




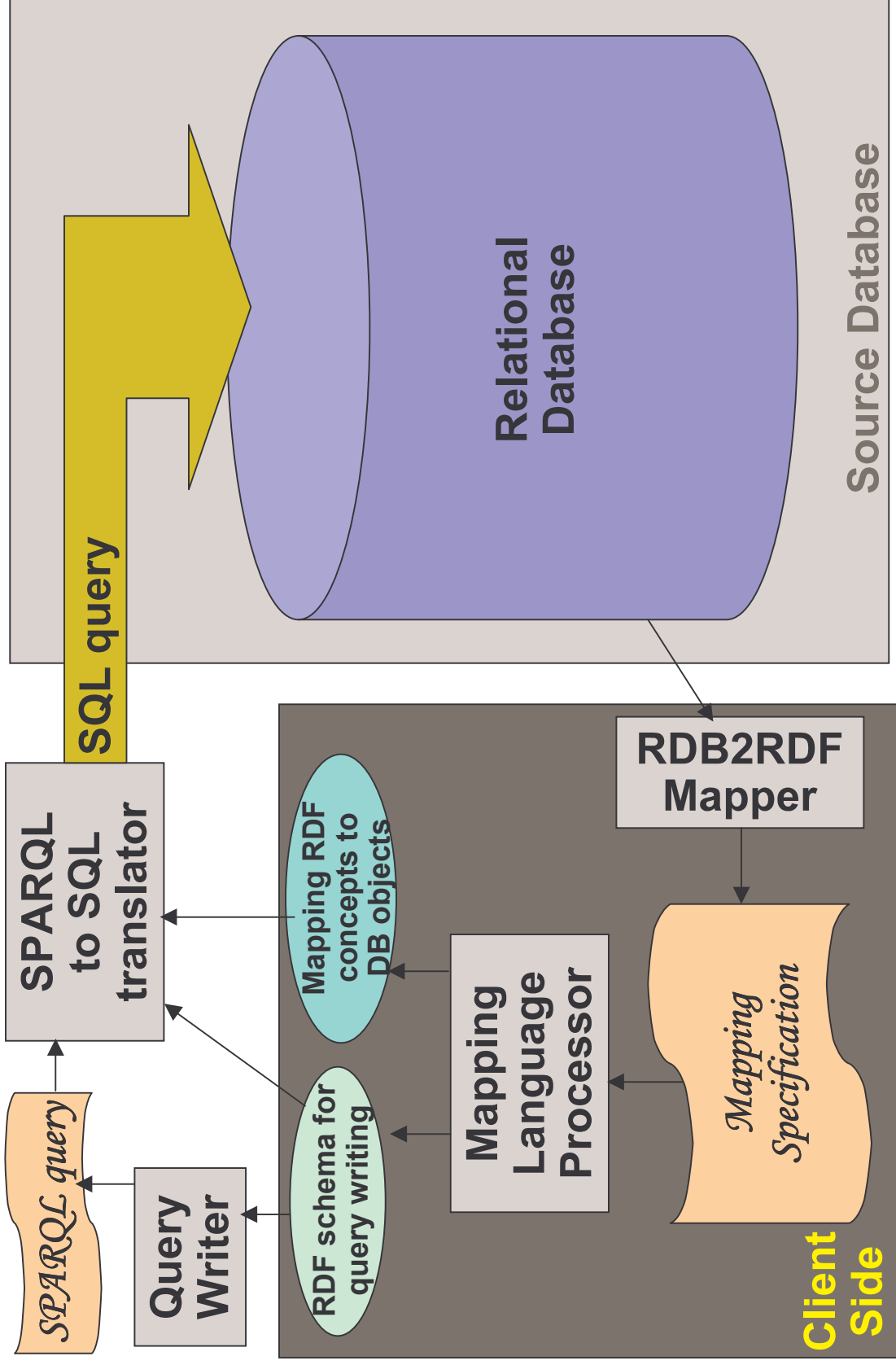
A SQL-based Approach for Mapping Relational Data to RDF

