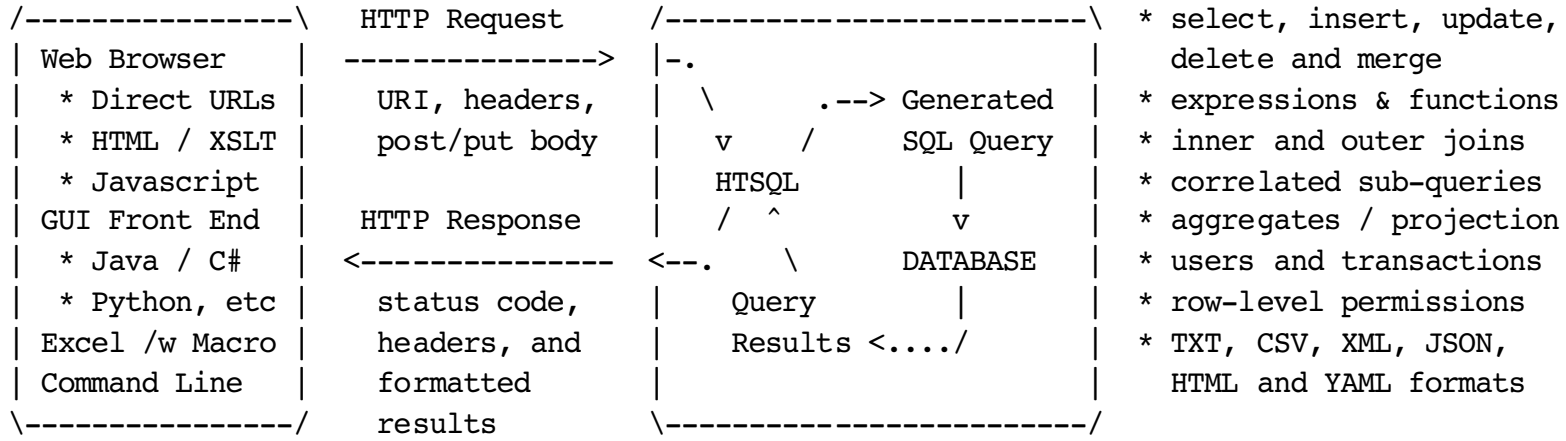
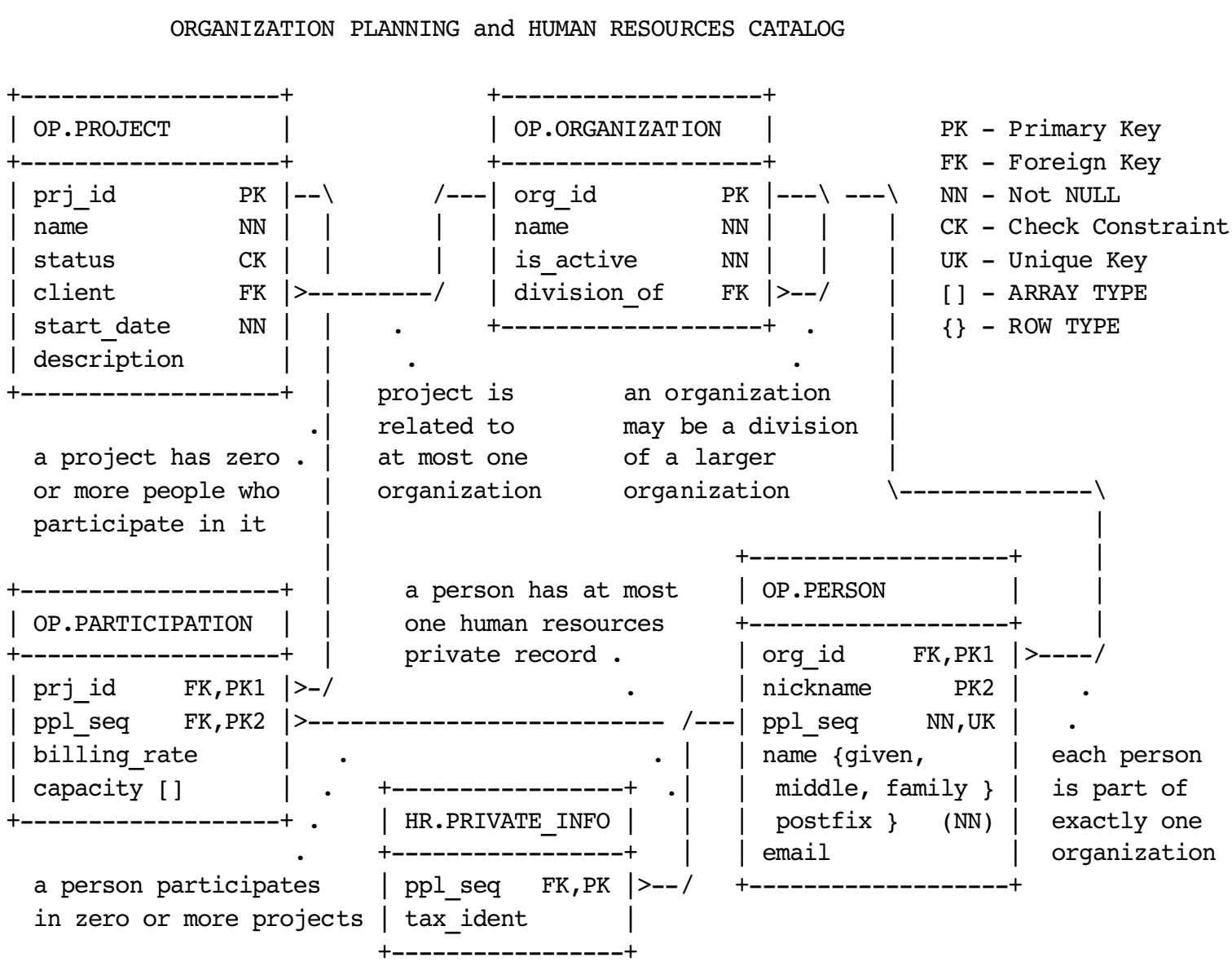
Hyper-Text Structured Query Language

