Dependent Tables

Hive supports both partitioned and unpartitioned external tables. In both cases, when a new table/partition is being added, the location is also specified for the new table/partition. Let us consider a specific example:

create table T (key string, value string) partitioned by (ds string, hr string);
insert overwrite table T partition (ds='1', hr='1') ...;
..
insert overwrite table T partition (ds='1', hr='24') ...;

T is a partitioned table by date and hour, and Tsignal is an external table which conceptually denotes the creation of the signal table.

create external table Tsignal (key string, value string) partitioned by (ds string);

When all the hourly partitions are created for a day (ds='1'), the corresponding partition can be added to Tsignal

alter table Tsignal add partition (ds='1') location 'Location of T'/ds=1;

There is a implicit dependency between Tsignal@ds=1 and T@ds=1/hr=1, T@ds=1/hr=2, ... T@ds=1/hr=24, but that dependency is not captured anywhere
in the metastore. It would be useful to have an ability to explicitly create that dependency. This dependency can be used for all kinds of auditing purposes. For eg. when the following query is performed:

select .. from Tsignal where ds = '1';

the inputs only contains Tsignal@ds=1, but is should also contain T@ds=1/hr=1, T@ds=1/hr=2,....T@ds=1/hr=24

This dependency should be captured by the metastore. For simplicity, let us assume we create a new notion of dependent tables (instead of overloading external tables).

create dependency table Tdependent (key string, value string) partitioned by (ds string);

This is like a external table but also captures the dependency (we can also enhance external tables for the same).

alter table Tdependent add partition (ds='1') location '/T/ds=1' dependent partitions table T partitions (ds='1');
specify the partial partition spec for the dependent partitions.
Note that each table can point to different locations - hive needs to ensure that all the dependent partitions are under the location 'T/ds=1'

- Specify the location

The metastore can store the dependencies completely or partially.

- - Materialize the dependencies both-ways
    Tdependent@ds=1 depends on T@ds=1/hr=1 to T@ds=1/hr=24
    T@ds=1/hr=1 is depended upon by T@ds=1
    Advantages: if T@ds=1/hr=1 is dropped, T@ds=1 can be notified or it can choose to dis-allow this
    Any property on Tdependent can be propagated to T

- - Is the dependency used for querying ? What happens if T@ds=1/hr=25 gets added ? The query 'select .. from Tdependent where ds = 1' includes T@ds=1/hr=25, but this is not shown in the inputs.
  - Dont use the location for querying - then why have the location ?

- - Store partial dependencies
    Tdependent@ds=1 depends on T@ds=1 (spec).
    At describe time, the spec is evaluated and all the dependent partitions are computed dynamically. At add partition time, verify that the location captures all dependent partitions.
    The partial spec is not used for querying - location is used for that. At query time, verify that the location captures all dependent partitions.

- The dependent table does not have a location.
  - The list of partitions are computed at query time - think of it like a view, where each partition has its own definition limited to 'select * from T where partial/full partition spec'. Query layer needs to change. Is it possible ? Unlike a view, it does not rewritten at semantic analysis time. After partition pruning is done (on a dependent table), rewrite the
    tree to contain the base table T - the columns remain the same, so it should be possible.

With this, it is possible that the partitions point to different tables.
For eg:

alter table Tdependent add partition (ds='1') depends on table T1 partition (ds='1');
alter table Tdependent add partition (ds='2') depends on table T2 partition (ds='2');

Something that can be achieved by external tables currently. The dependent partitions are computed dynamically - T1@ds=1/hr=1 does not know the fact that it is dependent upon by Tdependent@ds=1.
T1@ds=1/hr=1 can be dropped anytime, and Tdependent@ds=1 automatically stops depending upon it from that point.

I am leaning towards this - the user need not specify both the location and the dependent partitions.
Can the external tables be enhanced to support this ? Will create a problem for the query layer, since the external tables are handled differently today.

- - The list of dependent partitions are materialized and stored in the metastore, and use that for querying.
    A query like 'select .. from Tdependent where ds = 1' gets transformed to 'select .. from (select * from T where ((ds = 1 and hr = 1) or (ds

= 1 and hr = 2) .... or (ds=1 and hr=24))'
Can put a lot of load on the query layer.

= Final Proposal =
Instead of enhancing external tables to solve a convoluted usecase, create a new type of tables - dependent tables. The reason for the existence of external tables is to point to some existing data.

== Dependent Tables ==
Create a table which explicitly depends on another table. Consider the following scenario for the first use case mentioned:

```
create table T (key string, value string) partitioned by (ds string, hr string);

-- create a dependent table which specifies the dependencies explicitly
-- Tdep inherits the schema of T
-- Tdep is partitioned by a prefix of T (either ds or ds,hr)
create dependent table Tdep partitioned by (ds string) depends on table T;
```

In order to denote the end of a daily partition for T, the corresponding partition is added in Tdep

```
-- create partition T@ds=1/hr=1 to T@ds=1/hr=24
alter table Tdep add partition (ds=1);
```

The metastore stores the following dependencies

- Tdep depends on T, and T is dependent upon by Tdep
- Tdep@ds=1 depends on T@ds=1

In some usecases, it is possible that some partitions of Tdep depends on T, whereas the newer partitions of Tdep depend on T2.

```
create table T (key string, value string) partitioned by (ds string, hr string);

-- create a dependent table which specifies the dependencies explicitly
-- Tdep inherits the schema of T
-- Tdep is partitioned by a prefix of T (either ds or ds,hr)
create dependent table Tdep partitioned by (ds string) depends on table T;

-- create partition T@ds=1/hr=1 to T@ds=1/hr=24
alter table Tdep add partition (ds=1);

-- repeat this for T@ds=1 to T@ds=10

-- T is being deprecated, and T2 is the new table (with the same schema, possibly partitioned on different keys)

create table T2 (key string, value string) partitioned by (ds string, hr string, min string);

alter table Tdep depends on table T2;

-- create partition T2@ds=11/hr=1/min=1 to T2@ds=11/hr=24/min=60
alter table Tdep add partition (ds=1);
```

The metastore stores the following dependencies

- Tdep depends on (T,T2), and (T,T2) are dependent upon by Tdep
- T and T2 have the same schema
- Tdep@ds=1..Tdep@ds=10 depends on T@ds=1..T@ds=10 respectively
- Tdep@ds=11 depends on T2@ds=11

The query on Tdep is re-written to access the underlying tables. For eg. for the query 'select count(1) from Tdep where ds = 1', once partition pruning is done (on Tdep), the operator tree (TableScan(Tdep):ds=1 -> Select) is rewritten to (TableScan(T):ds=1/hr=1 to ds=1/hr=24 -> Select). The advantage of this over external tables is that all the table properties (bucketing/sorting/list bucketing) for the underlying tables are used. This can easily extend to multiple tables. For eg. for the query 'select count(1) from Tdep where ds = 1 or ds = 11', once partition pruning is done (on Tdep), the operator tree (TableScan(Tdep):ds=1,ds=11 -> Select) is rewritten to (TableScan(T):ds=1/hr=1 to ds=1/hr=24 -> Select -> Union) and (TableScan(T2):ds=1/hr=1 to ds=1/hr=24 -> Select -> Union).

- If the schema of T changes, it can either be dis-allowed or propagated to Tdep.
- desc extended will be enhanced to show all the dependent partitions.
- The dependency of an existing partition can be changed: alter table Tdep partition (ds=10) depends on table T2