

PostgreSQL at the centre of your dataverse

PostgreSQL Conference Europe 2011

Presented by Dave Page
19th October 2011

Who is Dave Page?

- ▶ Lead developer of pgAdmin
- ▶ postgresql.org webmaster and sysadmin
- ▶ Senior Software Architect at EnterpriseDB, responsible for:
 - PostgreSQL and component Installers
 - Postgres Enterprise Manager
 - Postgres Plus Standard Server
 - PostgreSQL Solution Pack
- ▶ Member of
 - PostgreSQL Core Team
 - PostgreSQL Europe Board of Directors
 - PostgreSQL Community Association of Canada Board of Directors

Centre of my dataverse?

- Embarrassing “marketing moment”

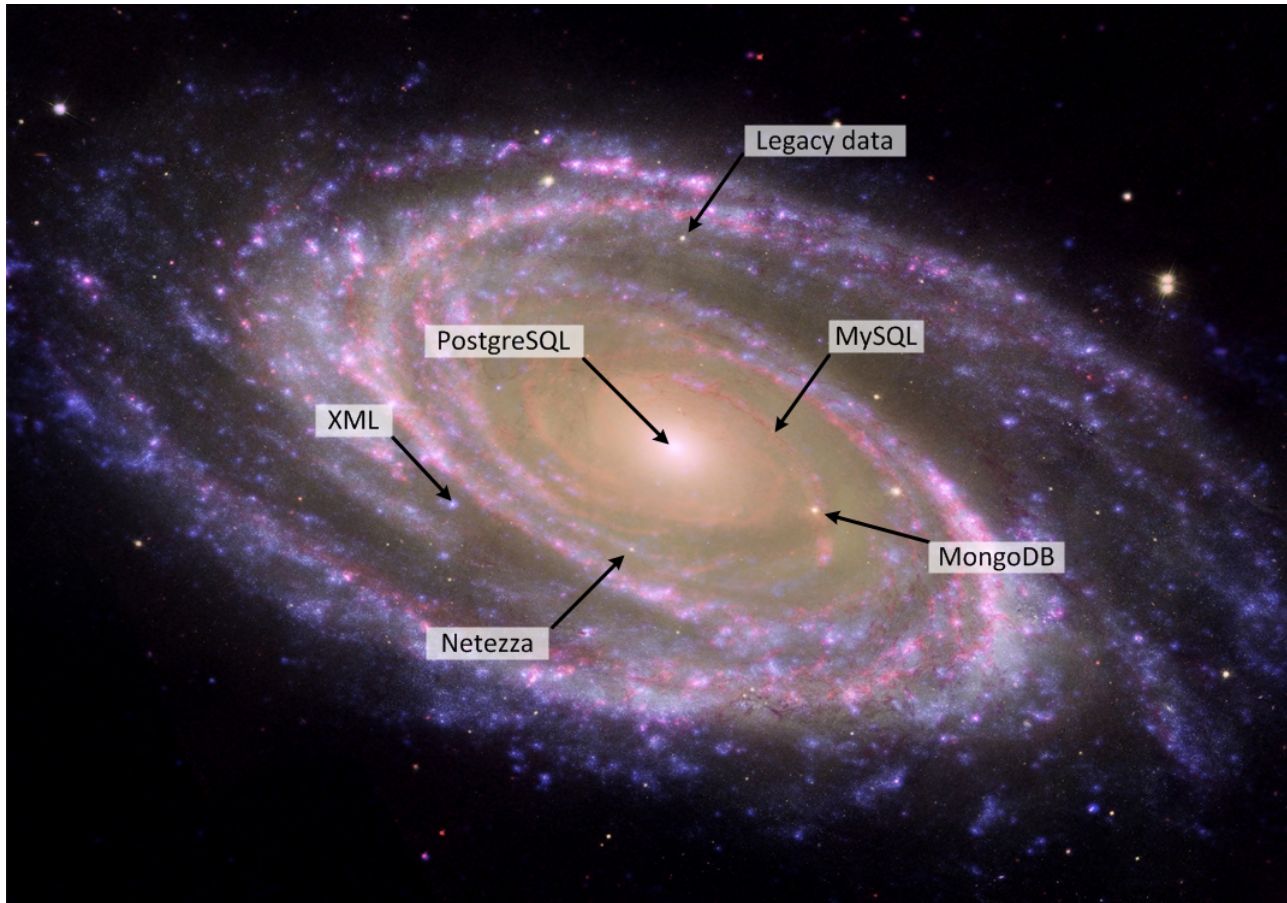


Image credit: NASA/JPL-Caltech/ESA/Harvard-Smithsonian CfA

Why?

