```
    // Match to the row's index. If we were using 'name' as an alternative index
    // to the user table, we would transform it here to the uid.
    $index_ids = $this->from_table->get_ids_for_queries($queries);

    // filter the set of ids by the WHERE clause and LIMIT params
    $result_ids = array();

    foreach ($ids as $id) {
      $where_result = $this->where->evaluate($id);

        // see if this row passes the 'WHERE' constraints
        // is not restricted by privacy
        if ($where_result && !($where_result instanceof FQLCantSee))
          $result_ids []= $id;
    }

    $result = array();
    $row_name = $this->from_table->get_name(); // e.g. "user"

    // fill in the result array with the requested data
    foreach ($result_ids as $id) {
      foreach ($this->select as $str => $expression) { // e.g. "books" or "pic"
        $name = $expression->get_name();
        $col = $expression->evaluate($id); // returns the value
        if ($col instanceof FQLCantSee)
          $col = null;

        $row->value[] = new xml_element($name, $col);
      }

      $result[] = $row;
    }
    return $result;
}
```

FQL has some other subtleties, but this general flow illustrates the union of existing internal data access and privacy implementations with a whole new query model. This allows the developer to process his request more quickly and access data in a more granular way than the APIs, while still retaining the familiarity of SQL syntax.

As many of our APIs internally wrap corresponding FQL methods, our overall architecture has evolved to the state shown in Figure 6-4.