HTSQL is a middleware component that translates a HTTP request into a SQL query, performs the query against a relational database, and returns the result as XML, HTML, CSV, JSON, or YAML.

HTSQL formalizes a URI-to-SQL translation, covering common database query constructs with a succinct, easy-to-learn syntax. HTSQL decouples the application from the data-store, putting the database itself "on the web". HTSQL is Open Source Technology.



HTSQL - Example / Regression Schema



HTSQL - Selection and Filters

GET /op:organization

This request selects all rows from the organization table in the op schema. By default, rows are ordered by primary key.

```
SELECT o.*
FROM op.organization AS o
ORDER BY o.org_id
```

```
200 OK
Content-Type: text/plain; charset=UTF-8
```

organization			
org_id	name	is_active	division_of
lakeside	Lake Side Partners, LLC		
lsapts	Lake Shore Apartments	True	lakeside
lstower	Lake Side Towers	True	lakeside
meyers	Meyers Group	True	
smith	Rudgen, Taupe, and Smith	False	

HTSQL uses percent-encoding for non-printable characters, UTF-8 sequences, or RFC 2396 unwise or reserved characters. Like SQL, catalog entries can be double-quoted for case-sensitive matching.

GET /op:organization.xml ?name~'meyers'

This query returns organizations where the name matches a case-insensitive regular expression. Unlike HTML form submission, string literals are always single-quoted.

```
SELECT o.*
FROM op.organization AS o
WHERE LOWER(name) LIKE '%meyers%'
ORDER BY o.org_id
```

```
200 OK
Content-Type: text/xml; charset=UTF-8
```

```
<organization htsql:schema="op">
  <_ org_id="meyers" name="Meyers Group"
     is_active="true" division_of=""
     htsql:is_null="division_of"/>
</organization>
```

In this example, the XML output format was requested. HTSQL attributes are used cases to indicate NULLs, to name a schema or when the table/column identifier is not a valid XML name.

HTSQL - Join Specifiers

```
GET /op:project
      ?client.name~'meyers'
```

An automatic join is constructed when a single-column foreign-key is used with the "dot" operator. In the `op` schema, the `client` column of the `project` table is a foreign key reference to `organization`.

```
SELECT p.*
FROM op.project AS p
JOIN organization AS o
 ON (p.client = o.org_id)
WHERE LOWER(o.name)
LIKE '%meyers%'
ORDER BY p.prj_id
```

When it is not ambiguous, a table name can likewise be used to indicate the foreign-key join. In the above example, the column specifier `client` could be replaced with `organization`.

```
GET /op:project
      ?participation
```

Foreign-keys are also used in reverse to create "plural" specifiers. The example above returns all projects that have at least one associated participation record.

```
SELECT p.*
FROM op.project AS p
WHERE EXISTS
(SELECT 'X'
 FROM op.participation AS x
 WHERE x.proj_id = p.proj_id)
ORDER BY p.prj_id
```

In a predicate expression, a plural join specifier is treated as an implicit test for existence. If a column is referenced by a plural specifier in this manner, it is implicitly converted to a boolean value.

