An Abstract Interpretation Framework for Structured Query Languages

Agostino Cortesi, Raju Halder

Department of Computer Science
Université Ca’ Foscari di Venezia, Italy
{cortesi, halder}@unive.it
Outline

1. Introduction and Overview
2. Application embedded with SQL or PL/SQL
   - Abstract Syntax
   - Environments and States
   - Semantics of the Expressions
   - Semantics of the SQL Statements
   - Semantics of the composite statements
3. Syntax and Semantics over Abstract Domain
   - Abstract Syntax over Abstract Domain
   - Abstract Semantics
4. Conclusions
Introduction

We deal with application programs embedded with SQL or PL/SQL

Application deals with two worlds/environments:
- User world
- Database World

We define two sets of variables corresponding to these two worlds:
- Set of Application variables ($V_a$)
- Set of Database Variables ($V_d$)
Introduction

We deal with application programs embedded with SQL or PL/SQL

Application deals with two worlds/environments:
- User world
- Database World

We define two sets of variables corresponding to these two worlds:
- Set of Application variables ($V_a$)
- Set of Database Variables ($V_d$)
We deal with application programs embedded with SQL or PL/SQL.

Application deals with two worlds/environments:
- User world
- Database World

We define two sets of variables corresponding to these two worlds:
- Set of Application variables ($V_a$)
- Set of Database Variables ($V_d$)
We denote any SQL command $C_{sql} \triangleq \langle A_{sql}, \phi \rangle$

- We call the first component $A_{sql}$ the *active part* and the second component $\phi$ the *passive part* of $C_{sql}$.

- The pre-condition $\phi$ in SQL commands is a well-formed formula of first-order logic.

- In an abstract sense, $C_{sql}$ first identifies an active data set from the database using the pre-condition $\phi$ and then performs the appropriate operations on that data set using $A_{sql}$.
We denote any SQL command $C_{sql} \triangleq \langle A_{sql}, \phi \rangle$

We call the first component $A_{sql}$ the *active part* and the second component $\phi$ the *passive part* of $C_{sql}$.

The pre-condition $\phi$ in SQL commands is a well-formed formula of first-order logic.

In an abstract sense, $C_{sql}$ first identifies an active data set from the database using the pre-condition $\phi$ and then performs the appropriate operations on that data set using $A_{sql}$.
We denote any SQL command $C_{sql} \triangleq \langle A_{sql}, \phi \rangle$

- We call the first component $A_{sql}$ the **active part** and the second component $\phi$ the **passive part** of $C_{sql}$.

- The pre-condition $\phi$ in SQL commands is a well-formed formula of first-order logic.

- In an abstract sense, $C_{sql}$ first identifies an active data set from the database using the pre-condition $\phi$ and then performs the appropriate operations on that data set using $A_{sql}$. 

---

*Agostino Cortesi, Raju Halder*  
*An Abstract Interpretation Framework for Structured Query Languages*
Introduction

- We denote any SQL command $C_{sql} \triangleq \langle A_{sql}, \phi \rangle$

- We call the first component $A_{sql}$ the **active part** and the second component $\phi$ the **passive part** of $C_{sql}$.

- The pre-condition $\phi$ in SQL commands is a well-formed formula of first-order logic.

- In an abstract sense, $C_{sql}$ first identifies an active data set from the database using the pre-condition $\phi$ and then performs the appropriate operations on that data set using $A_{sql}$.
Overview: Example

<table>
<thead>
<tr>
<th>eID</th>
<th>Name</th>
<th>Age</th>
<th>Dno</th>
<th>Pno</th>
<th>Sal</th>
<th>Child − no</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Matteo</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>2000</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Sukriti</td>
<td>22</td>
<td>1</td>
<td>2</td>
<td>1500</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Joy</td>
<td>50</td>
<td>2</td>
<td>3</td>
<td>2300</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Luca</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>1700</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Deba</td>
<td>40</td>
<td>3</td>
<td>4</td>
<td>3000</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>Andrea</td>
<td>70</td>
<td>1</td>
<td>2</td>
<td>1900</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>Alberto</td>
<td>18</td>
<td>3</td>
<td>4</td>
<td>800</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>Sanu</td>
<td>14</td>
<td>2</td>
<td>3</td>
<td>4000</td>
<td>3</td>
</tr>
</tbody>
</table>

Table: Concrete Table: $t_{emp}$

$\alpha(\text{Age}) \triangleq \begin{cases} [5, 11] & \text{if } 5 \leq \text{age} \leq 11 \\ [12, 24] & \text{if } 12 \leq \text{age} \leq 24 \\ [25, 59] & \text{if } 25 \leq \text{age} \leq 59 \\ [60, 100] & \text{if } 60 \leq \text{age} \leq 100 \end{cases}$

$\alpha(\text{Sal}) \triangleq \begin{cases} [500, 1499] & \text{if } 500 \leq \text{Sal} \leq 1499 \\ [1500, 2499] & \text{if } 1500 \leq \text{Sal} \leq 2499 \\ [2500, 10000] & \text{if } 2500 \leq \text{Sal} \leq 10000 \end{cases}$

$\alpha(\text{Child − no}) \triangleq \begin{cases} \text{Few} & \text{if } 0 \leq \text{child − no} \leq 1 \\ \text{Medium} & \text{if } 2 \leq \text{child − no} \leq 3 \\ \text{many} & \text{if } 4 \leq \text{child − no} \leq 10 \end{cases}$

<table>
<thead>
<tr>
<th>eID</th>
<th>Name</th>
<th>Age</th>
<th>Dno</th>
<th>Pno</th>
<th>Sal</th>
<th>Child − no</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Matteo</td>
<td>[25,59]</td>
<td>2</td>
<td>1</td>
<td>[1500,2499]</td>
<td>many</td>
</tr>
<tr>
<td>2</td>
<td>Sukriti</td>
<td>[12,24]</td>
<td>1</td>
<td>2</td>
<td>[1500,2499]</td>
<td>Medium</td>
</tr>
<tr>
<td>3</td>
<td>Joy</td>
<td>[25,59]</td>
<td>2</td>
<td>3</td>
<td>[1500,2499]</td>
<td>Medium</td>
</tr>
<tr>
<td>4</td>
<td>Luca</td>
<td>[5,11]</td>
<td>1</td>
<td>2</td>
<td>[1500,2499]</td>
<td>Few</td>
</tr>
<tr>
<td>5</td>
<td>Deba</td>
<td>[25,59]</td>
<td>3</td>
<td>4</td>
<td>[2500,10000]</td>
<td>many</td>
</tr>
<tr>
<td>6</td>
<td>Andrea</td>
<td>[60,100]</td>
<td>1</td>
<td>2</td>
<td>[1500,2499]</td>
<td>Medium</td>
</tr>
<tr>
<td>7</td>
<td>Alberto</td>
<td>[12,24]</td>
<td>3</td>
<td>4</td>
<td>[500,1499]</td>
<td>Few</td>
</tr>
<tr>
<td>8</td>
<td>Sanu</td>
<td>[12,24]</td>
<td>2</td>
<td>3</td>
<td>[2500,10000]</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table: Abstract table: $t^\alpha_{emp}$

Agostino Cortesi, Raju Halder

An Abstract Interpretation Framework for Structured Query Languages
The abstraction is sound i.e. $t_{res} \in \gamma(t_{res}')$.

```
SELECT Dno, MAX(Sal), MIN(Sal), AVG(Age), COUNT(*), SUM(Child - no)
FROM temp
WHERE sal > 1600
GROUP BY (Dno, Pno)
HAVING MAX(sal) < 4000
```