- ▶ Application integration
- ▶ Cross database/application reporting
- ▶ Data migration
- ▶ Data sharding
- ▶ Database federation



About SQL/MED

SQL/MED

- ▶ SQL standard for Management of External Data
- ▶ Defined in ISO/IEC 9075-9:2003
- ▶ Specifies how external data sources are accessed from an SQL database

Before SQL/MED

▶ Import data into [temporary] tables:

- COPY
- psql scripts
- Custom loader programs

▶ Stored functions/procedures:

- PL/Proxy – primarily intended for advanced partitioning
- DBI:Link – Perl project that uses views, functions and rules to make remote tables appear local
- dblink – Contrib module allowing access to remote PostgreSQL servers

Disadvantages

- ▶ Custom loader/management code may be required
- ▶ May need need to run batch tasks for incremental imports
- ▶ Data may not be presented as relations, but functions
- ▶ Custom stored functions may be required
- ▶ Different data sources may have different interfaces

With SQL/MED

- ▶ Data is presented to the user like any other table or relation
- ▶ Standardised set of object types and configuration commands used to setup and configure a remote data source
- ▶ Integrated credential management ensures usernames and passwords can be managed securely, for each role
- ▶ Deep integration with PostgreSQL's planner for improved query planning and optimisation

Current Limitations

- ▶ Data is read only
- ▶ Planner limitations:
 - No defined API for qual (WHERE clause) push down
 - No join push down
- ▶ API deficiency: no simple way to pass complex data from the planner callback to the executor callbacks



Using SQL/MED

SQL Objects – Foreign Data Wrapper

- ▶ Also known as an FDW
- ▶ Defines the “type” or remote data source
- ▶ Consists of:
 - Handler function
 - Validator function (optional)
- ▶ Refers to both the SQL object, and less formally, the binary code that implements the interface to the remote data source

Relational DBMS FDWs

▶ MySQL

- Written by Dave Page
- https://github.com/dpage/mysql_fdw

▶ ODBC

- Written by Zheng Yang (GSoC project)
- https://github.com/ZhengYang/odbc_fdw

▶ Oracle

- Written by Laurenz Albe
- <http://pgfoundry.org/projects/oracle-fdw>

NoSQL FDWs

▶ CouchDB

- Written by Zheng Yang (GSoC project)
- https://github.com/ZhengYang/couchdb_fdw

▶ Redis

- Written by Dave Page
- Includes experimental qual pushdown for key values
- https://github.com/dpage/redis_fdw

File FDWs

► CSV

- Included as an extension with PostgreSQL 9.1
- <http://www.postgresql.org/docs/9.1/static/file-fdw.html>

► Text Array

- Written by Andrew Dunstan
- Presents [ragged] CSV files as text[] data
- https://github.com/adunstan/file_text_array_fdw

Other FDWs

▶ LDAP

- Written by Dickson S. Guedes
- https://github.com/guedes/ldap_fdw

▶ Twitter

- Written by Hitoshi Harada
- https://github.com/umitanuki/twitter_fdw

Creating an FDW

► Create the functions:

```
CREATE FUNCTION mysql_fdw_handler()  
    RETURNS fdw_handler  
    AS '$libdir/mysql_fdw'  
    LANGUAGE C STRICT;
```

```
CREATE FUNCTION mysql_fdw_validator(text[], oid)  
    RETURNS void  
    AS '$libdir/mysql_fdw'  
    LANGUAGE C STRICT;
```

Creating an FDW

- Create the FDW object:

```
CREATE FOREIGN DATA WRAPPER mysql_fdw  
    HANDLER mysql_fdw_handler  
    VALIDATOR mysql_fdw_validator;
```

Creating an FDW

- Or... use PostgreSQL 9.1's EXTENSIONs mechanism:

```
CREATE EXTENSION mysql_fdw;
```

SQL Objects – Foreign Server

- ▶ Defines a specific “server” or source of data, for example:
 - A PostgreSQL database
 - A MySQL server
 - A Twitter account
 - A delimited file
- ▶ Each server uses one FDW. One FDW supports multiple servers.