HTSQL - Functions and Expressions

```
GET /op:organization
    ?org_id.lower()[:2]='me'
     &!division_of
```

Full predicate algebra is supported, as well as standard SQL functions and operators. The "slice" syntax sugar is included for substring operations.

```
SELECT o.*
FROM op.organization AS o
WHERE 'me' = SUBSTRING(
 LOWER(o.org_id) FROM 1 FOR 2)
AND (o.division_of IS NULL
  OR o.division_of = '')
ORDER BY o.org_id
```

In HTSQL, non-boolean values found in a predicate are implicitly cast as boolean: zero, empty string, array of zero length, and `NULL` are `FALSE`; all other values are `TRUE`. All other type casting in HTSQL is explicit. Functions are strictly typed.

```
GET /op:organization
?max(project.start_date)
.year<2004
```

Aggregate functions are supported on plural specifiers. Furthermore, fields of user defined types and date components can be accessed with the dot operator.

```
SELECT o.*
FROM op.organization AS o
WHERE EXTRACT(YEAR FROM
 (SELECT MAX(p.start_date)
  FROM op.project AS p
  WHERE p.client = o.org_id))
 < 2004
ORDER BY o.org_id
```

To enhance readability, a method syntax is provided for polymorphic functions.

NOTE: In the current implementation, some of the generated SQL isn't this pretty. However, it is equivalent.

Selector, Aliases and Projections

```
GET /op:person
{name+,organization.*,
 $last_four:=private_info
 .tax_ident[-4:]}
?$last_four.contains('33')
```

Curly braces are used to specify which values are to be returned. Custom sort order is provided by a trailing plus or minus. Column aliases , denoted by the dollar sign, are set with the `:=` operator.

```
SELECT p.name AS "person.name",
 o.org_id AS "organization.org_id",
 o.name AS "organization.name",
 o.is_active AS "organization.is_active",
 o.division_of AS "organization.division_of",
 SUBSTRING(r.tax_ident FROM
 (LENGTH(r.tax_ident)-4+1)) AS "last_four"
FROM op.person AS p
LEFT OUTER JOIN op.organization AS o
 ON (p.org_id = o.org_id)
LEFT OUTER JOIN hr.private_info AS r
 ON (r.ppl_seq = p.ppl_seq)
WHERE POSITION('33' IN SUBSTRING(r.tax_ident
 FROM (LENGTH(r.tax_ident)-4+1)))>0
ORDER BY p.name ASC, p.org_id
```

```
GET /op:project
{status|max(start_date)}
```

Aggregate functions work in a two step process. First, a 1-1 correspondence is setup with a table (or, in the case above, a virtual result set). Then, the aggregation happens relative to that basis. If the basis does not correspond exactly to a given table's rows, then the projection indicator (a vertical bar) is needed. Equivalently, `op:project{status|}` returns distinct status codes in the project table.

```
SELECT p.status AS "status",
 max(p.start_date) AS
 "max(start_date)"
FROM op.project AS p
GROUP BY p.status
ORDER BY p.status
```

The exact implementation of projection is a bit complicated once joined tables and multiple aggregates are considered.

HTSQL - Locators

GET /op:participation {id(), person.id()}

A location, constructed via the `id()` function, uniquely identifies a row in a table. It is based off primary key columns, recursively including the location of parent tables when a foreign key is used.

```
SELECT (a.prj_id || '.(' || o.org_id
 || '.' || n.nickname || ')') AS "id()",
 (o.org_id || '.' || n.nickname)
   AS "person.id()"
FROM op.participation AS a
JOIN op.project AS p ON (a.prj_id = p.prj_id)
JOIN op.person AS n
 ON (a._ppl_seq = n._ppl_seq)
JOIN op.organization AS o
 ON (n.org_id = n.org_id)
ORDER BY n.prj_id, o.org_id, n.nickname
```

```
200 OK
Content-Type: text/plain; charset=UTF-8

         id()           |  person.id()
------------------------+---------------
 smbl.(meyers.maggy)    | meyers.maggy
 smbl.(lakeside.maggy)  | lakeside.maggy
 la-102.(meyers.tom)    | meyers.tom
 la-802.(meyers.maggy)  | meyers.maggy
...
```

