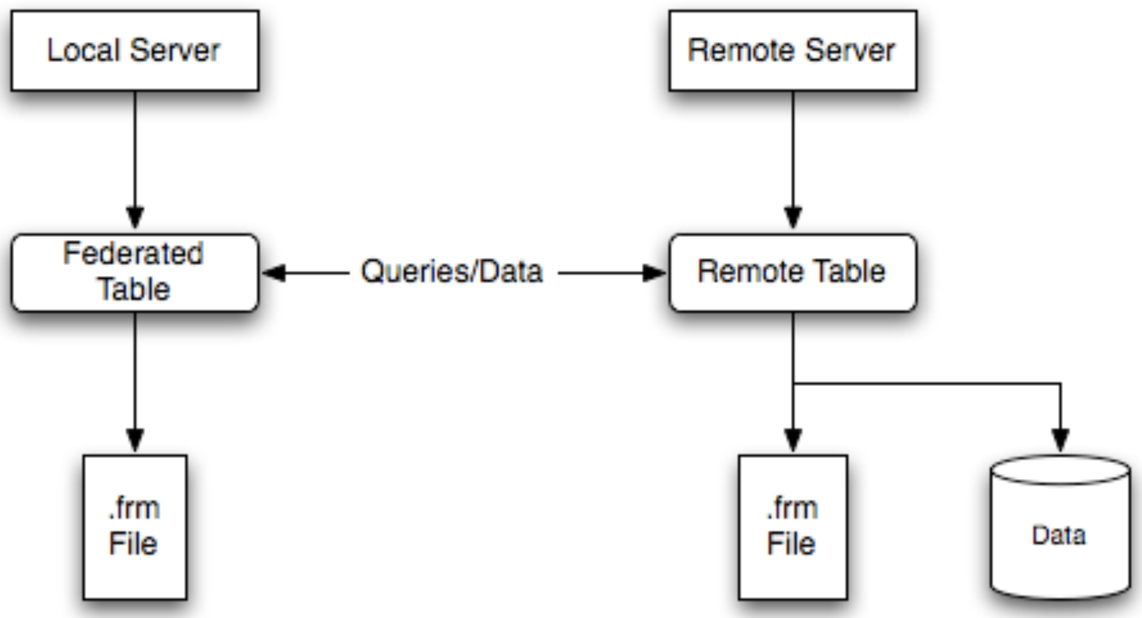
COMP283-Lecture 9
Applied Database Management

Introduction	
DB Linking	MySQL Federated Storage Engine
Migration	Reminder: Views and Stored Procedures
	Manual Migration

COMP283-Lecture 9

DB Linking: MySQL Federated Storage Engine

- Database Linking in MySQL is via MySQL’s Federated Storage Engine.
- Only supports linking between MySQL databases.
- The Federated Storage Engine must be included in the MySQL server build.
- When configured, it creates a local table structure that replicates the remote table.



COMP283-Lecture 9

DB Linking: MySQL Federated Storage Engine

- Steps to create a Federated table:
  - Create a table on the remote server (or use existing table)
  - Create an identical table on the local server
  - Add connection info to local table to link to remote, either:
    - Use a CONNECTION statement with details of remote server connection
    - Use an existing connection (from a CREATE SERVER statement)

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

COMP283-Lecture 9

DB Linking: MySQL Federated Storage Engine

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

- scheme: A recognized connection protocol.
- user_name: The user name for the connection.
  - User must have been created on the remote server, and must have suitable privileges to perform the required actions on the remote table.
- password: (Optional)
- host_name: The host name or IP address of the remote server.
- port_num: (Optional) The port number for the remote server.
- db_name: The name of the database holding the remote table.
- tbl_name: The name of the remote table.

COMP283-Lecture 9

DB Linking: MySQL Federated Storage Engine

```
CREATE TABLE federated_table (
    id      INT(20) NOT NULL AUTO_INCREMENT,
    name    VARCHAR(32) NOT NULL DEFAULT '',
    other   INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY   (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='mysql://fed_user@remote_host:9306/federated/
test_table';
```

```
CREATE SERVER s
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
```

```
CREATE TABLE t (s1 INT) ENGINE=FEDERATED CONNECTION='s';
```

COMP283-Lecture 9

DB Linking: MySQL Federated Storage Engine

- One federated table can point to another one (but beware of loops)
- The local “copy” of the table does not support indexes
  - (data is actually handled remotely - remote table can have indexes though)
  - Query that requires a full table scan will have to retrieve all rows from the remote server
  - Leads to poor performance
- Creating indexes at table creation time may not be supported
  - (best to create them separately)
- ALTER TABLE is not supported
- TRANSACTIONS not supported (though the remote table can).
- DROP TABLE only drops the local table (not the remote)

COMP283-Lecture 9

Stored Procedures & Views - reminder

- Views are stored queries. Treated like tables – they can be indexed and queried.
- Views are useful to restricting access to a limited subset of data attributes (can be across multiple tables)
- Views are useful to tunnel through security.
- Example – create a view:

```
CREATE VIEW vwRockMusic AS
SELECT strArtist, strAlbum, strSong FROM
tblAlbums
WHERE tblAlbums.Genre = “Rock”;
```

COMP283-Lecture 9

Stored Procedures & Views - reminder

- Stored procedures are a sequence of executable SQL statements, compiled and saved within the database.
- A stored procedure is often a saved query – the difference between a Stored Procedure and a view is that the stored procedure can have parameters passed to it and is more dynamic. i.e. You can pass query criteria (WHERE clause parameters) to the stored procedure.
- Can simplify client applications and avoid the need for changes to it if DB structure changes needed. How?
- The principal* must have, directly or inherited, Execute permission on the Stored Procedure.

*user or application program

COMP283-Lecture 9

Views and Stored Procedures

- Relevant to situation of merged databases
- Both can be used to hide structure of db from client programs
- Possible to make a hybrid db but client applications see original db structure instead

COMP283-Lecture 9

Migrating Databases - Manual Migration

- Examine every single table in your DB schema
- Find all tables that can be commonly migrated (common rules, same behaviour)
- Find all tables that have some table-specific rules
- Create backup dump
- Create TODO list
- Execute migration on test server
- Find some ways to test it
- Execute migration production server

http://blog.brunoraljic.com/how-to-merge-two-mysql-databases-manually-part-1/

COMP283-Lecture 9

Migrating Databases - Manual Migration

- Examine every single table in your DB schema
  - In order to successfully migrate your DB you'll need to know everything about it.
  - Include indexes, unique keys etc.
  - Involve more than one person in this phase if possible.
  - Take longer doing the analysis so you don't end up with errors during the migration or (worse case) after it.

COMP283-Lecture 9

Migrating Databases - Manual Migration

- Find all tables that can be commonly migrated (common rules, same behaviour)
  - This group is easier to migrate since all you need is to identify all the tables belonging to this group.
  - e.g. Let's say you have tables *products*, *orders* and many to many table *orders_products*.
  - You have products in both **dbA** and **dbB** databases, but those products are not the same, nor the orders (but they can have the same ID). Since the product_id is unique, you can't just simply move products from **dbB** to **dbA** (error, duplicate product_id). You need to update product_id in **dbB** and then move it to **dbA**.
  - You need to update product_id in both products table and orders_products (and in all other places where you can find product_id.

COMP283-Lecture 9

Migrating Databases - Manual Migration

- Find all tables that have some table-specific rules
  - Table *users*. Lets say you have users in **dbA** and **dbB**, but some of them are the same users (same person, same username but different user_id).
  - It's not possible just to increment values in user_id field and move them like in first group. You'll end up with duplicate users. You won't be able to do it at all if for example the username is unique.
  - For this table you have two rules: First you need to take care of duplicate users (adapt their user_id from one db to another). After that you increment user_id for the other users and move them freely.
  - You may have another table, some configurations for example where you will need only to adapt the values to a new db.

COMP283-Lecture 9

Migrating Databases - Manual Migration

- Create backup dump
  - Migration is high-risk. Prepare backups of both DBs just in case
- Create TODO list
  - Note down every single step you need to perform.
- Execute migration on test server
  - Use fresh dumps from the main DB.
  - Watch for errors.

COMP283-Lecture 9

Migrating Databases - Manual Migration

- Find some ways to test it
    - Simple and complex tests.
- Execute migration production server

COMP283-Lecture 9

Conclusions

- Talked about MySQL Federated Storage Engine
- Migration
  - Use of Views and Stored Procedures
  - Manual Migration