Creating a Foreign Server

- Create the foreign server object:

```
CREATE SERVER mysql_svr  
    FOREIGN DATA WRAPPER mysql_fdw  
    OPTIONS (address '127.0.0.1', port '3306');
```

- `mysql_fdw` supports the following server options:
 - *address* – The hostname or IP address of the MySQL server (default: 127.0.0.1)
 - *port* – The port number that the MySQL server is listening on (default: 3306)

SQL Objects – Foreign Table

- ▶ Defines a “table” representing data on a foreign server, e.g:
 - A PostgreSQL table or view
 - A delimited file
 - An SQL query against a MySQL database
- ▶ Each table uses one foreign server. Each server supports multiple tables
- ▶ The Foreign Table object may be used in PostgreSQL as a read-only table, e.g:

```
SELECT *  
    FROM foreign f JOIN local l ON (f.id = l.id)  
    ORDER BY f.name
```

Creating a Foreign Table

- ▶ Create the foreign table object:

```
CREATE FOREIGN TABLE tbl (c1 text, c2 text)  
    SERVER mysql_svr;
```

- ▶ `mysql_fdw` supports the following table options:

- *database* – The name of the MySQL database (optional)
- *query* – An SQL query to return the desired data
- *table* – The name of a table (quoted and qualified if needed) containing the desired data

Note: Either *table* or *query* must be specified, but not both.

SQL Objects – User Mapping

- ▶ Defines security information used to connect to a foreign server
- ▶ Other options may be specified, if the FDW supports it
- ▶ Each user mapping applies to one server. Each server supports multiple user mappings
- ▶ User mappings may be defined for “PUBLIC” or individual roles

Creating a User Mapping

- ▶ Create the user mapping object:

```
CREATE USER MAPPING FOR dpage  
    SERVER mysql_svr  
    OPTIONS (username 'dpage', password 'Foo');
```

- ▶ `mysql_fdw` supports the following user mapping options:

- *username* – the username to use to connect to the MySQL server
- *password* – the password corresponding to the username specified



Writing FDWs

Requirements – SQL Functions

▶ Handler function

- Must be written in C
- Provides pointers to callback functions in the FDW

▶ Validator function

- Must be written in C
- Optional
- Validates options for:
 - *Foreign Data Wrapper*
 - *Foreign Servers*
 - *Foreign Tables*
 - *User Mappings*

Handler Function (pseudo code)

```
/*
 * Foreign-data wrapper handler function: return a struct with pointers
 * to my callback routines.
 */
Datum
mysql_fdw_handler(PG_FUNCTION_ARGS)
{
    FdwRoutine *fdwroutine = makeNode(FdwRoutine);

    fdwroutine->PlanForeignScan = mysqlPlanForeignScan;
    fdwroutine->ExplainForeignScan = mysqlExplainForeignScan;
    fdwroutine->BeginForeignScan = mysqlBeginForeignScan;
    fdwroutine->IterateForeignScan = mysqlIterateForeignScan;
    fdwroutine->ReScanForeignScan = mysqlReScanForeignScan;
    fdwroutine->EndForeignScan = mysqlEndForeignScan;

    PG_RETURN_POINTER(fdwroutine);
}
```

Requirements – SQL Functions

▶ Handler function

- Must be written in C
- Provides pointers to callback functions in the FDW

▶ Validator function

- Must be written in C
- Optional
- Validates options for:
 - *Foreign Data Wrapper*
 - *Foreign Servers*
 - *Foreign Tables*
 - *User Mappings*