GET /op:person[meyers.tom]

A locator is a comma-separated list of locations which return a specific row.

```
SELECT p.*
FROM op.person AS p
JOIN op.organization AS o
 ON (p.org_id = o.org_id)
WHERE htsql_normalize(o.org_id)
 = htsql_normalize('meyers')
AND htsql_normalize(p.nickname)
 = htsql_normalize('tom')
ORDER BY o.org_id, n.prj_id
```

```
where htsql_normalize(x) :=
 TRANSLATE(TRIM(BOTH ' ' FROM LOWER(CAST($1 AS TEXT))),
          ' ~`!@#$%^&*()-_={}[]|\:;"<>,.?/''',
          '--------------------------------')
```

In other words, comparison by locator is case insensitive and ignores special characters. If this is not unique, then single-quoting is required, e.g. `'MeYeRs'`; further, for that table, **id**() will quote.

HTSQL - Request Segments and Commands

GET /op:organization[meyers]
/op:person{nickname}.xml

To support drill-down behavior and nested report structures, multiple segments are supported if there is a unique join from one to the other.

```
SELECT o.*, p.nickname
FROM op.organization AS o
JOIN op.person AS p
ON (p.org_id = o.org_id)
WHERE htsql_normalize(o.org_id) =
 htsql_normalize('meyers')
ORDER BY o.org_id, o.org_id, p.nickname
```

```
200 OK
Content-Type: text/xml; charset=UTF-8

<organization htsql:schema="op">
  <_ org_id="meyers" name="Meyers Group"
     is_active="true" division_of=""
     htsql:is_null="division_of">
   <person htsql:schema="op">
     <_ nickname="hill" />
     <_ nickname="jack" />
     <_ nickname="jim" />
   </person>
 </_>
</organization>
```

GET /op:organization
/select(limit=50,offset=50)
.json(indent=1)

By default we have been using a default command for our examples — SELECT, and either the default file format, the plain text debug output, or an XML format. Both commands and formatters can be provided, taking arguments.

```
SELECT o.*
FROM op.organization AS o
ORDER BY o.org_id
LIMIT 50 OFFSET 100
```

```
200 OK
Content-Type: text/json; charset=UTF-8

[
 {
  org_id: "meyers",
  name: "Meyers Group"
  is_active: true,
  division_of: null
 } ,
...
]
```

HTSQL - Insert/Update/Delete

```
/op:organization[lakeside]/
 /op:person/insert()
 ?nickname:='o-brien'
 &name{family,given}:=
 {'O''Brien','Mark'}
 &private_info.tax_ident:=
 '283-33-9999'
```

This command looks up the correct foreign key to link `organization` and `person`, creates row in person table, and then creates a row in the "facet" table (1-1 correspondence), `private_info`.

```
INSERT INTO "op"."individual"
(org_id, nickname, name)
SELECT o.org_id, 'o-brien',
 ROW('O''Brien',NULL,'Mark',NULL)
FROM op.organization AS o
WHERE htsql_normalize(o.org_id) =
 htsql_normalize('lakeside')
RETURNING ppl_seq;

INSERT INTO "op"."private_info"
(ppl_seq, tax_ident)
SELECT '283-33-9999', <returned-id>;
```

```
/op:person[lakeside.o-brien]
 /update()?organization:=
 @organization[ls-tower]
```

This command changes Mark's organization, by looking up (via location) the proper foreign key for ls-tower.