Souripriya Das and Seema Sundara
Database Semantic Technologies Group, Oracle

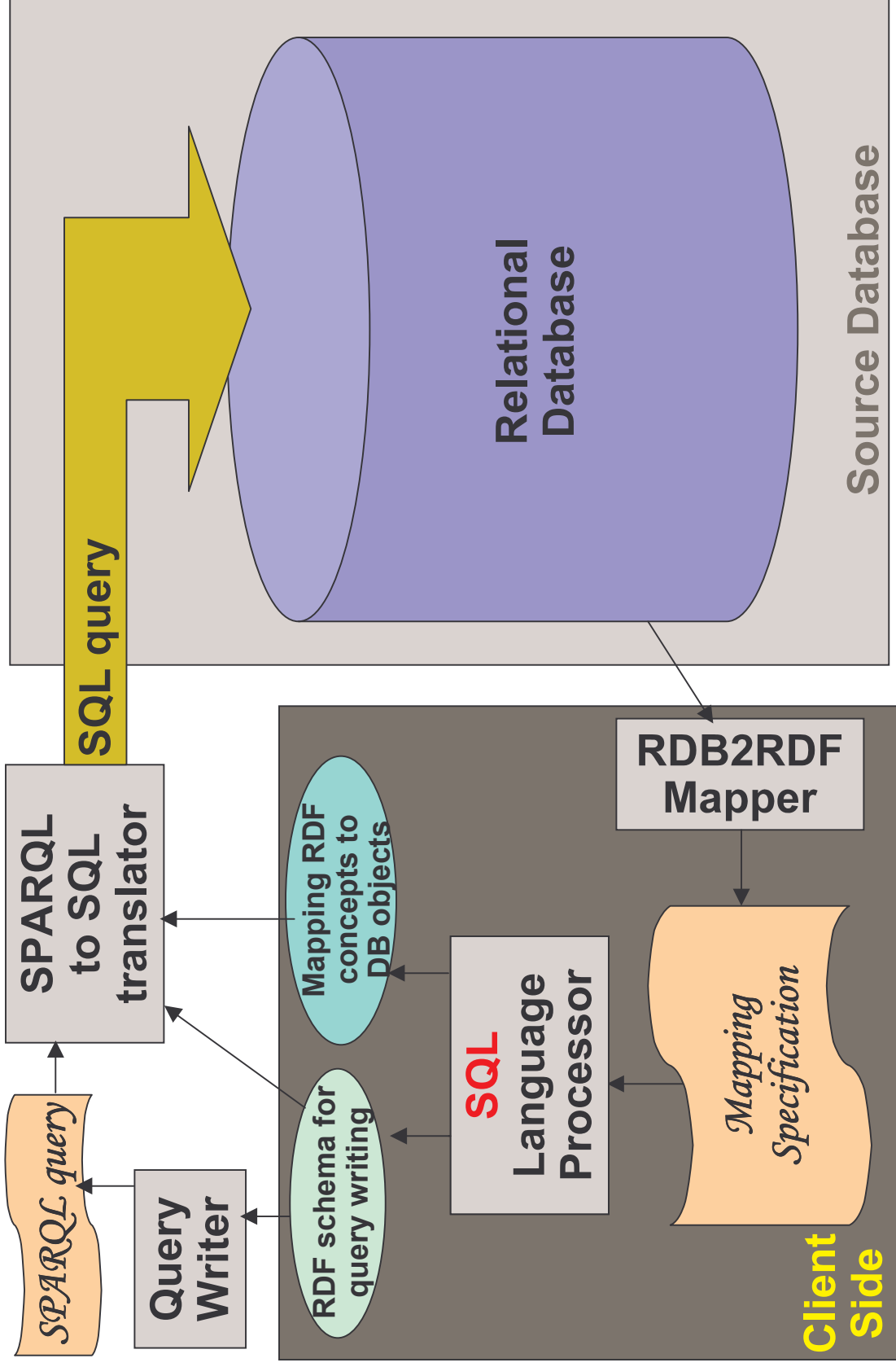


THE FOLLOWING IS INTENDED TO OUTLINE OUR GENERAL PRODUCT DIRECTION. IT IS INTENDED FOR INFORMATION PURPOSES ONLY, AND MAY NOT BE INCORPORATED INTO ANY CONTRACT. IT IS NOT A COMMITMENT TO DELIVER ANY MATERIAL, CODE, OR FUNCTIONALITY, AND SHOULD NOT BE RELIED UPON IN MAKING PURCHASING DECISION. THE DEVELOPMENT, RELEASE, AND TIMING OF ANY FEATURES OR FUNCTIONALITY DESCRIBED FOR ORACLE'S PRODUCTS REMAINS AT THE SOLE DISCRETION OF ORACLE.

