Table of Contents

1. Some Definitions

Metadata: higher level data.

In our zoo table, we store data about animals. For a dbms to work and for us to write queries, the dbms has to store data about the zoo table, its columns, their data types etc. This higher level of data is called metadata. The metadata is used to manage the use of the "regular" data (such as 'cat', 'Fluffy', 45.98). The term metadata is used in many places with somewhat different purposes; for a dbms the term metadata generally refers to the data maintained by the dbms to handle the database objects. The definition of metadata as being "data about data" is common but not terribly helpful.

Data dictionary, system catalog: The dbms has to store the metadata some place; a relational database needs to store the data in tables; this collection of tables is called the data dictionary or system catalog

2. Information_Schema

We need to be able to get the metadata about our databases, tables and other database objects via SQL from tables stored in and maintained by the dbms. The ANSI standard for this is called Information Schema- a series of tables (actually read-only views) that a user can read to see the metadata. Often the dbms will make some of this information available with proprietary commands such as the MySQL Show and Desc.

MySQL uses the Show statements to let the user get information about their database objects. The Show statement is a MySQL command and provides easy access to the more commonly useful metadata.

Demo 01:

show create table a_testbed.zoo_animals\G

```
*************************** 1. row ***************************
       Table: zoo_animals
Create Table: CREATE TABLE `zoo_animals` (
  `an_id` int(10) unsigned NOT NULL,
  `an_name` varchar(25) DEFAULT NULL,
  `an_type` varchar(25) NOT NULL,
  `an_cost` decimal(7,2) unsigned DEFAULT NULL,
  `an_dob` datetime NOT NULL,
  `an_aquired` date NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.02 sec)
```

With MySQL, there is also a virtual database, called the Information_Schema, which contains metadata. This set of tables follows the ANSI standards- which means that techniques you develop in MySQL are apt to be similar to techniques you would use with other dbms. Every dbms provider implements Information_Schema in its own way so you will find some differences between the MYSQL implementation and others- for example MySQL allows the use of several engines so it provides an Engine column for tables.

You will also find that some columns appear in the tables but are always null. MySQL plans for new versions and has added some columns to its tables ahead of anticipated use and to meet the standards in terms of columns even if not functionality. The data that is accessed through Information_Schema is pulled from the mysql system database and other system sources.

The information that a specific user can access is limited to objects for which that user has the appropriate privileges.

Since the Information_Schema database contains tables, you can use SQL queries to access the tables in this database.

Demo 02: This is a MySQL command to show the databases the account can access.

```
show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| a_book_orders      |
| a_books            |
. . .  rows omitted
```

Demo 03: These are some of the tables in the Information_Schema database.

```
show tables from information_schema;
+---------------------------------------+
| Tables_in_information_schema          |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| ENGINES                               |
| EVENTS                                |
| FILES                                 |
| GLOBAL_STATUS                         |
| GLOBAL_VARIABLES                      |
| KEY_COLUMN_USAGE                      |
| PARAMETERS                            |
| PARTITIONS                            |
| PLUGINS                               |
| PROCESSLIST                           |
| PROFILING                             |
| REFERENTIAL_CONSTRAINTS               |
| ROUTINES                              |
| SCHEMATA                              |
| SCHEMA_PRIVILEGES                     |
| SESSION_STATUS                        |
| SESSION_VARIABLES                     |
| STATISTICS                            |
| TABLES                                |
| TABLESPACES                           |
| TABLE_CONSTRAINTS                     |
| TABLE_PRIVILEGES                             |
| TRIGGERS                                     |
| USER_PRIVILEGES                              |
| VIEWS                                        |
. . .  rows omitted
```

Some of these tables are more oriented towards dba tasks while others are more useful for a developer. As you might anticipate some of these tables may have a lot of data and you do not want to just run select * against these- especially on a production system. First do a desc of these tables.

2.1. TABLES

The table_types values are Base Table and View. The value for table_rows is an estimate used for optimization. In this table the schema is stored in the Table_Schema column.