Validator Function (pseudo code)

```
/*
 * This tends to be a long and boring function, so here's some pseudo code
 * instead. See https://github.com/dpage/mysql\_fdw/blob/master/mysql\_fdw.c
 * for a working example.
 */
Datum
mysql_fdw_validator(PG_FUNCTION_ARGS)
{
    List *options_list = untransformRelOptions(PG_GETARG_DATUM(0));
    Oid catalog = PG_GETARG_OID(1); /* Object type - table, user mapping etc. */

    foreach(option, options_list)
    {
        if(!mysqlIsValidOption(option, catalog)
            ereport(ERROR, (errcode(ERRCODE_FDW_INVALID_OPTION_NAME),
                          errmsg("invalid option \"%s\"", option->name)));

        /* If the option is valid, we may also want to validate the value... */
    }
}
```

Requirements – API Functions

▶ PlanForeignScan

- Plans the foreign scan on the remote server
- May or may not actually do anything remotely
- Returns cost estimates to the planner

▶ ExplainForeignScan

- Optionally adds additional data to EXPLAIN output

▶ BeginForeignScan

- Performs initialisation required for the foreign scan

PlanForeignScan (pseudo code)

```
static FdwPlan *
mysqlPlanForeignScan(Oid foreigntableid, PlannerInfo *root, RelOptInfo *baserel)
{
    /* Connect to the remote server */
    MYSQL *conn = mysql_connect(server, port, username, password);

    /* Get statistics for the remote scan */
    rows = mysql_query("SELECT count(*) FROM table");

    /* Set the number of rows in the relation */
    baserel->rows = rows;

    /* Calculate a startup cost for the scan */
    fdwplan->startup_cost = 10;
    if (!IsLocal(server))
        fdwplan->startup_cost += 15;

    /* Finally, calculate the total cost */
    fdwplan->total_cost = rows + fdwplan->startup_cost;

    return fdwplan;
}
```


Requirements – API Functions

▶ PlanForeignScan

- Plans the foreign scan on the remote server
- May or may not actually do anything remotely
- Returns cost estimates to the planner

▶ ExplainForeignScan

- Optionally adds additional data to EXPLAIN output

▶ BeginForeignScan

- Performs initialisation required for the foreign scan

ExplainForeignScan (pseudo code)

```
static void
mysqlExplainForeignScan(ForeignScanState *node, ExplainState *es)
{
    /* Give some possibly useful info about startup costs, if needed */
    if (es->costs)
    {
        if (IsLocal(server))
            ExplainPropertyLong("Local server startup cost", 10, es);
        else
            ExplainPropertyLong("Remote server startup cost", 25, es);
    }
}
```

Requirements – API Functions

- ▶ PlanForeignScan
 - Plans the foreign scan on the remote server
 - May or may not actually do anything remotely
 - Returns cost estimates to the planner
- ▶ ExplainForeignScan
 - Optionally adds additional data to EXPLAIN output
- ▶ BeginForeignScan
 - Performs initialisation required for the foreign scan

BeginForeignScan (pseudo code)

```
static void
mysqlBeginForeignScan(ForeignScanState *node, int eflags)
{
    /* Connect to the remote server */
    MYSQL *conn = mysql_connect(server, port, username, password);

    /* Build the remote SQL query */
    query = sprintf(query, "SELECT * FROM %s", table);

    /* Stash away the state info for use by other API functions */
    festate = (MySQLFdwExecutionState *) malloc(sizeof(MySQLFdwExecutionState));
    node->fdw_state = (void *) festate;

    festate->conn = conn;
    festate->query = query;

    /* This will store the remote query result */
    festate->result = NULL;
}
```

Requirements – API Functions

▶ IterateForeignScan

- Begin executing the foreign scan on first invocation
- Returns one tuple per call

▶ ReScanForeignScan

- Reset the scan to start again from the beginning

▶ EndForeignScan

- Complete the foreign scan
- Release resources

IterateForeignScan (pseudo code)

```
static TupleTableSlot *
mysqlIterateForeignScan(ForeignScanState *node)
{
    MySQLFdwExecutionState *festate = (MySQLFdwExecutionState *) node->fdw_state;
    TupleTableSlot *slot = node->ss.ss_ScanTupleSlot;

    /* Execute the query, if required */
    if (!festate->result)
        festate->result = mysql_query(festate->conn, festate->query);

    /* Get the next row from the remote server */
    row = mysql_fetch_row(festate->result);

    /* If there's a row, convert to a tuple and store it in the slot */
    if (row)
    {
        ConvertMySqlRowToTuple(row, tuple);
        ExecStoreTuple(tuple, slot);
    }

    return slot;
}
```