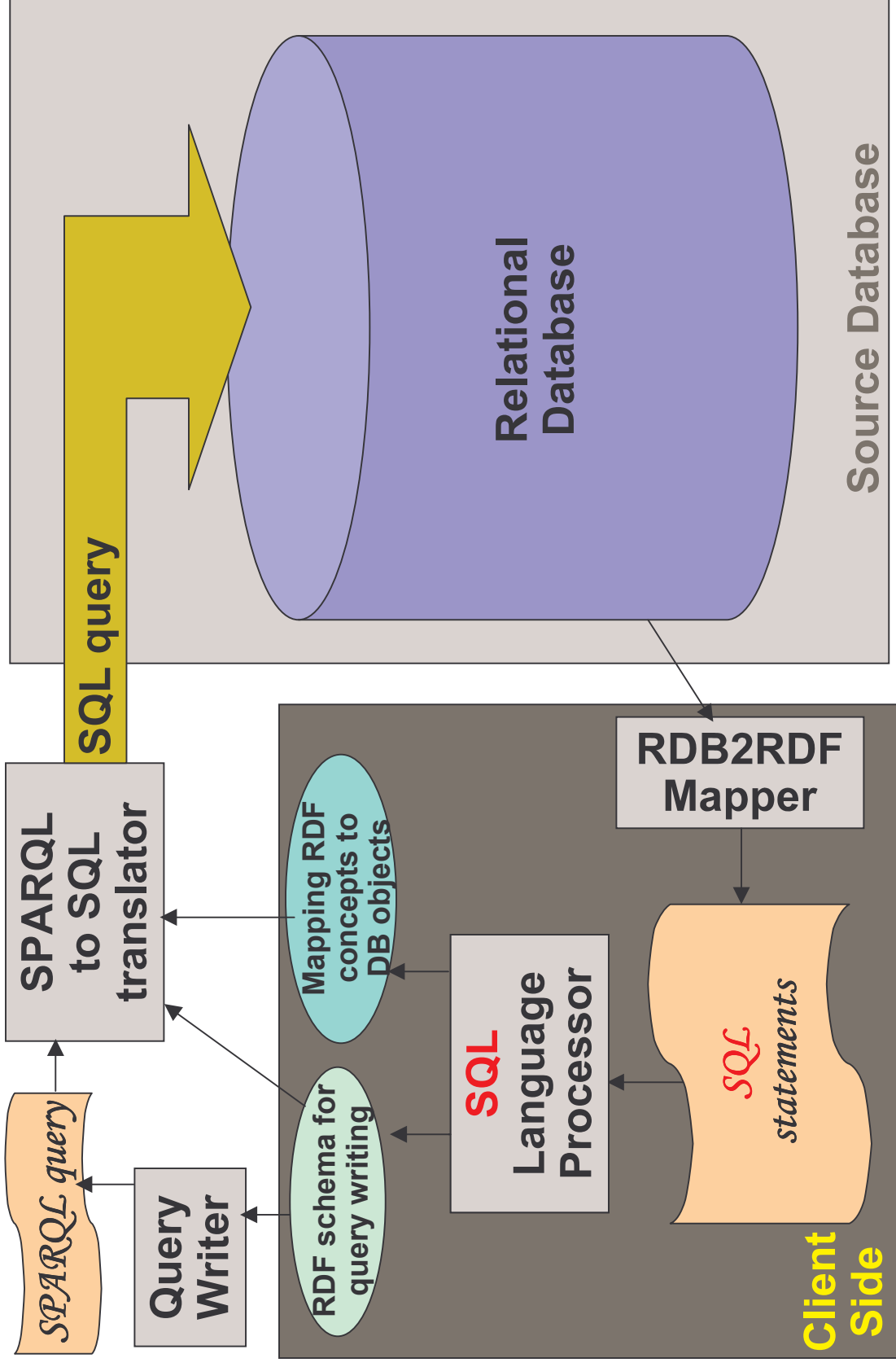
Architecture Diagram for RDB2RDF processing

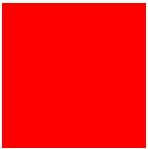


Architecture Diagram for RDB2RDF processing

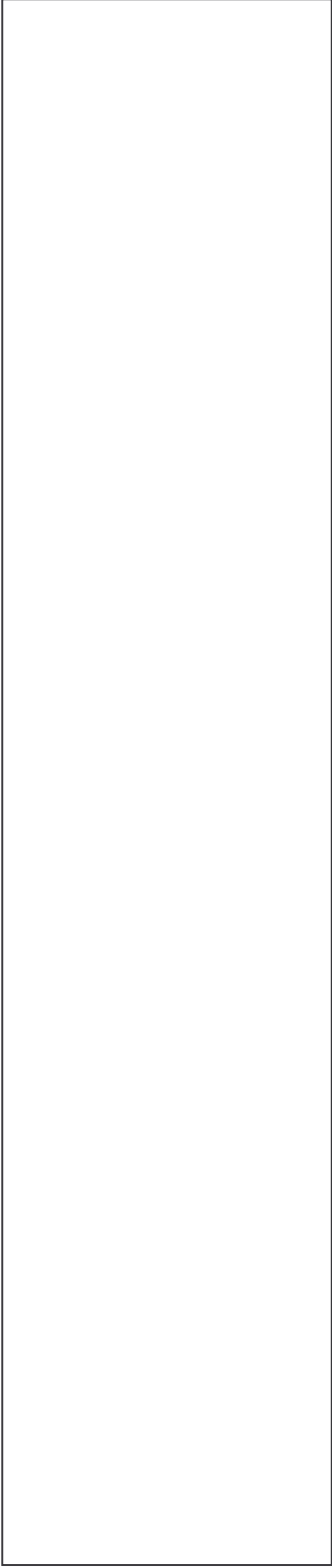


Architecture Diagram for RDB2RDF processing





Overview





Overview

- **Basis** for defining an RDFS/OWL class and its properties based on relational data stored in a source database:
 - A **SQL query** specifying **arbitrary transformation** of source relational data
 - Related foreign and unique (or primary) key constraint definitions.



