```php
<?php
/** Paging Ajax responder for Org Chart Viewer
 *
 *  Code and data for a quick LAMP stack example demo. Demo's a path
 *  table producing timely paged results for an org chart display of a
 *  20000 node tree.
 *
 *  The code here is acting entirely as a Ajax responder for a jqGrid
 *  object on the display page, index.html. Documentation for jqGrid
 *  can be found at:
 *  http://www.trirand.com/jqgridwiki/doku.php?id=wiki:options
 *
 *  @author Grant Tegtmeier <Grant@Tegt.com>
 *  @package closure
 *  @copyright just say no
 */

/** Safely fetch a $_GET value in strict form
 *
 *  Provides a quick central check that GET parameters are defined
 *  avoiding strict mode issues and unexpected coersions.
 *
 * @param string $name the key value in the $_GET array.
 * @param mixed $default the value used if key is absent.
 */
function safeGet($name, $default='') {
    if (! array_key_exists($name, $_GET))
        return $default;
    return $_GET[$name];
}

/** jqGrid Error msg
 *
 *  This is a last ditch error display. It simply passes the message
 *  out in the xhr stream, which will cause the json parser in jqGrid
 *  to fail. A JS function defined in index.html the loadError option
 *  in index.html's jqGrid parms pops an alert for the user. In a full
 *  application the function defined to redirect to another page or do
 *  something more helpful.
 */
function errOut($msg) {
    echo 'ERROR: '.$msg;
    exit (0);
}
error_reporting(E_ERROR | E_PARSE);

/** Quckie sql error reflector */
function errCheck($result) {
    global $database;
    if (! $result)
        errOut('SQL error: '.$database->error);
}

/* This is the only php file using this database so inline for
 * simplicity.
 */
$database = new mysqli('db.tegt.com', 'readtegtdb', 'read-access', 'closure');
if ($database->connect_errno)
```

```
59.     errOut('Database connection error '.$database->connect_errno.': '.$database-
    >connect_error);
60. 
61. // Obtain and condition GET parameters from jsGrid ajax URL
62. $pageNum = max(1, safeGet('page', 1));
63. $pageRows = min(100, safeGet('rows', 100));
64. $sortBy = safeGet('sidx', 'enum');
65. 
    $sortBy = in_array($sortBy, array('enum', 'ename', 'bname', 'dist', 'subs'),true)?
66.             $sortBy: 'enum';
67. $sortOrder = (safeGet('sord', 'asc') <> 'desc')?  // only two choices
68.             'asc': 'desc';
69. 
70. // Quick query for total number of employees (need for page count only)
71. $result = $database->query('SELECT COUNT(*) AS count FROM employees');
72. errCheck($result);
73. $totalRows = $result->fetch_object()->count;
74. 
75. // Use count to calc pages and condition current page num
76. $totalPages = ceil($totalRows/$pageRows);
77. $pageNum = min($totalPages, max(1, $pageNum)); // clamp to range 1..total
78. 
79. /** A query to find them all. The major speed advantage here is the
80.  *  use of a complete paths table also called a "closure" table. See
81.  *  smonkey.sql for the INSERT trigger that maintains this table.
82.  *
83.  *  The example here works from a stitc preset table so the paths
84.  *  table is only built once and only the insert trigger has been
85.  *  tested.
86.  *
87.  *  A paths row has only three numbers. The boss id, the employee id
88.  *  and the distance between them. BUT it has a row for every path in
89.  *  the tree including the zero length self reference path.
90.  *
91.  *  The first join adds the boss' name to the row.
92.  *
93.  *  The second join looks up the distance to the CEO by directly
94.  *  locating the path from the CEO to that employee.
95.  *
96.  *  The third join simply adds a row for every path where the employee
97.  *  is at the top. The group by then counts the number of people that
98.  *  were joined to him at any depth. (Less 1 for the path to self.)
99.  *
100.  *  Paging and sorting are altered by using the four variables
101.  *  calculated from the employee table size and the GET parameters set
102.  *  by the jqGrid scripts.
103.  */
104. $query = '
105. SELECT emp.id AS enum, emp.name AS ename, boss.name AS bname,
106.     span.distance AS dist, COUNT(*)-1 AS subs
107.     FROM employees AS emp
108.         JOIN employees AS boss ON emp.bossId=boss.id
109.         JOIN empPaths AS span ON span.bossId=1 AND span.empId=emp.id
110.         JOIN empPaths AS subs ON emp.id=subs.bossId
111.     GROUP BY enum
112.     ORDER BY '.$sortBy.' '.$sortOrder
113.  .' LIMIT '.$pageRows.' OFFSET '.$pageRows*($pageNum-1);
114. 
```

```php
$result = $database->query($query);
errCheck($result);

/** Construct the reply object. The AJAX requester is expecting a
 *  structured response to set the new page values and fill the table
 *  cells. By mimicing the structure with PHP objects and arrays the
 *  reply can be directly encoded into json with a single call.
 */
$reply->page = $pageNum;
$reply->total = $totalPages;
$reply->records = $totalRows;

for ($i = 0; $ro = $result->fetch_object(); $i++) {
    $reply->rows[$i]['id'] = $ro->enum;
    $reply->rows[$i]['cell'] =
        array ($ro->enum, $ro->ename, $ro->bname, $ro->dist, $ro->subs);
}

// Encode it in json and send it out.
echo json_encode($reply);
?>
```