Purpose for adding PK/FK constraints

The basic purposes for adding the integrity constraints in a Database are:

1. Data Cleanliness
2. Query Optimization

For HIVE-13076, we will concentrate on the Query Optimization aspect and implement the basic changes to support Query Optimization. With client specified NOVALIDATE RELY option, the Hive CBO eventually can rely on the PK/FK relationships and come up with better stats (cardinality) and perform unnecessary join eliminations and a better overall plan. Please note that the previous sentence is not implemented as part of HIVE-13076 in Hive CBO. The newly introduced JDBC APIs to retrieve the PK/FK associated with a table, can be used to perform unnecessary join elimination at the client side as well.

Usage Details

Basic Syntax for PK/FK

Examples:

```
CREATE TABLE product
  (
     product_id        INTEGER,
     product_vendor_id INTEGER,
     PRIMARY KEY (product_id) DISABLE NOVALIDATE RELY,
        CONSTRAINT product_fk_1 FOREIGN KEY (product_vendor_id)
REFERENCES vendor(vendor_id) DISABLE NOVALIDATE
  );

CREATE TABLE vendor
  (
     vendor_id INTEGER,
     CONSTRAINT pk1 PRIMARY KEY (vendor_id)
  );
```

Points to note :

- In the above syntax, [CONSTRAINT *constraint-Name*] is optional. If this is not specified by the user, we will use system generated constraint name. DISABLE and NOVALIDATE

options are mandatory because we do not support the intended default options (ENABLE and VALIDATE) as of now.

- RELY/NORELY is optional and the CBO is expected to use this information to perform join elimination if the constraint is defined with RELY option. The default value is NORELY.
- ENABLE and VALIDATE options can be implemented in future when data integrity aspect is covered. UNIQUE, NOT NULL, CHECK, DEFAULT constraints are out of scope for now and can be introduced later if required.
- A table can have at most one PRIMARY KEY constraint.
- The parent columns referred to in PK/FK relationship must be PRIMARY KEY columns and must belong the same table. We still don't support UNIQUE key, so we can lift this restriction to include UNIQUE keys as well once it is implemented.

Adding/Deleting PK/FK constraints via ALTER and DELETE commands

Examples:

```
ALTER TABLE sales
  ADD CONSTRAINT pk1 PRIMARY KEY (sales_id) DISABLE NOVALIDATE;

ALTER TABLE product
   ADD CONSTRAINT fk1 FOREIGN KEY p_id REFERENCES owner(o_id)DISABLE
NOVALIDATE;

ALTER TABLE sales
  DROP CONSTRAINT pk1;

ALTER TABLE product
  DROP CONSTRAINT fk1;
```

By default, the above option will be used while creating a constraint. We can specify rely to activate an existing constraint in NOVALIDATE mode.

Error Messages

Here are some of the errors you will encounter if you do not follow the specifications mentioned in the previous sections.

Example 1 : Unsupported features

```
CREATE TABLE product
  (
     product_id         INTEGER,
     product_vendor_id INTEGER,
     PRIMARY KEY (product_id) ENABLE NOVALIDATE,
        CONSTRAINT product_fk_1 FOREIGN KEY (product_vendor_id)
REFERENCES vendor(vendor_id) DISABLE NOVALIDATE
  );
```

We should throw an exception in the above scenario since we support only DISABLE CONSTRAINT.

Example 2 : Duplicate primary key.

```
ALTER TABLE sales
  ADD CONSTRAINT pk1 PRIMARY KEY (sales_1_id) DISABLE NOVALIDATE;
ALTER TABLE sales
  ADD CONSTRAINT pk2 PRIMARY KEY (sales_2_id) DISABLE NOVALIDATE;
```

We should throw an exception because we can have only one primary key for a table.

Example 3 : Duplicate constraint name.

```
ALTER TABLE sales_1
  ADD CONSTRAINT pk1 PRIMARY KEY (sales_1_id) DISABLE NOVALIDATE;
ALTER TABLE sales_2
  ADD CONSTRAINT pk1 PRIMARY KEY (sales_2_id) DISABLE NOVALIDATE;
```

We should throw an exception because we have a duplicate constraint name in the second ALTER statement.