Overview

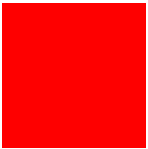
- **Basis** for defining an RDFS/OWL class and its properties based on relational data stored in a source database:
 - A **SQL query** specifying **arbitrary transformation** of source relational data
 - Related foreign and unique (or primary) key constraint definitions.
- Client side capabilities may vary
 - **Strong-DB**: An RDBMS with support for Views and Constraints on Views. (Example: Oracle database server)
 - **No-DB**: Only connects to database using JDBC or ODBC.

Overview

- **Basis** for defining an RDFS/OWL class and its properties based on relational data stored in a source database:
 - A **SQL query** specifying **arbitrary transformation** of source relational data
 - Related foreign and unique (or primary) key constraint definitions.

- Client side capabilities may vary
 - **Strong-DB**: An RDBMS with support for Views and Constraints on Views. (Example: Oracle database server)
 - **No-DB**: Only connects to database using JDBC or ODBC.

- Proposed mapping language
(Leverage **SQL on source database** and if available, **SQL on client side**)
 - **Strong-DB** → No new language. Employ some *conventions* with SQL (e.g., when naming Views, View cols, and constraints).
 - **No-DB** → Simple language to specify mapping between RDF classes, properties and SQL queries, query projections, constraints.