Requirements – API Functions

- ▶ IterateForeignScan
 - Begin executing the foreign scan on first invocation
 - Returns one tuple per call

- ▶ ReScanForeignScan
 - Reset the scan to start again from the beginning

- ▶ EndForeignScan
 - Complete the foreign scan
 - Release resources

ReScanForeignScan (pseudo code)

```
static void
mysqlReScanForeignScan(ForeignScanState *node)
{
    MySQLFdwExecutionState *festate = (MySQLFdwExecutionState *) node->fdw_state;

    /* Reset the scan so it can start over */
    mysql_free_result(festate->result);
    festate->result = NULL;
}
```


Requirements – API Functions

- ▶ IterateForeignScan
 - Begin executing the foreign scan on first invocation
 - Returns one tuple per call
- ▶ ReScanForeignScan
 - Reset the scan to start again from the beginning
- ▶ EndForeignScan
 - Complete the foreign scan
 - Release resources

EndForeignScan (pseudo code)

```
static void
mysqlReScanForeignScan(ForeignScanState *node)
{
    MySQLFdwExecutionState *festate = (MySQLFdwExecutionState *) node->fdw_state;

    /* Cleanup the query string */
    pfree(festate->query);
    festate->query = NULL;

    /* Cleanup the scan result */
    mysql_free_result(festate->result);
    festate->result = NULL;

    /* Cleanup the remote connection */
    mysql_close(festate->conn);
    festate->conn = NULL;

    /* Cleanup the FDW state */
    pfree(festate);
    festate = NULL;
}
```



Using FDWs

Create the Objects

```
raptor:pgsql91 dpage$ bin/psql fdw  
psql (9.1.0)  
Type "help" for help.
```

```
fdw=# CREATE EXTENSION mysql_fdw;  
CREATE EXTENSION
```

Create the Objects

```
raptor:pgsql91 dpage$ bin/psql fdw
psql (9.1.0)
Type "help" for help.
```

```
fdw=# CREATE EXTENSION mysql_fdw;
CREATE EXTENSION
fdw=# CREATE SERVER mysql_svr
fdw-#      FOREIGN DATA WRAPPER mysql_fdw
fdw-#      OPTIONS (address '127.0.0.1', port '3306');
CREATE SERVER
```

Create the Objects

```
raptor:pgsql91 dpage$ bin/psql fdw
psql (9.1.0)
Type "help" for help.
```

```
fdw=# CREATE EXTENSION mysql_fdw;
CREATE EXTENSION
fdw=# CREATE SERVER mysql_svr
fdw-#      FOREIGN DATA WRAPPER mysql_fdw
fdw-#      OPTIONS (address '127.0.0.1', port '3306');
CREATE SERVER
fdw=# CREATE FOREIGN TABLE employees (
fdw(#      id integer,
fdw(#      name text,
fdw(#      address text)
fdw-#      SERVER mysql_svr
fdw-#      OPTIONS (table 'hr.employees');
CREATE FOREIGN TABLE
```

Create the Objects

```
fdw=# CREATE EXTENSION mysql_fdw;
CREATE EXTENSION
fdw=# CREATE SERVER mysql_svr
fdw-#      FOREIGN DATA WRAPPER mysql_fdw
fdw-#      OPTIONS (address '127.0.0.1', port '3306');
CREATE SERVER
fdw=# CREATE FOREIGN TABLE employees (
fdw(#      id integer,
fdw(#      name text,
fdw(#      address text)
fdw-#      SERVER mysql_svr
fdw-#      OPTIONS (table 'hr.employees');
CREATE FOREIGN TABLE
fdw=# CREATE FOREIGN TABLE overtime_2010 (
fdw(#      id integer,
fdw(#      employee_id integer,
fdw(#      hours integer)
fdw-#      SERVER mysql_svr
fdw-#      OPTIONS (query 'SELECT id, employee_id, hours FROM hr.overtime WHERE year
= 2010;');
CREATE FOREIGN TABLE
```