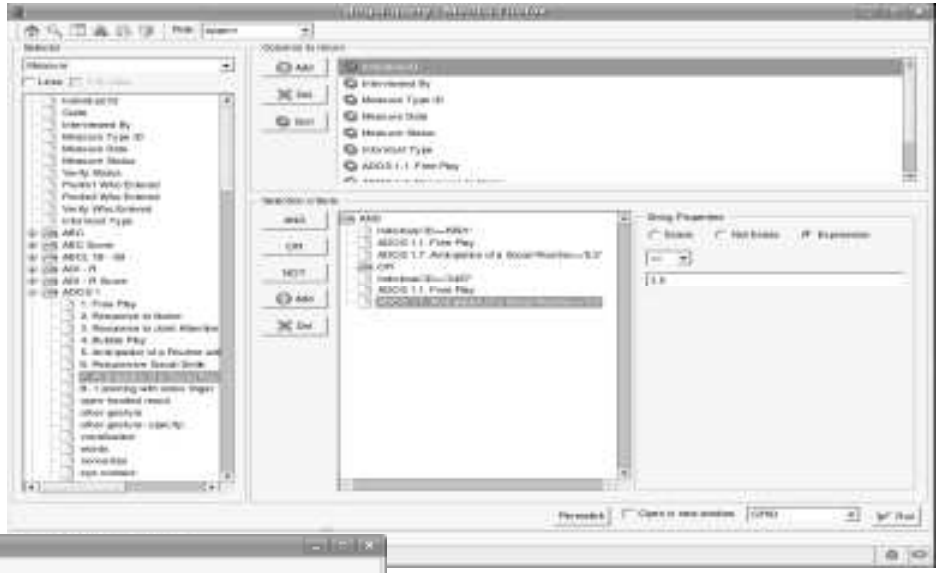
```
UPDATE op.person
   SET org_id =
     (SELECT org_id
        FROM op.organization
        WHERE htsql_norm(org_id)
            = htsql_norm('ls-tower'))
WHERE (org_id, nickname) IN
  (SELECT p.org_id, p.nickname
     FROM person p
     JOIN op.organization o
       ON (p.org_id = o.org_id)
    WHERE htsql_norm(o.org_id)
        = htsql_norm('lakeside')
    AND htsql_norm(p.nickname)
        = htsql_norm('o-brien'))
```

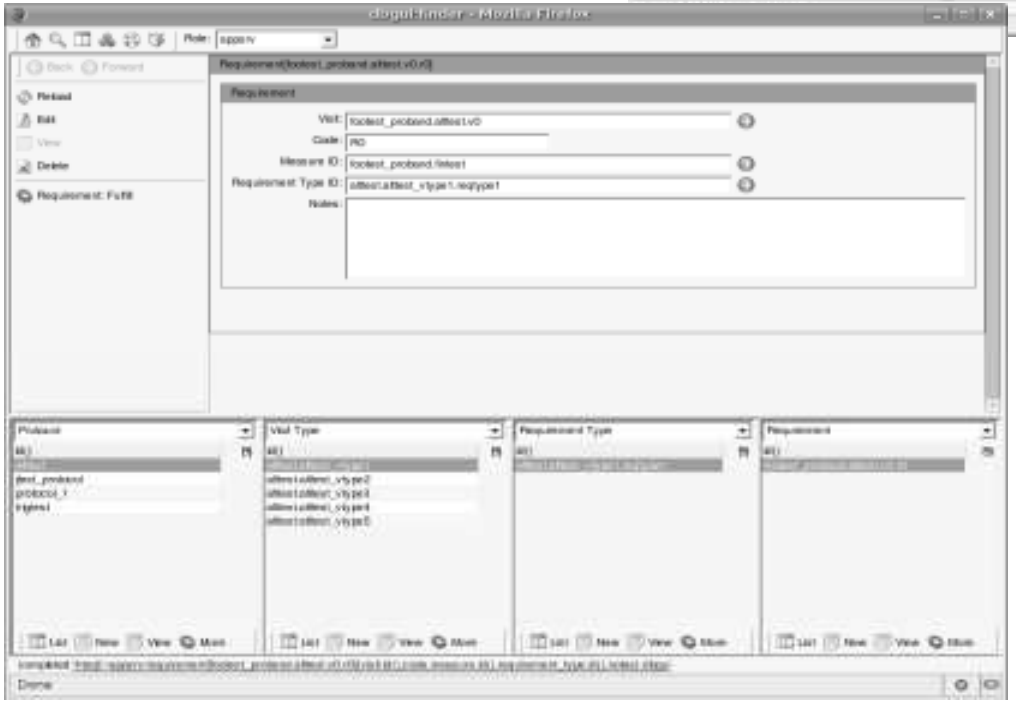
The extra complication (no-ops in this case) is needed to handle situations where the locator does not correspond to the primary key columns of the table.

HTSQL - It makes GUIs Easier

HTSQL enables reliable, scalable, and rapid development of Web Clients using Javascript.

By having a solid, heavily tested URL-to-SQL translation language, we've made huge strides meeting our other goals: a "DBGUI", which will be made open source this September.





HTSQL is an open source product of Prometheus Research, LLC. HTSQL was inspired by building web-apps with James Clark's XSLT. It was specified and prototyped by Clark Evans, and then implemented by Kirill Simonov.

This work would not have been possible without the generous support from the Simons Foundation.