<table>
<thead>
<tr>
<th>Dno</th>
<th>MAX(Sal)</th>
<th>MIN(Sal)</th>
<th>AVG(Age)</th>
<th>COUNT(*)</th>
<th>SUM(Child - no)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2000</td>
<td>2000</td>
<td>30</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3000</td>
<td>3000</td>
<td>40</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1900</td>
<td>1700</td>
<td>40</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table:** Resulting table: $t_{res}$ (concrete)

```
SELECT Dno#, MAX(Sal#), MIN(Sal#), AVG(Age#), COUNT(#), SUM(Child - no#)
FROM $t_{emp}^#$
WHERE sal# > [1500,1499]
GROUP BY (Dno#, Pno#)
HAVING MAX(sal#) < [2500,10000]
```

<table>
<thead>
<tr>
<th>Dno#</th>
<th>MAX(Sal#)</th>
<th>MIN(Sal#)</th>
<th>AVG(Age#)</th>
<th>COUNT(#)</th>
<th>SUM(Child - no#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[1500,2499]</td>
<td>[1500,2499]</td>
<td>[25,59]</td>
<td>1</td>
<td>many</td>
</tr>
<tr>
<td>3</td>
<td>[2500,10000]</td>
<td>[2500,10000]</td>
<td>[25,59]</td>
<td>1</td>
<td>many</td>
</tr>
<tr>
<td>1</td>
<td>[1500,2499]</td>
<td>[1500,2499]</td>
<td>[5,100]</td>
<td>3</td>
<td>$\top$</td>
</tr>
<tr>
<td>2</td>
<td>[2500,10000]</td>
<td>[1500,2499]</td>
<td>[12,59]</td>
<td>2</td>
<td>$\top$</td>
</tr>
</tbody>
</table>

**Table:** Resulting table: $t_{res}^#$ (abstract)
Overview: Example

- Suppose, $T$ and $T^\#$ represents a concrete and corresponding abstract tables respectively.

- The correspondence between $T$ and $T^\#$ are described using the concretization and abstraction maps $\gamma$ and $\alpha$ respectively.

- If $C_{sql}$ and $C^\#$ are representing the SQL queries on concrete and abstract domains respectively, let $T_{res}$ and $T^\#_{res}$ are the results of applying $C_{sql}$ and $C^\#$ on the $T$ and $T^\#$ respectively.

Soundness condition of abstraction:

$$
\begin{align*}
T & \xrightarrow{C_{sql}} T_{res} \subseteq \gamma(T^\#_{res}) \\
T^\# & \xrightarrow{C^\#} T^\#_{res}
\end{align*}
$$
Overview: Example

- In web-based service scenario to provide users only partial view rather than the complete view of the data in DB is permitted, and let them to download.

- Can serve as static analysis framework for the applications for optimization purpose (e.g. checking for integrity constraints)

- In Service Provider Model, deals with security and privacy issues (can serve as a means of encryption)

- Answering queries approximately as way to reduce query response time

- In online decision support system where precise answer is not helpful and early response is necessary

- To provide some statistical properties of data in the database
Outline

1. Introduction and Overview
2. Application embedded with SQL or PL/SQL
   - Abstract Syntax
   - Environments and States
   - Semantics of the Expressions
   - Semantics of the SQL Statements
   - Semantics of the composite statements
3. Syntax and Semantics over Abstract Domain
   - Abstract Syntax over Abstract Domain
   - Abstract Semantics
4. Conclusions
### Abstract Syntax: Syntactic Sets

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( \mathbb{Z} )</td>
<td>Integer</td>
</tr>
<tr>
<td>( k )</td>
<td>( S )</td>
<td>String</td>
</tr>
<tr>
<td>( c )</td>
<td>( C )</td>
<td>Constants</td>
</tr>
<tr>
<td>( v_a )</td>
<td>( V_a )</td>
<td>Application Variables</td>
</tr>
<tr>
<td>( v_d )</td>
<td>( V_d )</td>
<td>Database Variables (attributes)</td>
</tr>
<tr>
<td>( v )</td>
<td>( V \triangleq V_d \cup V_a )</td>
<td>Variables</td>
</tr>
<tr>
<td>( e )</td>
<td>( E )</td>
<td>Arithmetic Expressions</td>
</tr>
<tr>
<td>( b )</td>
<td>( B )</td>
<td>Boolean Expressions</td>
</tr>
<tr>
<td>( A_{sql} )</td>
<td>( A_{sql} )</td>
<td>SQL Actions (Active part of ( C_{sql} ))</td>
</tr>
<tr>
<td>( \tau )</td>
<td>( T )</td>
<td>Terms</td>
</tr>
<tr>
<td>( a_f )</td>
<td>( A_f )</td>
<td>Atomic Formulas</td>
</tr>
<tr>
<td>( \phi )</td>
<td>( W )</td>
<td>Well-formed formulas (passive part of ( C_{sql} ))</td>
</tr>
<tr>
<td>( C_{sql} )</td>
<td>( C_{sql} )</td>
<td>SQL Commands</td>
</tr>
<tr>
<td>( I )</td>
<td>( I )</td>
<td>Instructions/Commands</td>
</tr>
</tbody>
</table>

**Table:** Syntactic Sets
Abstract Syntax

\[
c ::= n | k
\]

\[
e ::= c | v_d | v_a | op \ e | e_1 \ op \ e_2
\]
where, \( op \) = arithmetic operator.

\[
b ::= e_1 = e_2 | e_1 \ op_r e_2 | \neg b | b_1 \lor b_2 | b_1 \land b_2 | true | false
\]
where, \( op_r \) = relational operator.

\[
\tau ::= c | v_a | v_d | f_n(\tau_1, \tau_2, \ldots, \tau_n) \text{ where, } f_n \text{ is an n-ary function.}
\]

\[
a_f ::= R_n(\tau_1, \tau_2, \ldots, \tau_n) | \tau_1 = \tau_2 \text{ where, } R_n \text{ is an n-ary relation.}
\]

\[
\phi ::= a_f | \neg \phi | \phi_1 \lor \phi_2 | \phi_1 \land \phi_2 | \forall x_i \phi | \exists x_i \phi
\]

\[
g(\vec{e}) ::= \text{GROUP BY}(\vec{e}) | \text{id}
\]

\[
f(\vec{e}) ::= \text{ORDER BY ASC}(\vec{e}) | \text{ORDER BY DESC}(\vec{e}) | \text{id}
\]

**Table:** Abstract syntax of the application programs
Abstract Syntax

\[
\begin{align*}
  r & ::= \text{DISTINCT} \mid \text{ALL} \\
  s & ::= \text{AVG} \mid \text{SUM} \mid \text{MAX} \mid \text{MIN} \mid \text{COUNT} \\
  h(e) & ::= s \circ r(e) \mid \text{DISTINCT}(e) \mid \text{id} \\
  h(*) & ::= \text{COUNT}(*) \\
  \tilde{h}(\tilde{x}) & ::= \langle h_1(x_1), \ldots, h_n(x_n) \rangle, \text{where } \tilde{h} = \langle h_1, \ldots, h_n \rangle \text{ and } \tilde{x} = \langle x_1, \ldots, x_n \rangle \\
  A_{\text{sql}} & ::= \text{select}(v_a, f(\tilde{e}'), r(\tilde{h}(\tilde{x})), \phi, g(\tilde{e}')) \mid \text{update}(\tilde{v_d}, \tilde{e}) \mid \text{insert}(\tilde{v_d}, \tilde{e}) \mid \text{delete} \\
  C_{\text{sql}} & ::= \langle A_{\text{sql}}, \phi \rangle \mid C'_{\text{sql}} \text{ UNION } C''_{\text{sql}} \mid C'_{\text{sql}} \text{ INTERSECT } C''_{\text{sql}} \mid C'_{\text{sql}} \text{ MINUS } C''_{\text{sql}} \\
  I & ::= \text{skip} \mid v_a := e \mid v_a := ? \mid C_{\text{sql}} \mid \text{if } b \text{ then } l_1 \text{ else } l_2 \mid \text{while } b \text{ do } l \mid l_1 ; l_2
\end{align*}
\]