Demo 04:

```
desc information_schema.tables;
+-----------------+---------------------+------+-----+---------+-------+
| Field           | Type                | Null | Key | Default | Extra |
+-----------------+---------------------+------+-----+---------+-------+
| TABLE_CATALOG   | varchar(512)        | NO   |     |         |       |
| TABLE_SCHEMA    | varchar(64)         | NO   |     |         |       |
| TABLE_NAME      | varchar(64)         | NO   |     |         |       |
| TABLE_TYPE      | varchar(64)         | NO   |     |         |       |
| ENGINE          | varchar(64)         | YES  |     | NULL    |       |
| VERSION         | bigint(21) unsigned | YES  |     | NULL    |       |
| ROW_FORMAT      | varchar(10)         | YES  |     | NULL    |       |
| TABLE_ROWS      | bigint(21) unsigned | YES  |     | NULL    |       |
| AVG_ROW_LENGTH  | bigint(21) unsigned | YES  |     | NULL    |       |
| DATA_LENGTH     | bigint(21) unsigned | YES  |     | NULL    |       |
| MAX_DATA_LENGTH | bigint(21) unsigned | YES  |     | NULL    |       |
| INDEX_LENGTH    | bigint(21) unsigned | YES  |     | NULL    |       |
| DATA_FREE       | bigint(21) unsigned | YES  |     | NULL    |       |
| AUTO_INCREMENT  | bigint(21) unsigned | YES  |     | NULL    |       |
| CREATE_TIME     | datetime            | YES  |     | NULL    |       |
| UPDATE_TIME     | datetime            | YES  |     | NULL    |       |
| CHECK_TIME      | datetime            | YES  |     | NULL    |       |
| TABLE_COLLATION | varchar(32)         | YES  |     | NULL    |       |
| CHECKSUM        | bigint(21) unsigned | YES  |     | NULL    |       |
| CREATE_OPTIONS  | varchar(255)        | YES  |     | NULL    |       |
| TABLE_COMMENT   | varchar(2048)       | NO   |     |         |       |
+-----------------+---------------------+------+-----+---------+-------+

Select table_name, create_time
From information_schema.tables
Where table_schema= 'a_emp';
+----------------+---------------------+
| table_name     | create_time         |
+----------------+---------------------+
| departments    | 2011-01-05 23:10:59 |
| employees      | 2011-01-05 23:11:27 |
| jobs           | 2011-01-05 23:11:27 |
| location_types | 2011-06-21 21:51:30 |
| locations      | 2011-01-05 23:10:59 |
+----------------+---------------------+
```

2.2. TABLE_CONSTRAINTS

The constraint_types are Primary Key, Foreign Key and Unique. The constraint name for the pk seems to always be Primary even if you define it as a constraint with a different name. You can define a check on a table but since this version of MySQL does not support that constraint, it does not appear in this table.

Demo 05:

```
desc information_schema.table_constraints;
+--------------------+--------------+------+-----+---------+-------+
| Field              | Type         | Null | Key | Default | Extra |
+--------------------+--------------+------+-----+---------+-------+
| CONSTRAINT_CATALOG | varchar(512) | NO   |     |         |       |
| CONSTRAINT_SCHEMA  | varchar(64)  | NO   |     |         |       |
| CONSTRAINT_NAME    | varchar(64)  | NO   |     |         |       |
| TABLE_SCHEMA       | varchar(64)  | NO   |     |         |       |
| TABLE_NAME         | varchar(64)  | NO   |     |         |       |
| CONSTRAINT_TYPE    | varchar(64)  | NO   |     |         |       |
+--------------------+--------------+------+-----+---------+-------+
```

Demo 06: -- how many foreign keys do we have for each table that has any foreign key?

```
Select TC.table_name , count(*)
From information_schema.table_constraints TC
Where TC.table_schema= 'a_emp'
and TC.constraint_type = 'FOREIGN KEY'
Group by TC.table_name;
+-------------+----------+
| table_name  | count(*) |
+-------------+----------+
| departments |        1 |
| employees   |        3 |
| locations   |        1 |
+-------------+----------+
```

2.3. COLUMNS

The data_type reports values such as varchar; column_type reports values such as varchar (25). The privileges column lists the privileges for this user for this table column.

The column_key values are PRI- for a pk, UNI for unique, MUL indicates this column is part of an index (often a fk)

Demo 07:

```
desc information_schema.columns;
+--------------------------+---------------------+------+-----+---------+-------+
| Field                    | Type                | Null | Key | Default | Extra |
+--------------------------+---------------------+------+-----+---------+-------+
| TABLE_CATALOG            | varchar(512)        | NO   |     |         |       |
| TABLE_SCHEMA             | varchar(64)         | NO   |     |         |       |
| TABLE_NAME               | varchar(64)         | NO   |     |         |       |
| COLUMN_NAME              | varchar(64)         | NO   |     |         |       |
| ORDINAL_POSITION         | bigint(21) unsigned | NO   |     | 0       |       |
| COLUMN_DEFAULT           | longtext            | YES  |     | NULL    |       |
| IS_NULLABLE              | varchar(3)          | NO   |     |         |       |
| DATA_TYPE                | varchar(64)         | NO   |     |         |       |
| CHARACTER_MAXIMUM_LENGTH | bigint(21) unsigned | YES  |     | NULL    |       |
| CHARACTER_OCTET_LENGTH   | bigint(21) unsigned | YES  |     | NULL    |       |
| NUMERIC_PRECISION          | bigint(21) unsigned | YES  |     | NULL    |       |
| NUMERIC_SCALE              | bigint(21) unsigned | YES  |     | NULL    |       |
| CHARACTER_SET_NAME         | varchar(32)         | YES  |     | NULL    |       |
| COLLATION_NAME             | varchar(32)         | YES  |     | NULL    |       |
| COLUMN_TYPE                | longtext            | NO   |     | NULL    |       |
| COLUMN_KEY                 | varchar(3)          | NO   |     |         |       |
| EXTRA                      | varchar(27)         | NO   |     |         |       |
| PRIVILEGES                 | varchar(80)         | NO   |     |         |       |
| COLUMN_COMMENT             | varchar(1024)       | NO   |     |         |       |
+---------------------------+---------------------+------+-----+---------+-------+
```