Implementation Details

Hive Metastore changes

In order to support the above semantics from Hive, we need to introduce changes at the hive metastore schema side. Add the following new table.

KEY_CONSTRAINTS

CHILD_CD_ID	Child Column Descriptor ID
CHILD_INTEGER_IDX	Child Integer Index. I.e. the Position of the Column in the Child Table's Column Descriptor
CHILD_TBL_ID	Child Table ID
PARENT_CD_ID	Parent Column Descriptor ID
PARENT_INTEGER_IDX	Parent Integer Index. I.e. the Position of the Column in the Parent Table's Column Descriptor
PARENT_TBL_ID	Parent Table ID
POSITION	The key sequence or the position of this column in this constraint (One based Index)
CONSTRAINT_NAME	Name of the constraint
CONSTRAINT_TYPE	0 - Primary Key, 1 - Foreign Key
UPDATE_RULE	Unimplemented as of now, 0 by default
DELETE_RULE	Unimplemented as of now, 0 by default
ENABLE_VALIDATE_RELY	Bit vector containing Enable Validate and Rely details

The primary key will be (CONSTRAINT_NAME, POSITION)

JDBC APIs to retrieve PK/FK relationships

1. public OperationHandle getPrimaryKeys(SessionHandle sessionHandle, String catalog, String schema, String table) throws HiveSQLException
2. public OperationHandle getCrossReference(SessionHandle sessionHandle, String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable) throws HiveSQLException

The getPrimaryKeys() thrift response should have the following fields in order:

TABLE_CAT String => table catalog (may be null)
TABLE_SCHEM String => table schema (may be null)
TABLE_NAME String => table name
COLUMN_NAME String => column name
KEY_SEQ short => sequence number within primary key(a value of 1 represents the first column of the primary key, a value of 2 would represent the second column within the primary key).
PK_NAME String => primary key name (may be null)

The getCrossReference() thrift response should have the following fields in order:

PKTABLE_CAT String => parent key table catalog (may be null)
PKTABLE_SCHEM String => parent key table schema (may be null)
PKTABLE_NAME String => parent key table name
PKCOLUMN_NAME String => parent key column name
FKTABLE_CAT String => foreign key table catalog (may be null) being exported (may be null)
FKTABLE_SCHEM String => foreign key table schema (may be null) being exported (may be null)
FKTABLE_NAME String => foreign key table name being exported
FKCOLUMN_NAME String => foreign key column name being exported
KEY_SEQ short => sequence number within foreign key(a value of 1 represents the first column of the foreign key, a value of 2 would represent the second column within the foreign key).
UPDATE_RULE short => What happens to foreign key when parent key is updated:
importedNoAction - do not allow update of parent key if it has been imported
importedKeyCascade - change imported key to agree with parent key update
importedKeySetNull - change imported key to NULL if its parent key has been updated
importedKeySetDefault - change imported key to default values if its parent key has been updated
importedKeyRestrict - same as importedKeyNoAction (for ODBC 2.x compatibility)
DELETE_RULE short => What happens to the foreign key when parent key is deleted.
importedKeyNoAction - do not allow delete of parent key if it has been imported
importedKeyCascade - delete rows that import a deleted key
importedKeySetNull - change imported key to NULL if its primary key has been deleted
importedKeyRestrict - same as importedKeyNoAction (for ODBC 2.x compatibility)
importedKeySetDefault - change imported key to default if its parent key has been deleted
FK_NAME String => foreign key name (may be null)
PK_NAME String => parent key name (may be null)
DEFERRABILITY short => can the evaluation of foreign key constraints be deferred until commit
importedKeyInitiallyDeferred - see SQL92 for definition
importedKeyInitiallyImmediate - see SQL92 for definition
importedKeyNotDeferrable - see SQL92 for definition