Run a Query

```
fdw=# CREATE FOREIGN TABLE employees (  
fdw(#      id integer,  
fdw(#      name text,  
fdw(#      address text)  
fdw-#      SERVER mysql_svr  
fdw-#      OPTIONS (table 'hr.employees');  
CREATE FOREIGN TABLE  
fdw=# CREATE FOREIGN TABLE overtime_2010 (  
fdw(#      id integer,  
fdw(#      employee_id integer,  
fdw(#      hours integer)  
fdw-#      SERVER mysql_svr  
fdw-#      OPTIONS (query 'SELECT id, employee_id, hours FROM hr.overtime WHERE year  
= 2010;');  
CREATE FOREIGN TABLE  
fdw=# SELECT * FROM employees;  
 id |      name      |      address  
----+-----+-----  
  1 | Dave Page     | 123 High Street, Oxford  
  2 | John Smith    | 54 Church Lane, Glasgow  
  3 | Fred Bloggs   | 3b Grouse Court, Birmingham  
(3 rows)
```


Explain a Query

```
fdw=# CREATE FOREIGN TABLE overtime_2010 (  
fdw(#      id integer,  
fdw(#      employee_id integer,  
fdw(#      hours integer)  
fdw-#      SERVER mysql_svr  
fdw-#      OPTIONS (query 'SELECT id, employee_id, hours FROM hr.overtime WHERE year  
= 2010;');
```

```
CREATE FOREIGN TABLE
```

```
fdw=# SELECT * FROM employees;
```

id	name	address
1	Dave Page	123 High Street, Oxford
2	John Smith	54 Church Lane, Glasgow
3	Fred Bloggs	3b Grouse Court, Birmingham

(3 rows)

```
fdw=# EXPLAIN SELECT * FROM employees;  
          QUERY PLAN
```

```
-----  
Foreign Scan on employees  (cost=10.00..13.00 rows=3 width=68)  
  Local server startup cost: 10  
  MySQL query: SELECT * FROM hr.employees  
(3 rows)
```

Run another Query

```
fdw=# SELECT * FROM employees;
```

id	name	address
1	Dave Page	123 High Street, Oxford
2	John Smith	54 Church Lane, Glasgow
3	Fred Bloggs	3b Grouse Court, Birmingham

(3 rows)

```
fdw=# EXPLAIN SELECT * FROM employees;  
          QUERY PLAN
```

```
-----  
Foreign Scan on employees  (cost=10.00..13.00 rows=3 width=68)  
  Local server startup cost: 10  
  MySQL query: SELECT * FROM hr.employees  
(3 rows)
```

```
fdw=# SELECT e.id, e.name, hours FROM employees e LEFT OUTER JOIN overtime_2010 o  
ON (e.id = o.employee_id);
```

id	name	hours
1	Dave Page	23
2	John Smith	
3	Fred Bloggs	14

(3 rows)

Explain another Query

```
id |      name      | hours
----+-----+-----
 1 | Dave Page      |    23
 2 | John Smith     |
 3 | Fred Bloggs    |    14
```

(3 rows)

```
fdw=# EXPLAIN SELECT e.id, e.name, hours FROM employees e LEFT OUTER JOIN
overtime_2010 o ON (e.id = o.employee_id);
```

QUERY PLAN

```
-----
Nested Loop Left Join  (cost=20.00..25.09 rows=3 width=40)
  Join Filter: (e.id = o.employee_id)
    -> Foreign Scan on employees e  (cost=10.00..13.00 rows=3 width=36)
        Local server startup cost: 10
        MySQL query: SELECT * FROM hr.employees
    -> Materialize  (cost=10.00..12.01 rows=2 width=8)
        -> Foreign Scan on overtime_2010 o  (cost=10.00..12.00 rows=2 width=8)
            Local server startup cost: 10
            MySQL query: SELECT id, employee_id, hours FROM hr.overtime WHERE
year = 2010;
(9 rows)
```



Questions?

Email: dpag@pgadmin.org

Twitter: [@pgsnake](https://twitter.com/pgsnake)



Thank you!

Session feedback:

<http://2011.pgconf.eu/feedback>