To initiate the whole flow, the SQL-like string input is transformed via *lex* and *yacc* into the main `FQLStatement`'s `$select` expression array and the `$where` expression. `FQLStatement`'s `evaluate()` function returns the objects we've requested. The main statement evaluation loop in Example 6-21 goes through the following steps in this simple high-level sequence:

1. Get all constraints on indexes of the rows we wish to return. For example, when selecting on the `user` table, these would be the UIDs we want to query. If we were looking at an events table indexed by time, say, these would be the boundary times.
2. Translate these to the canonical IDs of the table. The `user` table is also queryable by field `name`; if an FQL expression used `name`, this function would use an internal `user_name ->user_id` lookup function.
3. For each candidate ID, see if it matches the RHS expression clause (Boolean logic, comparisons, "IN" operations, etc.). If not, toss it out.
4. Evaluate each expression (in our case, fields in the `SELECT` clause), and create an XML element of the form *<COL_NAME>COL_VALUE</COL_NAME>*, where *COL_NAME* is the name of the field in the `FQLTable`, and *COL_VALUE* is the result of the evaluation of the field through its corresponding `FQLField`'s `evaluate` function.

*EXAMPLE 6-21. The FQL's main evaluation flow*

```
class FQLStatement {

  // contains the following members:
  // $select: array of FQLExpressions from the SELECT clause of the query
  //  corresponding to, say, "books", "pic", and "name"
  // $from: FQLTable object for the source table
  // $where: FQLExpression containing the constraints for the query.
  // $user, $app_id: calling user and app_id

  public function __construct($select, $from, $where, $user, $app_id) { ... }

  // A listing of all known tables in the FQL system.
  public static $tables = array(
    'user'        => 'FQLUserTable',
    'friend'      => 'FQLFriendTable',
  );

  // returns XML elements to be translated to service output
  public function evaluate() {

    // based on the WHERE clause, we first get a set of query expressions that
    // represent the constraints on values for the indexable columns contained
    // in the WHERE clause

    // Get all "right hand side" (RHS) constants matching ids (e.g. X, in 'uid = X')
    $queries = $this->where->get_queries();
```