARAM_ID" NUMBER "NAME" VARCHAR "TYPE" VARCHAR "OWNER_NAME" VARCHAR "LOCATION" VARCHAR "RETENTION" NUMBER "CREATION_TIME" NUMBER "LAST_MODIFIED_TIME" NUMBER "LAST_ACCESS_TIME" NUMBER "VIEW_EXPANDED_TEXT" CLOB "VIEW_ORIGINAL_TEXT" CLOB)	Added references to V2_TBL_COLS and V2_*_PARAMS. Added LAST_MODIFIED_TIME from TBL_PARAMS.
CREATE TABLE "PARTITIONS" ("PART_ID" NUMBER "CREATE_TIME" NUMBER "LAST_ACCESS_TIME" NUMBER "PART_NAME" VARCHAR "SD_ID" NUMBER "TBL_ID" NUMBER)	CREATE TABLE "V2_PARTITIONS" ("PART_ID" NUMBER "TBL_ID" NUMBER "SD_ID" NUMBER "CD_ID" NUMBER "SD_PARAM_ID" NUMBER "SERDE_PARAM_ID" NUMBER "NAME" VARCHAR "LOCATION" VARCHAR "CREATION_TIME" NUMBER "LAST_MODIFIED_TIME" NUMBER "LAST_ACCESS_TIME" NUMBER)	Added references to V2_TBL_COLS and V2_*_PARAMS. Added LAST_MODIFIED_TIME from TBL_PARAMS.
CREATE TABLE "CDS" ("CD_ID" NUMBER)		Removed. Replaced with V2_TBL_COLS_SEQ sequence.
CREATE TABLE "PARTITION_KEY_VALS" ("PART_ID" NUMBER "PART_KEY_VAL" VARCHAR "INTEGER_IDX" NUMBER)		Removed. Used by DirectSQL only; filtered queries are implemented differently.