Let's start with this user table as an example and build the FQL system to support queries on it. Underneath all the layers of data abstraction through the Platform (the internal calls, the users_getInfo external API call, and the new user table of FQL), imagine Facebook had a table named 'user' in its own database (Example 6-16).

EXAMPLE 6-16. Example Facebook data table

```
> describe user;
+--------------+--------------+-----+
| Field        | Type         | Key |
+--------------+--------------+-----+
| uid          | bigint(20)   | PRI |
| name         | varchar(255) |     |
| pic          | varchar(255) |     |
| books        | varchar(255) |     |
| loc_city     | varchar(255) |     |
| loc_state    | varchar(255) |     |
| loc_country  | varchar(255) |     |
| loc_zip      | int(5)       |     |
+--------------+--------------+-----+
```

Within the Facebook stack, suppose our method for accessing this table is:

```
function user_get_info($uid)
```

which returns an object in the language of our choice (PHP), usually used before applying privacy logic and rendering on *http://facebook.com*. Our web service implementation did much the same, transforming the GET/POST content of a web request to such a call, obtaining a similar stack object, applying privacy, and then using Thrift to render this as an XML response (Figure 6-2).

We can wrap user_get_info within FQL to programmatically apply this model, with tables, fields, internal functions, and privacy all fitting together in a logical, repeatable form.

Following are some key objects created in the FQL call in Example 6-15 and the methods that describe how they relate. Discussion of the entire string parsing, grammar implementation, alternative indexing, intersecting queries, and implementing the many different combining expressions (comparisons, "in" statements, conjunction, and disjunction) are beyond the scope of this chapter. Instead, we'll just focus on the data-facing pieces: the high-level specification of the data's corresponding field and table objects within FQL, and transforming the input statement to queries to each field's can_see and evaluate functions (Example 6-17).

EXAMPLE 6-17. Example FQL fields and tables

```
class FQLField {
  // e.g. table="user", name="current_location"
  public function __construct($user, $app_id, $table, $name) { ... }

  // mapping: "index" id -> {0,1} (visible or invisible)
  public function can_see($id) { ... }
```