Demo 08: show all of the non-string columns for the tables in a_emp.

```
Select table_name, column_name, data_type
From information_schema.columns
Where table_schema= 'a_emp'
and data_type not in ('CHAR', 'VARCHAR');
+-------------+-------------+-----------+
| table_name  | column_name | data_type |
+-------------+-------------+-----------+
| departments | dept_id     | int       |
| departments | loc_id      | int       |
| employees   | emp_id      | int       |
| employees   | emp_mng     | int       |
| employees   | dept_id     | int       |
| employees   | hire_date   | date      |
| employees   | salary      | decimal   |
| employees   | job_id      | int       |
| jobs        | job_id      | int       |
| jobs        | min_salary  | decimal   |
| jobs        | max_salary  | decimal   |
| locations   | loc_id      | int       |
+-------------+-------------+-----------+
```

When you run a query such as the following, you will get data for all of your tables from all the databases where you have privileges. The display from this query is not limited to the current database. This may take some time to run- do not do this on a production system!

```
Select table_schema, table_name, table_type, table_rows
From information_schema.tables;
```

Demo 09: If we want to limit our retrieval of data to a specific database, we can include a filter on table_schema

```
Select table_schema, table_name, table_type, table_rows
From information_schema.tables
Where table_schema= 'a_emp';
+--------------+-------------+------------+------------+
| table_schema | table_name  | table_type | table_rows |
+--------------+-------------+------------+------------+
| a_emp        | departments | BASE TABLE |          7 |
| a_emp        | employees   | BASE TABLE |         17 |
| a_emp        | jobs        | BASE TABLE |          6 |
| a_emp        | locations   | BASE TABLE |          6 |
+--------------+-------------+------------+------------+
```

3. Using the Show command versus querying Information_Schema

For some tasks the show command is much quicker. Assume I am in the a_emp database and I have forgotten how I spelled a table name. It is a lot easier to enter the show tables command than to write the query to get this from Information_Schema. We get the same results; the column alias is different but the rows are the same.

Demo 10:

```
Show tables;
+-----------------+
| Tables_in_a_emp |
+-----------------+
| departments     |
| employees       |
| jobs            |
| location_types  |
| locations       |
+-----------------+
```

4. Advantages of using Information_Schema

For quick answers use the Show command but you need to be aware of some advantages of learning to use the Information_Schema tables.

- Information_Schema is an ANSI standard; MySQL seems to be a bit slower picking up some techniques but when it does it is closer to the ANSI standard. So techniques you use with Information_Schema should be standard. The time you spend learning the variations on Show helps you with MySQL; the time you spend learning the variations on Information_Schema helps you with all dbms.
- Show is a proprietary MySQL command; this means that is more likely to be changed in future versions of MySQL. If you are writing procedures that need to use metadata you are more likely to have a procedure that will survive new versions of MySQL if you use Information_Schema approach.
- If you are working with people who are more used to SQL Server they will get the Information_Schema right away; Oracle people will have to translate their view names- but the concepts will be the same. "Show" does not translate well.
- You can run select command against the tables in Information_Schema and use the filters, grouping, aggregate functions that you know (and love?). You can use joins to work with data from more than one of these tables.
- You can use the selects against Information_Schema inside procedures and functions passing in arguments for database object names. You can capture and manipulate the results of these queries.
- There is more stuff in the Information_Schema tables than the show command displays.

Demo 11:   We want to see which constraints we have on the tables in this database.

```
Select TC.table_name, TC.constraint_name, TC.constraint_type
From information_schema.table_constraints TC
Where TC.table_schema= 'a_emp';
+----------------+-----------------+-----------------+
| table_name     | constraint_name | constraint_type |
+----------------+-----------------+-----------------+
| departments    | PRIMARY         | PRIMARY KEY     |
| departments    | dept_loc_fk     | FOREIGN KEY     |
| employees      | PRIMARY         | PRIMARY KEY     |
| employees      | ssn_un          | UNIQUE          |
| employees      | dept_emp_fk     | FOREIGN KEY     |
| employees      | job_emp_fk      | FOREIGN KEY     |
| employees      | mng_emp_fk      | FOREIGN KEY     |
| jobs           | PRIMARY         | PRIMARY KEY     |
| location_types | PRIMARY         | PRIMARY KEY     |
| locations      | PRIMARY         | PRIMARY KEY     |
| locations      | loc_loc_type_fk | FOREIGN KEY     |
+----------------+-----------------+-----------------+
```