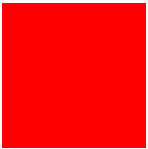


DB View → RDF Class and Properties

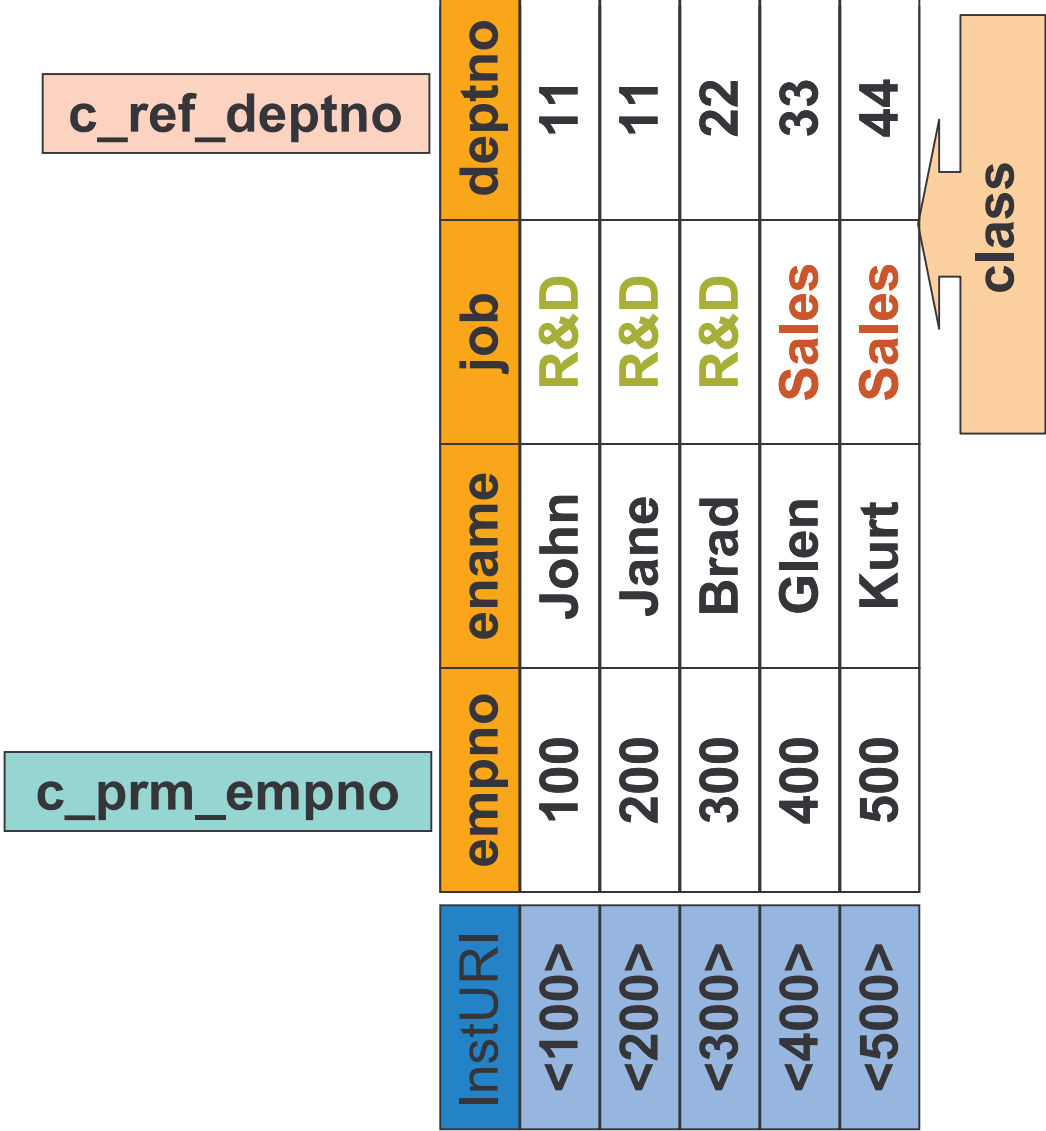
c_prm_empno

c_ref_deptno

empno	ename	job	deptno
100	John	R&D	11
200	Jane	R&D	11
300	Brad	R&D	22
400	Glen	Sales	33
500	Kurt	Sales	44



DB View → RDF Class and Properties



DB View → RDF Class and Properties

InstURI	empno	ename	job	deptno	rdf:type	etype	rdf:type
<100>	100	John	R&D	11	○	part	<part>
<200>	200	Jane	R&D	11	○	full	<full>
<300>	300	Brad	R&D	22	○	intrn	<intrn>
<400>	400	Glen	Sales	33	⊘	full	<full>
<500>	500	Kurt	Sales	44	⊘	full	<full>

c_prm_empno c_ref_deptno job- based classes etype- based classes

class

Example: DEPT view defines Class and Properties

```
CREATE VIEW "<xyz.com/dept>" AS
```

```
SELECT
```

```
'<xyz.com/dept/' || deptno || '>'
```

```
, deptno
```

```
, dname
```

```
, loc
```

```
FROM dept
```

Example: DEPT view defines Class and Properties

```
CREATE VIEW "<xyz.com/dept>" AS
SELECT
  '<xyz.com/dept/' || deptno || '>' InstURI
  , deptno
  , dname
  , loc
FROM dept
```

Example: DEPT view defines Class and Properties

```
CREATE VIEW "<xyz.com/dept>" AS
SELECT
  '<xyz.com/dept/' || deptno || '>' InstURI
  , deptno
  , dname
  , loc
FROM dept
```

```
ALTER VIEW "<xyz.com/dept>"
add constraint "<xyz.com/dept/c_unq_deptno>"
unique ("<xyz.com/dept/deptno>")
disable novalidate;
```



Example: EMP view defines Class and Properties

```
CREATE VIEW "<xyz.com/emp>" AS
SELECT
    '<xyz.com/emp/' || empno || '>'
    , empno
    , ename
    , '<xyz.com/emp/job/' || job || '>'
    , job
    , deptno
FROM emp
```

Example: EMP view defines Class and Properties

CREATE VIEW	“<xyz.com/emp>” AS	InstURI
SELECT	‘<xyz.com/emp/’ empno ‘>’	“<xyz.com/emp/empno>”
	, empno	“<xyz.com/emp/Name>”
	, ename	“rdf:type”
	, ‘<xyz.com/emp/job/’ job ‘>’	“<xyz.com/emp/job>”
	, job	“<xyz.com/emp/deptNum>”
	, deptno	
FROM	emp	

Example: EMP view defines Class and Properties

```
CREATE VIEW "<xyz.com/emp>" AS
SELECT
  '<xyz.com/emp/' || empno || '>'
  , empno
  , ename
  , '<xyz.com/emp/job/' || job || '>'
  , job
  , deptno
FROM emp
ALTER VIEW "<xyz.com/emp>"
add constraint "<xyz.com/emp/c_prm_empno>"
primary key ("<xyz.com/emp/empno>") disable novalidate;
```

'<xyz.com/emp/' empno '>'	InstURI
, empno	"<xyz.com/emp/empno>"
, ename	"<xyz.com/emp/Name>"
, '<xyz.com/emp/job/' job '>'	"rdf:type"
, job	"<xyz.com/emp/job>"
, deptno	"<xyz.com/emp/deptNum>"

primary Key

Example: EMP view defines Class and Properties

```

CREATE VIEW "<xyz.com/emp>" AS
SELECT
    '<xyz.com/emp/' || empno || '>' InstURI
    , empno
    , ename
    , '<xyz.com/emp/job/' || job || '>'
    , job
    , deptno
FROM emp

ALTER VIEW "<xyz.com/emp>"
add constraint "<xyz.com/emp/c_prm_empno>"
primary key ("<xyz.com/emp/empno>") disable novalidate;

ALTER VIEW "<xyz.com/emp>"
add constraint "<xyz.com/emp/c_ref_deptno>"
foreign key ("<xyz.com/emp/deptNum>")
references "<xyz.com/dept>" ("<xyz.com/dept/deptno>")
disable novalidate;

```

primary Key

foreign Key

Example: EMP view with multiple rdf:type columns

```
CREATE VIEW "<xyz.com/emp>" AS  
SELECT
```

'<xyz.com/emp/' empno '>'	InstURI
, empno	"<xyz.com/emp/empno>"
, ename	"<xyz.com/emp/Name>"
, '<xyz.com/emp/job/' job '>'	"<xyz.com/emp/job/rdf:type>"
, job	"<xyz.com/emp/job>"
, deptno	"<xyz.com/emp/deptNum>"
, '<xyz.com/emp/etype/' etype '>'	"<xyz.com/emp/etype/rdf:type>"
, etype	"<xyz.com/emp/etype>"

```
FROM  
emp
```

Example: EMP view with multiple rdf:type columns

**CREATE VIEW “<xyz.com/emp>” AS
SELECT**

	InstURI
, empno	“<xyz.com/emp/empno>”
, ename	“<xyz.com/emp/Name>”
, ‘<xyz.com/emp/job/’ job ‘>’	“<xyz.com/emp/job/rdf:type>”
, job	“<xyz.com/emp/job>”
, deptno	“<xyz.com/emp/deptNum>”
, ‘<xyz.com/emp/etype/’ etype ‘>’	“<xyz.com/emp/etype/rdf:type>”
, etype	“<xyz.com/emp/etype>”

FROM
emp

**ALTER VIEW “<xyz.com/emp>”
add constraint “<xyz.com/emp/c_prm_empno>”
primary key (“<xyz.com/emp/empno>”) disable novalidate;**

**primary
Key**

Example: EMP view with multiple rdf:type columns

CREATE VIEW "**<xyz.com/emp>**" AS
SELECT

	InstURI
'<xyz.com/emp/' empno '>'	
, empno	"<xyz.com/emp/empno>"
, ename	"<xyz.com/emp/Name>"
, '<xyz.com/emp/job/' job '>'	"<xyz.com/emp/job/rdf:type>"
, job	"<xyz.com/emp/job>"
, deptno	"<xyz.com/emp/deptNum>"
, '<xyz.com/emp/etype/' etype '>'	"<xyz.com/emp/etype/rdf:type>"
, etype	"<xyz.com/emp/etype>"

FROM
emp

ALTER VIEW "<xyz.com/emp>"
add constraint "<xyz.com/emp/c_prm_empno>"
primary key ("<xyz.com/emp/empno>") disable novalidate;

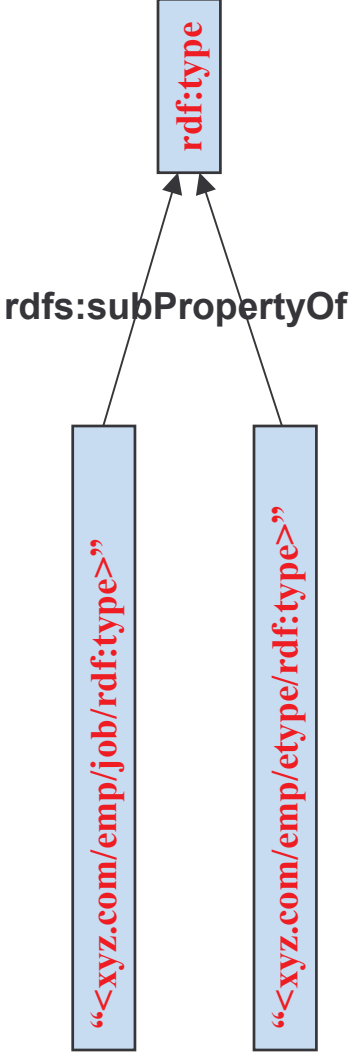
primary
Key

ALTER VIEW "<xyz.com/emp>"
add constraint "**<xyz.com/emp/c_ref_deptno>**"
foreign key ("<xyz.com/emp/deptNum>")
references "<xyz.com/dept>" ("<xyz.com/dept/deptno>")
disable novalidate;

foreign
Key



Subproperties of `rdf:type`





SPARQL Query: Involving generic rdf:type

```
SELECT ?x ?ename
WHERE {
  ?x <xyz.com/emp/Name> ?ename .
  ?x rdf:type <xyz.com/emp/job/SALES>
}
```

SPARQL Query: Involving generic rdf:type

```
SELECT ?x ?ename
WHERE {
  ?x <xyz.com/emp/Name> ?ename .
  ?x rdf:type <xyz.com/emp/job/SALES>
}
```

```
SELECT
  e.instURI x
, e.“<xyz.com/emp/Name>” ename
FROM
  “<xyz.com/emp>” e
WHERE
  e.“<xyz.com/emp/job/rdf:type>” = “<xyz.com/emp/job/SALES>”
or
  e.“<xyz.com/emp/etype/rdf:type>” = “<xyz.com/emp/job/SALES>”
```

Prefix-based DB identifiers (stay within length-limits)

RDF_PREFIX_MAP\$

View	Prefix	Expansion	
"xyz:emp"	xyz:	<xyz.com/>	Prefix for class name
"xyz:emp"	emp:	<xyz.com/emp/>	Prefix for cols/cons
"xyz:dept"	xyz:	<xyz.com/>	Prefix for class name
"xyz:dept"	dept:	<xyz.com/dept/>	Prefix for cols/cons

- **ADD Unique constraint (View, Prefix)**
- **ADD Unique constraint (View, Expansion)**



No DB: <ViewName, SQLdefString> info

ViewName	SQLdefString
"xyz:dept"	<pre>SELECT ,deptno ,dname ,loc FROM dept</pre>
"xyz:emp"	<pre>SELECT ,empno ,ename , '<xyz.com/emp/job/' job '>' ,job ,deptno FROM emp</pre>

No DB: <ViewName, SQLdefString> info

ViewName	SQLdefString
"xyz:dept"	<pre>SELECT '<xyz.com/dept/' deptno '>' , deptno , dname , loc FROM dept</pre>
"xyz:emp"	<pre>SELECT '<xyz.com/emp/' empno '>' , empno , ename , '<xyz.com/emp/job/' job '>' , job , deptno FROM emp</pre>

No DB: <ViewName, SQLdefString> info

ViewName	SQLdefString
"xyz:dept"	<pre>SELECT '<xyz.com/dept/' deptno '>' , deptno , dname , loc FROM dept InstURI "dept:deptno" "dept:Name" "dept:Location"</pre>
"xyz:emp"	<pre>SELECT '<xyz.com/emp/' empno '>' , empno , ename , '<xyz.com/emp/job/' job '>' , job , deptno FROM emp InstURI "emp:empno" "emp:Name" "rdf:type" "emp:job" "emp:deptNum"</pre>



No DB: View Constraint info

ConsName	ConsType	ViewName	RefConsName
“dept:c_unq_deptno”	Unique	“xyz:dept”	
“emp:c_prim_empno”	Primary	“xyz:emp”	
“emp:c_ref_deptno”	Reference	“xyz:emp”	“dept:c_unq_deptno”

ConsName	ViewName	ColName	ColPosInKey
“dept:c_unq_deptno”	“xyz:dept”	“dept:deptno”	1
“emp:c_prim_empno”	“xyz:emp”	“emp:empno”	1
“emp:c_ref_deptno”	“xyz:emp”	“emp:deptNum”	1



No DB: <ViewName, ColPos, Property> info

ViewName	ColPos	Property
"xyz:dept"	1	InstURI
"xyz:dept"	2	dept:deptno
"xyz:dept"	3	dept:Name
"xyz:dept"	4	dept:Location
"xyz:emp"	1	InstURI
"xyz:emp"	2	emp:empno
"xyz:emp"	3	emp:Name
"xyz:emp"	4	rdf:type
"xyz:emp"	5	emp:job
"xyz:emp"	6	emp:deptNum
"xyz:emp"	0	emp:c_ref_deptno



No DB: A simple syntax for Classes

```
Class (ViewName, SQLdefString)  
Class ("xyz:dept", <sql-def-for-dept>)  
Class ("xyz:emp", <sql-def-for-emp>)
```

No DB: A simple syntax for Properties¹

```
Property (PropertyName, ViewName, ColPos)
```

```
Property ("dept:InstURI", "xyz:dept", 1)
```

```
Property ("dept:deptno", "xyz:dept", 2)
```

```
Property ("dept:Name", "xyz:dept", 3)
```

```
Property ("dept:location", "xyz:dept", 4)
```

```
Property ("emp:InstURI", "xyz:emp", 1)
```

```
Property ("emp:empno", "xyz:emp", 2)
```

```
Property ("emp:Name", "xyz:emp", 3)
```

```
Property ("emp:rdf:type", "xyz:emp", 4)
```

```
Property ("emp:job", "xyz:emp", 5)
```

```
Property ("emp:deptNum", "xyz:emp", 6)
```

```
Property ("emp:c_ref_deptno", "xyz:emp", 0)
```

¹ Inverse function specification for properties not shown, but can be added easily.