Table: Abstract syntax of the application programs
If *SELECT* statement uses *GROUP BY*($\vec{e}$), then there must be an $\vec{h}(\vec{x})$ which is applied on each partition obtained by *GROUP BY* operation. In that case,

$$h_i \triangleq \begin{cases} \text{DISTINCT} & \text{if } x_i \in \vec{x} \cap \vec{e} \text{ or,} \\ \text{COUNT} & \text{if } x_i = * \text{ or,} \\ s \circ r & \text{otherwise} \end{cases}$$

When the *SELECT* statement does not use any *GROUP BY*($\vec{e}$) function, then $\vec{h}(\vec{x})$ is performed only over one group containing all tuples of the table. In that case, either each $h_i(\in \vec{h}) = id$ or

$$h_i \triangleq \begin{cases} \text{COUNT} & \text{if } x_i = * \text{ or,} \\ s \circ r & \text{otherwise} \end{cases}$$
If `SELECT` statement uses `GROUP BY(\vec{e})`, then there must be an \( \vec{h}(\vec{x}) \) which is applied on each partition obtained by `GROUP BY` operation. In that case, 

\[
\begin{align*}
    h_i \triangleq \begin{cases} 
    \text{DISTINCT} & \text{if } x_i \in \vec{x} \cap \vec{e} \text{ or,} \\
    \text{COUNT} & \text{if } x_i = * \text{ or,} \\
    s \circ r & \text{otherwise}
    \end{cases}
\end{align*}
\]

When the `SELECT` statement does not use any `GROUP BY(\vec{e})` function, then \( \vec{h}(\vec{x}) \) is performed only over one group containing all tuples of the table. In that case, either each \( h_i(\in \vec{h}) = id \) or 

\[
\begin{align*}
    h_i \triangleq \begin{cases} 
    \text{COUNT} & \text{if } x_i = * \text{ or,} \\
    s \circ r & \text{otherwise}
    \end{cases}
\end{align*}
\]
Outline

1. Introduction and Overview

2. Application embedded with SQL or PL/SQL
   - Abstract Syntax
   - Environments and States
     - Semantics of the Expressions
     - Semantics of the SQL Statements
     - Semantics of the composite statements

3. Syntax and Semantics over Abstract Domain
   - Abstract Syntax over Abstract Domain
   - Abstract Semantics

4. Conclusions
Environments

- The application program $P$ acts on:
  - set of constants $\text{const}(P) \in \emptyset(C)$
  - set of variables $\text{var}(P) \in \emptyset(V)$

- These variables take their values from semantic domain

$$\mathcal{D}_\emptyset = \{\mathcal{D} \cup \{\emptyset\}\}$$

where $\emptyset$ represents the undefined value.
Environments

Definition

(Application Environment)

An application environment $\rho_a \in \mathcal{C}_a$ maps a variable $v \in \text{dom}(\rho_a) \subseteq V_a$ to its value $\rho_a(v)$. So,

$\mathcal{C}_a \triangleq V_a \rightarrow \mathcal{D}_\mathcal{O}$
Environments

Definition

(Database Environment)

A database is a set of tables \( \{ t_i \mid i \in I \} \) for a given set of indexes \( I \). We may define a function \( \rho_d \) whose domain is \( I \), such that for \( i \in I \),

\[
\rho_d(i) = t_i
\]

Suppose, a database \( d \) consists of three tables \( t_{emp}, t_{dept}, t_{prj} \). Thus, the index set \( I \) is \( \{ emp, dept, prj \} \). So, \( \rho_d(emp) = t_{emp} \), for example.
Environments

Definition

(Database Environment)

A database is a set of tables \( \{ t_i \mid i \in I \} \) for a given set of indexes \( I \). We may define a function \( \rho_d \) whose domain is \( I \), such that for \( i \in I \),

\[
\rho_d(i) = t_i
\]

Suppose, a database \( d \) consists of three tables \( t_{emp}, t_{dept}, t_{prj} \). Thus, the index set \( I \) is \( \{ emp, dept, prj \} \). So, \( \rho_d(emp) = t_{emp} \), for example.
Environments

- Given a database environment $\rho_d$ and a table $t \in d$.
- Let $\text{attr}(t) = \{a_1, a_2, ..., a_k\}$. So, $t \subseteq D_1 \times D_2 \times ... \times D_k$ where, $a_i$ is the attribute corresponding to the typed domain $D_i$.

**Definition**

*(Table Environment)*

A table environment $\rho_t$ for a table $t \in DB$ is defined as a function such that for any attribute $a_i \in \text{attr}(t)$,

$$\rho_t(a_i) = \langle \pi_i(l_j) \mid l_j \in t \rangle$$

That is $\rho_t$ maps $a_i$ to the ordered set of values over the rows of the table $t$ where $j$ ranges over the list of rows in $t$.

In the example, $\text{dom}(\rho_{\text{emp}}) = \{\text{eID}, \text{Name}, \text{Age}, \text{Dno}, \text{Pno}, \text{Sal}\}$. So, for example, $\rho_{\text{emp}}(\text{age}) = \langle 28, 30, 27, 25, 29, 33, 35, 25 \rangle$. 
Environments

- Given a database environment $\rho_d$ and a table $t \in d$.
- Let $\text{attr}(t) = \{a_1, a_2, ..., a_k\}$. So, $t \subseteq D_1 \times D_2 \times ... \times D_k$ where, $a_i$ is the attribute corresponding to the typed domain $D_i$.

**Definition**

*(Table Environment)*

A table environment $\rho_t$ for a table $t \in DB$ is defined as a function such that for any attribute $a_i \in \text{attr}(t)$,

$$\rho_t(a_i) = \langle \pi_i(l_j) \mid l_j \in t \rangle$$

That is $\rho_t$ maps $a_i$ to the ordered set of values over the rows of the table $t$ where $j$ ranges over the list of rows in $t$.

In the example, $\text{dom}(\rho_{\text{temp}})=\{\text{eID}, \text{Name}, \text{Age}, \text{Dno}, \text{Pno}, \text{Sal}\}$. So, for example, $\rho_{\text{temp}}(\text{age})=\langle 28, 30, 27, 25, 29, 33, 35, 25 \rangle$. 

Agostino Cortesi, Raju Halder
*An Abstract Interpretation Framework for Structured Query Languages*
We define a state \( \sigma(\in \Sigma) \triangleq (l, \rho_d, \rho_a) \)

\( l = \) instruction to be executed;
\( \rho_d = \) database environment;
\( \rho_a = \) application environment.

\( \Sigma \triangleq I \times E_d \times E_a \)

The set of states for \( P: \Sigma[P] \triangleq P \times E_d \times E_a \).

The state transition relation: \( \Gamma \triangleq \Sigma \longmapsto \wp(\Sigma) \).