Demo 12: We could filter out the pk constraints.

```
Select TC.table_name, TC.constraint_name, TC.constraint_type
From information_schema.table_constraints TC
Where TC.table_schema= 'a_emp'
and constraint_type <> 'PRIMARY KEY';
+-------------+-----------------+-----------------+
| table_name  | constraint_name | constraint_type |
+-------------+-----------------+-----------------+
| departments | dept_loc_fk     | FOREIGN KEY     |
| employees   | ssn_un          | UNIQUE          |
| employees   | dept_emp_fk     | FOREIGN KEY     |
| employees   | job_emp_fk      | FOREIGN KEY     |
| employees   | mng_emp_fk      | FOREIGN KEY     |
| locations   | loc_loc_type_fk | FOREIGN KEY     |
+-------------+-----------------+-----------------+
```

Suppose we are concerned that we might have a table in a database that is missing a PK. For testing create the following table, which has no pk, in the employees database.

Demo 13:

```
Create table a_emp.vt_temp (col_id int);
```

Then run the query again

```
Select TC.table_name, TC.constraint_name, TC.constraint_type
From information_schema.table_constraints TC
Where TC.table_schema= 'a_emp';
```

We won't see this new table because the Information_Schema.table_constraints table only contains rows for table with constraints.

Demo 14: But we know how to do a "not found" query.

```
Select Concat(T.table_schema,'- - - ', T.table_name)
From information_schema.tables T
Where T.table_schema= 'a_emp'
and
  Concat(T.table_schema,'- - - ', T.table_name) NOT IN (
   Select Concat(TC.table_schema,'- - - ', TC.table_name)
   From information_schema.table_constraints TC
   Where TC.table_schema= 'a_emp'
   and TC.constraint_type = 'PRIMARY KEY'
   );
+-------------------------------------------------+
| Concat(T.table_schema,'- - - ', T.table_name) |
+-------------------------------------------------+
| a_emp- - - vt_temp                              |
+-------------------------------------------------+
```

Demo 15: demo

```
Select Concat(T.table_schema,'- - - ', T.table_name)
From information_schema.tables T
Where
   T.table_schema= 'a_emp'
and
  Concat(T.table_schema,'- - - ', T.table_name) NOT IN (
     Select Concat(TC.table_schema,'- - - ', TC.table_name)
     From information_schema.table_constraints TC
     Where TC.table_schema= 'a_emp'
     and  TC.constraint_type = 'PRIMARY KEY'
);
```

But this also returns views; the table Information_Schema.Tables includes both base tables and views. We can improve our query.

Demo 16:

```
Select Concat(T.table_schema,'- - - ', T.table_name)
From information_schema.tables T
Where
   T.table_schema= 'a_emp'
and
   T.TABLE_TYPE = 'BASE TABLE'
and
  Concat(T.table_schema,'- - - ', T.table_name) NOT IN (
     Select Concat(TC.table_schema,'- - - ', TC.table_name)
     From information_schema.table_constraints TC
     Where TC.table_schema= 'a_emp'
     and  TC.constraint_type = 'PRIMARY KEY'
);
```

Demo 17: drop this table

```
drop table a_emp.vt_temp;
```

5. Show options

Demo 18: The show commands have options that you might want to explore.

Show tables shows you the table names,

```
Show tables;
+------------------+
| Tables_in_a_emp |
+------------------+
| departments     |
| employees       |
| jobs            |
| location_types  |
| locations       |
+------------------+
```

This is a show table with more options.

```
Show full tables from a_testbed Like 'ddl%' ;
+----------------------------+------------+
| Tables_in_a_testbed (ddl%) | Table_type |
+----------------------------+------------+
| ddl_alter                    | BASE TABLE |
| ddl_dept                     | BASE TABLE |
| ddl_emp                      | BASE TABLE |
| ddl_emp_proj                 | BASE TABLE |
| ddl_proj                     | BASE TABLE |
+------------------------------+------------+

Show full tables from a_oe;
+-------------------------+------------+
| Tables_in_a_oe          | Table_type |
+-------------------------+------------+
| credit_ratings          | BASE TABLE |
| cust_orders             | VIEW       |
| customers               | BASE TABLE |
| custreportgoodcredit_01 | VIEW       |
| order_details           | BASE TABLE |
| order_headers           | BASE TABLE |
| ordreport_01            | VIEW       |
| ordreport_02            | VIEW       |
| shipping_modes          | BASE TABLE |
+-------------------------+------------+
```