•Range information can be derived from type info obtained from SQL engine at source database.

No DB: A simple syntax for Constraints

```
Constraint (ConsName, ConsType, ViewName, RefConsName)
```

```
Constraint ("dept:c_unq_deptno", Unique, "xyz:dept", NULL)
```

```
Constraint ("emp:c_prm_empno", Primary, "xyz:emp", NULL)
```

```
Constraint ("emp:c_ref_deptno", Reference, "xyz:emp", "dept:c_unq_deptno")
```

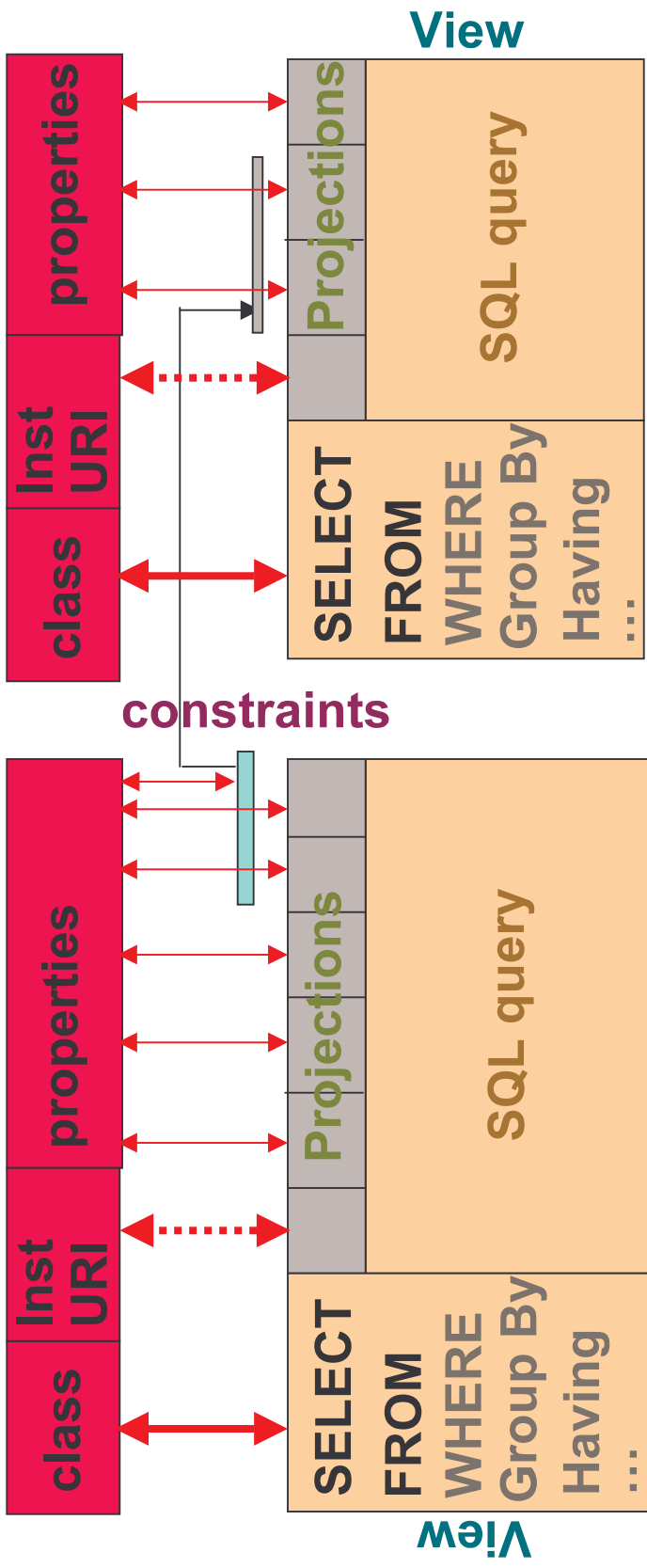
```
ConstraintColumn (ConsName, ViewName, ColName, ColPosInKey)
```

```
ConstraintColumn ("dept:c_unq_deptno", "xyz:dept", "dept:deptno", 1)
```

```
ConstraintColumn ("emp:c_prm_empno", "xyz:emp", "emp:empno", 1)
```

```
ConstraintColumn ("emp:c_ref_deptno", "xyz:emp", "emp:deptNum", 1)
```

Summary



• Proposed mapping language

(Leverage SQL on source database and if available, SQL on client side)

- Strong-DB → No new language. Employ some *conventions* with SQL (e.g., when *naming Views*, *View cols* (projections), and *constraints*).
- No-DB → Simple language to specify *mapping* between RDF classes, properties and *SQL queries*, *query projections*, *constraints*.



ORACLE IS THE **INFORMATION** COMPANY