The transitional semantics of \( P: \Gamma[P] \triangleq \Sigma[P] \longmapsto \wp(\Sigma[P]) \).
### Outline

<table>
<thead>
<tr>
<th></th>
<th>Introduction and Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><strong>Application embedded with SQL or PL/SQL</strong></td>
</tr>
<tr>
<td></td>
<td>- Abstract Syntax</td>
</tr>
<tr>
<td></td>
<td>- Environments and States</td>
</tr>
<tr>
<td></td>
<td>- <strong>Semantics of the Expressions</strong></td>
</tr>
<tr>
<td></td>
<td>- Semantics of the SQL Statements</td>
</tr>
<tr>
<td></td>
<td>- Semantics of the composite statements</td>
</tr>
<tr>
<td>3</td>
<td><strong>Syntax and Semantics over Abstract Domain</strong></td>
</tr>
<tr>
<td></td>
<td>- Abstract Syntax over Abstract Domain</td>
</tr>
<tr>
<td></td>
<td>- Abstract Semantics</td>
</tr>
<tr>
<td>4</td>
<td>Conclusions</td>
</tr>
</tbody>
</table>
Semantics of the expressions

1. \( E[c](\rho_d, \rho_a) = c \)

2. \( E[v_a](\rho_d, \rho_a) = \rho_a(v_a) \)

3. \( \text{let } \exists t \in \text{dom}(\rho_d) : v_d = a_i \in \text{attr}(t) \text{ in} \)
   \[
   E[v_d](\rho_d, \rho_a) = E[v_d](\rho_t, \rho_a) = \rho_t(a_i)
   \]

4. \( \text{let } \exists t \in \text{dom}(\rho_d) : v_d = a_i \in \text{attr}(t) \text{ and } \text{op} : D_i \times D_j \rightarrow D_k \text{ in} \)
   \[
   E[v_d \text{ op } c](\rho_d, \rho_a) = E[v_d \text{ op } c](\rho_t, \rho_a) = \langle (m \text{ op } c) \in D_k \mid m \in \rho_t(a_i) \land a_i \in D_i \land c \in D_j \rangle
   \]
Semantics of the Expressions

1. \( E[c](\rho_d, \rho_a) = c \)

2. \( E[v_a](\rho_d, \rho_a) = \rho_a(v_a) \)

3. \( \text{let } \exists t \in \text{dom}(\rho_d) : v_d = a_i \in \text{attr}(t) \text{ in} \)

   \[
   E[v_d](\rho_d, \rho_a) = E[v_d](\rho_t, \rho_a) = \rho_t(a_i)
   \]

4. \( \text{let } \exists t \in \text{dom}(\rho_d) : v_d = a_i \in \text{attr}(t) \text{ and } \text{op} : D_i \times D_j \rightarrow D_k \text{ in} \)

   \[
   E[v_d \circ\text{op}\ c](\rho_d, \rho_a) = E[v_d \circ\text{op}\ c](\rho_t, \rho_a) = \langle (m \circ\text{op}\ c) \in D_k | m \in \rho_t(a_i) \land a_i \in D_i \land c \in D_j \rangle
   \]
Semantics of the expressions

1. $E[c](\rho_d, \rho_a) = c$

2. $E[v_a](\rho_d, \rho_a) = \rho_a(v_a)$

3. $\text{let } \exists t \in \text{dom}(\rho_d) : v_d = a_i \in \text{attr}(t) \text{ in}$

   $$E[v_d](\rho_d, \rho_a) = E[v_d](\rho_t, \rho_a) = \rho_t(a_i)$$

4. $\text{let } \exists t \in \text{dom}(\rho_d) : v_d = a_i \in \text{attr}(t) \text{ and op : } D_i \times D_j \rightarrow D_k \text{ in}$

   $$E[v_d \text{ op } c](\rho_d, \rho_a)$$
   $$= E[v_d \text{ op } c](\rho_t, \rho_a)$$
   $$= \langle (m \text{ op } c) \in D_k \mid m \in \rho_t(a_i) \land a_i \in D_i \land c \in D_j \rangle$$
Semantics of the expressions

1. \[ E[c](\rho_d, \rho_a) = c \]

2. \[ E[v_a](\rho_d, \rho_a) = \rho_a(v_a) \]

3. \[ \text{let } \exists t \in \text{dom}(\rho_d) : v_d = a_i \in \text{attr}(t) \text{ in} \]

   \[ E[v_d](\rho_d, \rho_a) = E[v_d](\rho_t, \rho_a) = \rho_t(a_i) \]

4. \[ \text{let } \exists t \in \text{dom}(\rho_d) : v_d = a_i \in \text{attr}(t) \text{ and } \text{op} : D_i \times D_j \rightarrow D_k \text{ in} \]

   \[ E[v_d \text{ op } c](\rho_d, \rho_a) = E[v_d \text{ op } c](\rho_t, \rho_a) = \langle (m \text{ op } c) \in D_k \mid m \in \rho_t(a_i) \land a_i \in D_i \land c \in D_j \rangle \]
Semantics of the expressions

5  let \( \exists t \in \text{dom}(\rho_d) \) : \( v_d = a_i \in \text{attr}(t) \) and \( \text{op} : D_i \times D_j \rightarrow D_k \) in

\[
E[v_d \text{ op } v_a]((\rho_d, \rho_a)) \\
= E[v_d \text{ op } v_a]((\rho_t, \rho_a)) \\
= (\langle m \text{ op } n \rangle \in D_k \mid m \in \rho_t(a_i) \land \rho_a(v_a) = n \land a_i \in D_i \land v_a \in D_j)
\]

6  let \( \exists t \in \text{dom}(\rho_d) \) : \( v_{d_1} = a_i, v_{d_2} = a_j, \{a_i, a_j\} \subseteq \text{attr}(t) \) and \( \text{op} : D_i \times D_j \rightarrow D_k \) in

\[
E[v_{d_1} \text{ op } v_{d_2}]((\rho_d, \rho_a)) \\
= E[v_{d_1} \text{ op } v_{d_2}]((\rho_t, \rho_a)) \\
= (\langle m_r \in D_k \mid m_r = \pi_i(l_r) \text{ op } \pi_j(l_r) \rangle \text{ where } l_r \text{ is the } r^{th} \text{ row of}
\]
Semantics of the expressions

5 \[\text{let } \exists t \in \text{dom}(\rho_d) : v_d = a_i \in \text{attr}(t) \text{ and } op : D_i \times D_j \rightarrow D_k \text{ in}\]

\[E[v_d \ op \ v_a](\rho_d, \rho_a) \]
\[= E[v_d \ op \ v_a](\rho_t, \rho_a) \]
\[= \langle (m \ op \ n) \in D_k \mid m \in \rho_t(a_i) \land \rho_a(v_a) = n \land a_i \in D_i \land v_a \in D_j \rangle \]

6 \[\text{let } \exists t \in \text{dom}(\rho_d) : v_{d_1} = a_i, v_{d_2} = a_j, \{a_i, a_j\} \subseteq \text{attr}(t) \text{ and } op : D_i \times D_j \rightarrow D_k \text{ in}\]

\[E[v_{d_1} \ op \ v_{d_2}](\rho_d, \rho_a) \]
\[= E[v_{d_1} \ op \ v_{d_2}](\rho_t, \rho_a) \]
\[= \langle m_r \in D_k \mid m_r = \pi_i(l_r) \ op \ \pi_j(l_r) \text{ where } l_r \text{ is the } r^{th} \text{ row of} \rangle \]
Semantics of the expressions

7 \[ E[e_1 \text{ op } e_2](\rho_d, \rho_a) \]

\[ = \begin{cases} 
\text{Case 1 :} \\
\exists! \ t \in \text{dom}(\rho_d) : \ \text{if } v_d \text{ occurs in } e_1 \text{ or } e_2 : \ v_d = a \in \text{attr}(t). \\
= E[e_1 \text{ op } e_2](\rho_t, \rho_a) \\
= E[e_1](\rho_t, \rho_a) \text{ op } E[e_2](\rho_t, \rho_a) 
\end{cases} \]

\[ \text{Case 2 :} \]
\[ \text{Let } T = \{ t \in \text{dom}(\rho_d) \mid \exists v_d \text{ occurring in } e_1 \text{ or } e_2 \ s.t. v_d = a \in \text{attr}(t) \}. \]
\[ \text{Let, } T = \{ t_1, t_2, \ldots, t_n \} \text{ and } t' = t_1 \times t_2 \times \ldots \times t_n. \]
\[ = E[e_1 \text{ op } e_2](\rho_{t'}, \rho_a) \]
Semantics of the expressions

1. $B[true](\rho_d, \rho_a) = true$

2. $B[false](\rho_d, \rho_a) = false$

3. $B[e_1 \ op_r \ e_2](\rho_d, \rho_a) = E[e_1](\rho_d, \rho_a) \ op_r \ E[e_2](\rho_d, \rho_a)$ where, $op_r$ represents the relational operator.

4. $B[\neg b](\rho_d, \rho_a) = \neg B[b](\rho_d, \rho_a)$

5. $B[b_1 \lor b_2](\rho_d, \rho_a) = B[b_1](\rho_d, \rho_a) \lor B[b_2](\rho_d, \rho_a)$

6. $B[b_1 \land b_2](\rho_d, \rho_a) = B[b_1](\rho_d, \rho_a) \land B[b_2](\rho_d, \rho_a)$
Outline

1. Introduction and Overview

2. Application embedded with SQL or PL/SQL
   - Abstract Syntax
   - Environments and States
   - Semantics of the Expressions
   - Semantics of the SQL Statements
   - Semantics of the composite statements

3. Syntax and Semantics over Abstract Domain
   - Abstract Syntax over Abstract Domain
   - Abstract Semantics

4. Conclusions
Semantics of SELECT

\[ S[\langle \text{select}(v_a, f(e'), r(h(x)), \phi_2, g(e)), \phi_1 \rangle]_c(\rho_d, \rho_a) = \begin{cases} 
S[\langle \text{select}(v_a, f(e'), r(h(x)), \phi_2, g(e)), \phi_1 \rangle]_c(\rho_t, \rho_a) 
\text{if } \exists! \ t \in \text{dom}(\rho_d) : \text{target}(\langle \text{select}(v_a, f(e'), r(h(x)), \phi_2, g(e)), \phi_1 \rangle) = \{t\} \\
S[\langle \text{select}(v_a, f(e'), r(h(x)), \phi_2, g(e)), \phi_1 \rangle]_c(\rho_{t'}, \rho_a) \text{ otherwise, where} \\
T = \{t_1, \ldots, t_n \in \text{dom}(\rho_d) \mid t_i \text{ occurs in } C_{\text{select}}\} \text{ and } t' = t_1 \times t_2 \times \cdots \times t_n. 
\end{cases} \]
Semantics of SELECT

Step 1. Absorbing $\phi_1$:

$$S\[\langle \text{select}(v_a, f(\vec{e'}), r(\vec{h}(\vec{x})), \phi_2, g(\vec{e})), \phi_1\rangle\]_c(\rho_t, \rho_a)$$

$$= S\[\langle \text{select}(v_a, f(\vec{e'}), r(\vec{h}(\vec{x})), \phi_2, g(\vec{e})), \text{true}\rangle\]_c(\rho_{t'}, \rho_a)$$

where, $t' = \langle l_i \in t_0 \mid \text{let } \text{var}(\phi_1) = \vec{v}'_d \cup \vec{v}'_a \text{ with } \vec{v}'_d = \vec{a} \subseteq \text{attr}(t_0) :$

$$\exists \phi_1[\pi_{\vec{a}}(l_i)/\vec{v}'_d] \rho_a(\vec{v}'_a)/\vec{v}'_a ]$$

Step 2. Grouping:

$$S\[\langle \text{select}(v_a, f(\vec{e'}), r(\vec{h}(\vec{x})), \phi, g(\vec{e})), \text{true}\rangle\]_c(\rho_t, \rho_a)$$

$$= S\[\langle \text{select}(v_a, f(\vec{e'}), r(\vec{h}(\vec{x})), \phi, \text{id}), \text{true}\rangle\]_c(\bigcup_i \rho_{t_i}, \rho_a)$$

$$= S\[\langle \text{select}(v_a, f(\vec{e'}), r(\vec{h}(\vec{x})), \phi, \text{id}), \text{true}\rangle\]_c(\rho_T, \rho_a)$$

where, $g(\vec{e}) = \text{Group By}(\vec{e})$ and $\text{Group By}(\vec{e})[t]$ is the maximal partition $T = \{t_1, t_2, ..., t_n\}$ of $t$, s.t. $\forall t_i \in T, t_i \subseteq t$ and

$$\forall e_i \in \vec{e}, \forall m_k, m_l \in E[e_i](\rho_{t_i}, \rho_a) : m_k = m_l$$
Semantics of SELECT

Step 1. Absorbing $\phi_1$:

$$S\left[\langle \text{select}(v_a, f(e'), r(h(x)), \phi_2, g(e)), \phi_1 \rangle \right]_c(\rho_t, \rho_a)$$

$$= S\left[\langle \text{select}(v_a, f(e'), r(h(x)), \phi_2, g(e)), \text{true} \rangle \right]_c(\rho_t', \rho_a)$$

where, $t' = \langle l_i \in t_0 \mid \text{let var}(\phi_1) = \vec{v}^d \cup \vec{v}^a \text{ with } \vec{v}^d = \vec{a} \subseteq \text{attr}(t_0) : \right.$

$$\Rightarrow \ c \models \phi_1[\pi_{\vec{a}}(l_i)/\vec{v}^d][\rho_a(\vec{v}^a)/\vec{v}^a]$$

Step 2. Grouping:

$$S\left[\langle \text{select}(v_a, f(e'), r(h(x)), \phi, g(e)), \text{true} \rangle \right]_c(\rho_t, \rho_a)$$

$$= S\left[\langle \text{select}(v_a, f(e'), r(h(x)), \phi, \text{id} \rangle \right]_c(\bigcup_{i \rho_t_i, \rho_a}$$

$$= S\left[\langle \text{select}(v_a, f(e'), r(h(x)), \phi, \text{id} \rangle \right]_c(\rho_T, \rho_a)$$

where, $g(e) = \text{Group By}(\vec{e})$ and Group By$(\vec{e})[t]$ is the maximal partition

$$T = \{t_1, t_2, ..., t_n\} \text{ of } t, \text{ s.t. } \forall t_i \in T, t_i \subseteq t \text{ and }$$

$$\forall e_i \in \vec{e}, \forall m_k, m_l \in E[e_i](\rho_{t_i}, \rho_a) : m_k = m_l$$
Step 3. Absorbing $\phi$:

$$S[[\text{select}(v_a, f(e'), r(h(x))), \phi, id), true]]_c(\rho_T, \rho_a)$$

$$= S[[\text{select}(v_a, f(e'), r(h(x))), \phi, id), true]]_c(\cup_i \rho_{t_i}, \rho_a) \ [t_i \in T]$$

$$= S[[\text{select}(v_a, f(e'), r(h(x))), true, id), true]]_c(\rho_{T'}, \rho_a)$$

where $T'$ is defined as follows:

There is a sequence of functions $h'$ occurring in $\phi$, operating on each group, such that:

$$h'(x') \ni h'_i(x'_i) \triangleq \begin{cases} 
\text{DISTINCT}(e) \text{ or,} \\
\text{COUNT}(*) \text{ or,} \\
 s \circ r(e)
\end{cases}$$

Let $\hat{v}_a$ be a sequence of application variables occurring in $\phi$ and, $\forall t_i \in T$, $h'(\langle E[\hat{x}'](\rho_{t_i}, \rho_a)\rangle) = \tilde{c}_i$ and, $T' = \{t_i \in T \mid \varsigma \models \phi[\tilde{c}_i/h'(\hat{x}')] [\rho_a(\hat{v}_a)/\hat{v}_a]\}$
Semantics of the SELECT

Step 4. Applying $r(\vec{h}(\vec{x}))$ on each group:

$$
= S[\langle \text{select}(v_a, f(\vec{e}'), r(\vec{h}(\vec{x})), \text{true}, \text{id}), \text{true} \rangle]_{c}(\rho_T, \rho_a)
= S[\langle \text{select}(v_a, f(\vec{e}'), \text{id}, \text{true}, \text{id}), \text{true} \rangle]_{c}(\rho_t, \rho_a)
$$

where, \( t' = \langle \vec{h}( E[\vec{x}](\rho_{t_i}, \rho_a)) | t_i \in T \rangle \) and \( t = r[t'] \)

Step 5. Possibly applying the ordering:

$$
S[\langle \text{select}(v_a, f(\vec{e}'), \text{id}, \text{true}, \text{id}), \text{true} \rangle]_{c}(\rho_t, \rho_a) = S[\langle \text{select}(v_a, \text{id}, \text{id}, \text{true}, \text{id}), \text{true} \rangle]_{c}(\rho_{t'}, \rho_a) \text{ where, } t' = f(\vec{e})[t]
$$

Step 6. Set the resulting values to the Record/ResultSet type application variable \( v_a \) with fields \( \vec{w} \):

$$
S[\langle \text{select}(v_a, \text{id}, \text{id}, \text{true}, \text{id}), \text{true} \rangle]_{c}(\rho_t, \rho_a) = (\rho_{t_0}, \rho_{a'})
$$

where, \( \rho_{a'} = \rho_a[\rho_t(\vec{a})/v_a(\vec{w})] \) with \( \vec{a} = \text{attr}(t) \) and \( t_0 = \text{initial table of step 1} \).

Here, the \( i^{th} \) field \( w_i \in \vec{w} \) is substituted by the values of \( i^{th} \) attribute \( a_i \in \vec{a} \).
Step 4. Applying $r(h(x^*))$ on each group:

$$S[\langle \text{select}(v_a, f(e^*'), r(h(x)), \text{true}, \text{id}), \text{true} \rangle]_c(\rho_T, \rho_a)$$

$$= S[\langle \text{select}(v_a, f(e^*'), \text{id}, \text{true}, \text{id}), \text{true} \rangle]_c(\rho_t, \rho_a)$$

where, $t' = \langle h(E[x^*](\rho_t, \rho_a)) \mid t_i \in T \rangle$ and $t = r[t']$

Step 5. Possibly applying the ordering:

$$S[\langle \text{select}(v_a, f(e^*'), \text{id}, \text{true}, \text{id}), \text{true} \rangle]_c(\rho_t, \rho_a)$$

$$= S[\langle \text{select}(v_a, \text{id}, \text{id}, \text{true}, \text{id}), \text{true} \rangle]_c(\rho_{t'}, \rho_a)$$

where, $t' = f(e)[t]$ where $\rho_{a'} = \rho_a[\rho_t(\tilde{a})/v_a(\tilde{w})]$ with $\tilde{a} = \text{attr}(t)$ and $t_0 = \text{initial table of step 1}$.

Here, the $i^{th}$ field $w_i \in \tilde{w}$ is substituted by the values of $i^{th}$ attribute $a_i \in \tilde{a}$.
Semantics of the SELECT

Step 4. Applying $r(h(\vec{x}))$ on each group:

$$S[\langle \text{select}(v_a, f(\vec{e}'), r(h(\vec{x})), true, id), true \rangle]_\varsigma(\rho_T, \rho_a)
= S[\langle \text{select}(v_a, f(\vec{e}'), id, true, id), true \rangle]_\varsigma(\rho_t, \rho_a)$$

where, $t' = \langle h(E[\vec{x}](\rho_{t_i}, \rho_a)) \mid t_i \in T \rangle$ and $t = r[t']$

Step 5. Possibly applying the ordering:

$$S[\langle \text{select}(v_a, f(\vec{e}'), id, true, id), true \rangle]_\varsigma(\rho_t, \rho_a)
= S[\langle \text{select}(v_a, id, id, true, id), true \rangle]_\varsigma(\rho_{t'}, \rho_a)$$

where, $t' = f(\vec{e})[t]$

Step 6. Set the resulting values to the Record/ResultSet type application variable $v_a$ with fields $\vec{w}$:

$$S[\langle \text{select}(v_a, id, id, true, id), true \rangle]_\varsigma(\rho_t, \rho_a) = (\rho_{t_0}, \rho_{a'})$$

where, $\rho_{a'} = \rho_a[\rho_t(\vec{a})/v_a(\vec{w})]$ with $\vec{a} = \text{attr}(t)$ and $t_0 = \text{initial table of step 1}$.

Here, the $i^{th}$ field $w_i \in \vec{w}$ is substituted by the values of $i^{th}$ attribute $a_i \in \vec{a}$. 
Semantics of the UPDATE

Step 1: Absorbing $\phi$:

$$S[[\langle \text{update}(\vec{v}_d, \vec{e}), \phi \rangle]]_\zeta(\rho_t, \rho_a) = S[[\langle \text{update}(\vec{v}_d, \vec{e}), \text{true} \rangle]]_\zeta(\rho_t', \rho_a)$$

where, $t' = \langle l_i \in t \mid \text{let } \var{\phi} = \vec{v}_d \cup \vec{a} \text{ with } \vec{v}_d = \vec{a} \subseteq \text{attr}(t) : \zeta \models \phi[\pi_{\vec{a}}(l_i)/\vec{v}_d][\rho_a(\vec{v}_d)/\vec{v}_a]\rangle$

Step 2: Update:

$$S[[\langle \text{update}(\vec{v}_d, \vec{e}), \text{true} \rangle]]_\zeta(\rho_t, \rho_a) = (\rho_t', \rho_a) \text{ where,}$$

let $\vec{v}_d = \vec{a} \subseteq \text{attr}(t)$ and $\vec{e} = \langle e_1, e_2, \ldots, e_n \rangle$ and $E[\vec{e}](\rho_t, \rho_a) = \langle \vec{m}_i \mid i = 1, \ldots, h \rangle$. Let $m_i^j$ be the $j^{th}$ element of the sequence $\vec{m}_i$ and $a_i$ be the $i^{th}$ element of the sequence $\vec{a}$.

$$t' \triangleq \langle l_j[m_i^j/a_i] \mid l_j \in t \rangle$$
Semantics of the INSERT

\[
S[\langle \text{insert}(\vec{v}_d, \vec{e}), \phi \rangle \xi(\rho_d, \rho_a)] = S[\langle \text{insert}(\vec{v}_d, \vec{e}), \phi \rangle \xi(\rho_t, \rho_a)] = S[\langle \text{insert}(\vec{v}_d, \vec{e}), \text{true} \rangle \xi(\rho_t, \rho_a)] = (\rho_{t'}, \rho_a)
\]

where,

let \( \vec{v}_d \triangleq \vec{a} \subseteq \text{attr}(t) \) and \( E[\vec{e}](\rho_a) = \vec{x} \),

\( \vec{a} = \langle a_1, a_2, ..., a_n \rangle \); \( \vec{x} = \langle x_1, x_2, ..., x_n \rangle \) and \( l_{\text{new}} = \langle x_1/a_1, x_2/a_2, ..., x_n/a_n \rangle \) in \( t' \triangleq t \cup \{l_{\text{new}}\} \)
\[
S[\langle \text{delete}, \phi \rangle]_\zeta(\rho_d, \rho_a) = S[\langle \text{delete}, \phi \rangle]_\zeta(\rho_t, \rho_a) = (\rho_{t'}, \rho_a)
\]

where, \( t_d = \langle l_i \in t \mid \text{let var}(\phi) = \vec{v}_d \cup \vec{v}_a \text{ with } \vec{v}_d = \vec{a} \subseteq \text{attr}(t) : \zeta \models \phi[\pi_{\vec{a}}(l_i)/\vec{v}_d][\rho_a(\vec{v}_a)/\vec{v}_a] \rangle \)

\[
t' = t \setminus t_d
\]
Outline

1. Introduction and Overview

2. Application embedded with SQL or PL/SQL
   - Abstract Syntax
   - Environments and States
   - Semantics of the Expressions
   - Semantics of the SQL Statements
   - Semantics of the composite statements

3. Syntax and Semantics over Abstract Domain
   - Abstract Syntax over Abstract Domain
   - Abstract Semantics

4. Conclusions
Semantics of the composite statements

\[
S[\text{SQL}_1](\rho_d, \rho_a) = t_1 \quad S[\text{SQL}_2](\rho_d, \rho_a) = t_2 \\
\frac{\text{S[SQL}_1 \text{ UNION SQL}_2](\rho_d, \rho_a) = t_1 \cup t_2}
\]

\[
S[\text{SQL}_1](\rho_d, \rho_a) = t_1 \quad S[\text{SQL}_2](\rho_d, \rho_a) = t_2 \\
\frac{\text{S[SQL}_1 \text{ INTERSECT SQL}_2](\rho_d, \rho_a) = t_1 \cap t_2}
\]

\[
S[\text{SQL}_1](\rho_d, \rho_a) = t_1 \quad S[\text{SQL}_2](\rho_d, \rho_a) = t_2 \\
\frac{\text{S[SQL}_1 \text{ MINUS SQL}_2](\rho_d, \rho_a) = t_1 \setminus t_2}
\]

\[
S[A_1](\rho_d, \rho_a) = (\rho_d', \rho_a') \quad S[A_2](\rho_d, \rho_a) = (\rho_d'', \rho_a'') \\
\frac{\text{S[A}_1; A_2](\rho_d, \rho_a) = (\rho_d'', \rho_a'')}
Outline

1. Introduction and Overview
2. Application embedded with SQL or PL/SQL
   - Abstract Syntax
   - Environments and States
   - Semantics of the Expressions
   - Semantics of the SQL Statements
   - Semantics of the composite statements
3. Syntax and Semantics over Abstract Domain
   - Abstract Syntax over Abstract Domain
   - Abstract Semantics
4. Conclusions
Abstract Syntax over Abstract Domain

Syntax of SQL Statement ($C^\#$) and SQL Action ($A^\#$) over an abstract domain:

$$A^\# ::= \text{select}^\#(v^\#, f^\#(e^\#), r^\#(h^\#(x^\#)), \phi^\#, g^\#(e^\#)) \mid \text{update}^\#(v^\#, e^\#) \mid \text{insert}^\#(v^\#, e^\#) \mid \text{delete}^\#$$

$$C^\# ::= \langle A^\#, \phi^\# \rangle \mid C_1^\# \text{ UNION } C_2^\# \mid C_1^\# \text{ INTERSECT } C_2^\# \mid C_1^\# \text{ MINUS } C_2^\#$$
Different abstract functions/operators involved in $A^\#$ are:

\[
\begin{align*}
g^\# & ::= \text{GROUP BY}^\# \mid \text{id} \\
r^\# & ::= \text{DISTINCT}^\# \mid \text{ALL}^\# \\
s^\# & ::= \text{AVG}^\# \mid \text{SUM}^\# \mid \text{MAX}^\# \mid \text{MIN}^\# \mid \text{COUNT}^\# \\
h^\#(e^\#) & ::= s^\# \circ r^\#(e^\#) \mid \text{DISTINCT}^\#(e^\#) \mid \text{id} \\
h^\#(\ast) & ::= \text{COUNT}^\#(\ast) \\
f^\# & ::= \text{ORDER BY ASC}^\# \mid \text{ORDER BY DESC}^\# \mid \text{id}
\end{align*}
\]
Arithmetic and Boolean expressions in abstract domain:

\[ c^\# ::= n^\# | k^\# \]
\[ e^\# ::= c^\# | v_d^\# | v_a^\# | op^\# e^\# | e_1^\# op^\# e_2^\#; \text{ where, } op^\# = \text{abstract arithmetic operator.} \]

\[ b^\# ::= e_1^\# op_r^\# e_2^\# | \neg b^\# | b_1^\# \lor b_2^\# | b_1^\# \land b_2^\# | \text{true} | \text{false} | T; \text{ where, } op_r^\# = \text{abstract relational operator.} \]
Abstract Syntax over Abstract Domain

Well-formed Formula and Instructions in abstract domain:

\[ \tau^\# ::= c^\# | v_a^\# | v_d^\# | f_n^\#(\tau_1^\#, \tau_2^\#, ..., \tau_n^\#); \quad \text{where, } f_n^\# \text{ is an } n - \text{ary function.} \]

\[ a_i^\# ::= R_n^\#(\tau_1^\#, \tau_2^\#, ..., \tau_n^\#) \mid \tau_1^\# = \tau_2^\#; \quad \text{where } R_n^\# \text{ is an } n - \text{ary relation: } R_n^\#(\tau_1^\#, \tau_2^\#, ..., \tau_n^\#) \in \{true, false, \top\} \]

\[ \phi^\# ::= a_i^\# \mid \neg \phi_1^\# \mid \phi_1^\# \lor \phi_2^\# \mid \phi_1^\# \land \phi_2^\# \mid \forall x_i^\# \phi_1^\# \mid \exists x_i^\# \phi_1^\# \]

\[ I^\# ::= skip \mid v_a^\# := e^\# \mid v_a^\# := ? \mid C^\# \mid if b^\# then I_1^\# else I_2^\# \mid while b^\# do I^\# \mid I_1^\# ; I_2^\# \]
Outline

1. Introduction and Overview

2. Application embedded with SQL or PL/SQL
   - Abstract Syntax
   - Environments and States
   - Semantics of the Expressions
   - Semantics of the SQL Statements
   - Semantics of the composite statements

3. Syntax and Semantics over Abstract Domain
   - Abstract Syntax over Abstract Domain
   - Abstract Semantics

4. Conclusions
Abstract Semantics: Soundness and Completeness

Theorem

(Soundness and Completeness)

Let $\gamma$ be a representation function.

The soundness and completeness conditions for an abstract functions/operators $f^\#$ in abstract domain with respect to the corresponding concrete function $f$ are:

- $f^\#$ is sound if $\gamma \circ f^\# \supseteq f \circ \gamma$
- $f^\#$ is complete if $\gamma \circ f^\# = f \circ \gamma$
Abstract Semantics

The correspondence between \( g \) and \( g^\# \):

\[
g \ ::= \text{GROUP BY } | \ id \\
g^\# \ ::= \text{GROUP BY}^\# | \ id
\]

- \( t^\# \) = Abstract table obtained from \( t \).
- \( e^\# \) = abstract version of \( e \).

- If \( g^\# \) is applied on \( t^\# \) based on \( e^\# \), then the number of partitions in abstract domain would be less as we are losing some information due to abstraction.

- Some partitions in concrete domain will be merged together in abstract domain.

- Hence we can write, \( \gamma \circ g^\# \sqsubseteq g \circ \gamma \).
The correspondence between $g$ and $g^\#$:

- $g ::= \text{GROUP BY} | id$
- $g^\# ::= \text{GROUP BY}^\# | id$

- $t^\# = \text{Abstract table obtained from } t.$
- $e^\# = \text{abstract version of } \vec{e}.$

If $g^\#$ is applied on $t^\#$ based on $e^\#$, then the number of partitions in abstract domain would be less as we are losing some information due to abstraction.

Some partitions in concrete domain will be merged together in abstract domain.

Hence we can write, $\gamma \circ g^\# \sqsubseteq g \circ \gamma.$
The correspondence between $r$ and $r^\#$: 

\[
\begin{align*}
  r & ::= \text{DISTINCT} \mid \text{ALL} \\
  r^\# & ::= \text{DISTINCT}^\# \mid \text{ALL}^\#
\end{align*}
\]

- The function $r$ is used to deal with the duplication.

- Whenever the representation function is injective, $r$ is complete 
  \( i.e. \gamma \circ r^\# = r \circ \gamma \).
The correspondence between $s$ and $s^\#$:

$s ::= \text{AVG} \mid \text{SUM} \mid \text{MAX} \mid \text{MIN} \mid \text{COUNT}$

$s^\# ::= \text{AVG}^\# \mid \text{SUM}^\# \mid \text{MAX}^\# \mid \text{MIN}^\# \mid \text{COUNT}^\#$

For all these arithmetic operations $s$, the corresponding abstract operations $s^\#$ has to be provided, satisfying

$$s^\#(X^\#) \sqsubseteq \text{lub}\{s(X) \mid X \in \gamma(X^\#)\}$$
Abstract Semantics

The correspondence between $f$ and $f^#$:

$$
\begin{align*}
  f & ::= \text{ORDER BY ASC} \mid \text{ORDER BY DESC} \mid id \\
  f^# & ::= \text{ORDER BY ASC}^# \mid \text{ORDER BY DESC}^# \mid id
\end{align*}
$$

- $f$ applied over a set of rows of a table $t$ and it sorts them based on a sequence of expressions $\bar{e}$ in ascending or descending order.

- As the representation function $\gamma$ is assumed to be monotonic, the function $f$ is complete in the sense that

$$
\gamma \circ f^# = f \circ \gamma
$$
The correspondence between $R_n$ and its abstract version $R^\#_n$ involved in $\phi$ and $\phi^\#$ respectively, should guarantee that,

If $R^\#_n(\tau_1^\# , \cdots , \tau_n^\#)$ is true, then

$$\forall \tau_1 \in \gamma(\tau_1^\#), \cdots , \tau_n \in \gamma(\tau_n^\#) : R_n(\tau_1, \cdots , \tau_n) = true$$

If $R^\#_n(\tau_1^\# , \cdots , \tau_n^\#)$ is false, then

$$\forall \tau_1 \in \gamma(\tau_1^\#), \cdots , \tau_n \in \gamma(\tau_n^\#) : R_n(\tau_1, \cdots , \tau_n) = false.$$
The values of abstract pre-condition $\phi^\#$ belong to the set \{true, false, $\top$\} according to the actual three-valued propositional logics.

For instance, if we consider the binary relation ’$<$’ among integers and the abstract domain is the domain of intervals, then the abstract relation ’$<$\#'’ corresponding to ’$<$’ is defined as follows:

$$[l_i, h_i] < \# [l_j, h_j] \triangleq \begin{cases} 
true & \text{if } h_i < l_j \\
false & \text{if } h_j \leq l_i \\
\top & \text{otherwise}
\end{cases}$$
The correspondence between $f_n$ and $f_n^{\#}$ involved in terms of well-formed-formula:

- This correspondence is obviously sound and depends upon the abstract domain.
- For example, consider an abstract domain with property $PARITY$. The concrete operation ‘$\times$’ over concrete domain is mapped to its abstract version as:

\[
odd(\times^{\#})odd = odd, \quad even(\times^{\#})odd = even, \quad even(\times^{\#})even = even
\]
Suppose, $T$ and $T^\#$ represents a concrete and corresponding abstract tables respectively.

The correspondence between $T$ and $T^\#$ are described using the concretization and abstraction maps $\gamma$ and $\alpha$ respectively.

If $C_{sql}$ and $C^\#$ are representing the SQL queries on concrete and abstract domains respectively, let $T_{res}$ and $T^\#_{res}$ are the results of applying $C_{sql}$ and $C^\#$ on the $T$ and $T^\#$ respectively.

Soundness condition of abstraction:

$$
T \xrightarrow{C_{sql}} T_{res} \subseteq \gamma(T^\#_{res})
$$

$$
T^\# \xrightarrow{C^\#} T^\#_{res} \subseteq \gamma(T^\#_{res})
$$
Abstract Semantics: Soundness

Theorem

(Soundness)
Let $T^\#$ be an abstract table and $C^\#$ be an abstract query. $C^\#$ is sound iff

$$\forall T \in \gamma(T^\#), \forall C_{sql} \in \gamma(C^\#) : C_{sql}(T) \subseteq \gamma(C^\#(T^\#))$$
Conclusions

As far as we know, this is the first attempt to formalize a Concrete/Abstract semantics for SQL query languages within the Abstract Interpretation framework.

The semantic description using RA or RC covers only a subset of SQL. This motivates our theoretical work aiming at defining a complete denotational semantics of SQL or PL/SQL embedded applications and an Abstract Interpretation based framework to cover these lack-points.

This approach can easily be applied in the web-services scenario in order to provide users either partial views or "customized replicas" of the database, where only the abstract values of the database are let available for downloading.
Conclusions

- As far as we know, this is the first attempt to formalize a Concrete/Abstract semantics for SQL query languages within the Abstract Interpretation framework.

- The semantic description using RA or RC covers only a subset of SQL. This motivates our theoretical work aiming at defining a complete denotational semantics of SQL or PL/SQL embedded applications and an Abstract Interpretation based framework to cover these lack-points.

- This approach can easily be applied in the web-services scenario in order to provide users either partial views or "customized replicas" of the database, where only the abstract values of the database are let available for downloading.
Conclusions

- As far as we know, this is the first attempt to formalize a Concrete/Abstract semantics for SQL query languages within the Abstract Interpretation framework.

- The semantic description using RA or RC covers only a subset of SQL. This motivates our theoretical work aiming at defining a complete denotational semantics of SQL or PL/SQL embedded applications and an Abstract Interpretation based framework to cover these lack-points.

- This approach can easily be applied in the web-services scenario in order to provide users either partial views or "customized replicas" of the database, where only the abstract values of the database are let available for downloading.
Conclusions

- Can serve as static analysis framework for the application for optimization purpose (e.g. checking for integrity constraints)

- Can serve as analysis of data privacy and security issues in Service-Provider Model (Provides a means of encryption)

- Formal foundation of "answering queries approximately as a way to reduce query response times in on-line decision support systems, when the precise answer is not necessary or early feedback is helpful".
Conclusions

- Can serve as static analysis framework for the application for optimization purpose (e.g. checking for integrity constraints)

- Can serve as analysis of data privacy and security issues in Service-Provider Model (Provides a means of encryption)

- Formal foundation of "answering queries approximately as a way to reduce query response times in on-line decision support systems, when the precise answer is not necessary or early feedback is helpful".
Conclusions

- Can serve as static analysis framework for the application for optimization purpose (e.g. checking for integrity constraints)

- Can serve as analysis of data privacy and security issues in Service-Provider Model (Provides a means of encryption)

- Formal foundation of "answering queries approximately as a way to reduce query response times in on-line decision support systems, when the precise answer is not necessary or early feedback is helpful".
Thank You all!