Data Integration Systems

1. Federated Database Systems
2. Data Warehouses:
3. Mediator Systems:

Issues:
- Data type differences
- Value differences
- Semantic differences
- Missing values

**Federated Database Systems:**
- Independent data sources
- For n databases to talk to each other, need to write n(n-1) pieces of code to support queries between systems.

_e.g._:

Dealer1:
Needed(model, color, autoTrans)

Dealer2:
Autos(serial, model, color)
Options(serial, option)

For Dealer1 to query Dealer2 for needed cars, we can use the following:

```sql
for (each tuple (:m, :c, :a) in NeededCars) {
    if (:a = TRUE) {
        SELECT serial
        FROM Autos, Options
        WHERE Autos.serial = Options.serial AND
              Options.option = 'autoTrans' AND
              Autos.model = :m AND
              Autos.color = :c;
    } else {
        SELECT serial
        FROM Autos
        WHERE Autos.model = :m AND
              Autos.color = :c AND
              NOT EXISTS (SELECT * FROM Options
                           WHERE serial = Autos.serial AND
                           Options.option = 'autoTrans');
    }
}
```
Data Warehouses:
Basic Idea:
Use global schemas to accommodate local schemas.
Queries are based on global schemas.

- Warehouse is periodically reconstructed from the current data sources. The system may be unavailable during reconstruction.
- Incremental update is important.
- Minimize the staleness of data is important.

e.g.:

Dealer1: Cars(serialNo, model, color, autoTrans, cdPlayer, …)
Dealer2 Autos(serial, model, color)
          Options(serial, option)

Global schema: AutosWhse(serialNo, model, color, autoTrans, dealer)

Extractor for Dealer1:
\[
\begin{align*}
\text{INSERT INTO AutosWhse} & \text{ (serialNo, model, color, autoTrans, dealer) } \\
& \text{SELECT serialNo, model, color, autoTrans, ‘dealer1’ } \\
& \text{FROM Cars } \\
\end{align*}
\]

Extractor for Dealer2:
\[
\begin{align*}
\text{INSERT INTO AutosWhse} & \text{ (serialNo, model, color, autoTrans, dealer) } \\
& \text{SELECT serial, model, color, ‘yes’, ‘dealer2’ } \\
& \text{FROM Autos, Options } \\
& \text{WHERE Autos.serial = Options.serial AND} \\
& \text{option = ‘autoTrans’; } \\
\end{align*}
\]
\[
\begin{align*}
\text{INSERT INTO AutosWhse} & \text{ (serialNo, model, color, autoTrans, dealer) } \\
& \text{SELECT serial, model, color, ‘no’, ‘dealer2’ } \\
& \text{FROM Autos } \\
& \text{WHERE NOT EXISTS (} \\
& \text{SELECT * } \\
& \text{FROM Options } \\
& \text{WHERE serial = Autos.serial AND} \\
& \text{option = ‘autoTrans’ } \\
\end{align*}
\]
**Mediators:**

Basic Idea:

- Use virtual views to integrate data sources
- Do not store data in the mediator.

Issues:

- Require more complex wrappers than most warehouses.
- The wrapper must be able to accept a variety of queries from the mediator and translate any of them to the terms of the source.
- The wrapper must communicate the result to the mediator.

e.g.:

Integrated schema:  AutosMed(serialNo, model, color, autoTrans, dealer)

Query to mediator:

```sql
SELECT serialNo, model
FROM AutosMed
WHERE color = 'red';
```

Queries forwarded to wrappers:

Wrapper1:   Cars(serialNo, model, color, autoTrans, cdPlayer, …)

```sql
SELECT serialNo, model
FROM Cars
WHERE color = 'red';
```

Wrapper2:   Autos(serial, model, color)
Options(serial, option)

```sql
SELECT serial, model
FROM Autos
WHERE color = 'red';
```

Mediator: take union of the resulting sets and return the result to user.

**Design of Wrappers:**
Create Templates for Query Patterns:

Classify the possible queries that the mediator can ask into templates, which are queries with parameters that represent constants.

e.g.:

Dealer1 schema: Cars(serialNo, model, color, autoTrans, cdPlayer, …)

Mediator schema: AutosMed(serialNo, model, color, autoTrans, dealer)

Wrapper template for queries for cars of a given color:

```
SELECT * 
FROM AutosMed 
WHERE color = '$c';
```

```
SELECT serialNo, model, color, autoTrans, ‘dealer’
FROM Cars
WHERE color = ‘$c’;
```

Wrapper Generator: translate templates into executable codes.

Create a table that holds the various query patterns contained in the templates and the associated source queries.

Driver:

A driver is used in each wrapper to:

- Accept a query from the mediator.
- Search the table for a template that matches the query and instantiate the source query.
- Send source query to the data source and collect the response.
- Process the response if necessary, and return the result to the mediator.

Filters: used to support as many queries as possible by filtering the results of queries that sent to the source.

Wrapper template for queries for cars of a given model:

```
SELECT * 
FROM AutosMed 
WHERE model = ‘$m’;
```

```
SELECT serialNo, model, color, autoTrans, ‘dealer’
FROM Cars
WHERE model = ‘$m’;
```
Mediator query:

SELECT * 
FROM AutosMed 
WHERE color = 'blue' and model = 'Gobi';

Can be supported by previous templates using filters as follows: