CS 525: Advanced Database Organisation

08: Query Processing
Parsing and Analysis

Boris Glavic
Slides: adapted from a course taught by Hector Garcia-Molina, Stanford InfoLab

Parsing, Analysis, Conversion
1. Parsing
   - Transform SQL text into syntax tree
2. Analysis
   - Check for semantic correctness
   - Use database catalog
   - E.g., unfold views, lookup functions and attributes, check scopes
3. Conversion
   - Transform into internal representation
   - Relational algebra or QBM

Analysis and Conversion
- Usually intertwined
- The internal representation is used to store analysis information
- Create an initial representation and complete during analysis

Parsing
- SQL -> Parse Tree
- Covered in compiler courses and books
- Here only short overview
SQL Standard

- Standardized language
  - 86, 89, 92, 99, 03, 06, 08, 11
- DBMS vendors developed their own dialects

Example: SQL query

```sql
SELECT title
FROM StarsIn
WHERE starName IN (SELECT name
                    FROM MovieStar
                    WHERE birthdate LIKE '%1960');
```

(Find the movies with stars born in 1960)

Example: Parse Tree

```
<Query>
    <Query Block>
        SELECT title FROM StarsIn
        WHERE starName IN (SELECT name FROM MovieStar
                            WHERE birthdate LIKE '%1960');
```

SQL Query Structure

- Organized in Query blocks
  - `SELECT <select_list>
    FROM <from_list>
    WHERE <where_condition>
    GROUP BY <group_by_expressions>
    HAVING <having_condition>
    ORDER BY <order_by_expressions>`

Query Blocks

- Only `SELECT` clause is mandatory
  - Some DBMS require `FROM`

```sql
SELECT (1 + 2) AS result
```

SELECT clause

- List of expressions and optional name assignment + optional `DISTINCT`
  - Attribute references: `R.a`, `b`
  - Constants: 1, 'hello', '2008-01-20'
  - Operators: `(R.a + 3) * 2`
  - Functions (maybe UDF): `substr(R.a, 1,3)`
  - Single result or `set functions`
  - Renaming: `(R.a + 2) AS x`
**SELECT clause - example**

```
SELECT substring(p.name,1,1) AS initial 
p.name
FROM person p
```

<table>
<thead>
<tr>
<th>person</th>
<th>initial</th>
<th>name</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>J</td>
<td>male</td>
<td></td>
</tr>
<tr>
<td>Jim</td>
<td>J</td>
<td>male</td>
<td></td>
</tr>
</tbody>
</table>

**SELECT clause – set functions**

• Function extrChar(string)

```
SELECT extrChar(p.name) AS n
FROM person p
```

<table>
<thead>
<tr>
<th>person</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>J</td>
</tr>
<tr>
<td>Jim</td>
<td>i</td>
</tr>
</tbody>
</table>

**SELECT clause – DISTINCT**

```
SELECT DISTINCT gender
FROM person p
```

<table>
<thead>
<tr>
<th>person</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>male</td>
</tr>
<tr>
<td>Jim</td>
<td>male</td>
</tr>
</tbody>
</table>

**FROM clause**

• List of table expressions
  - Access to relations
  - Subqueries (need alias)
  - Join expressions
  - Table functions
  - Renaming of relations and columns

**FROM clause examples**

```
FROM R
  -access table R
FROM R, S
  -access tables R and S
FROM R JOIN S ON (R.a = S.b)
  -join tables R and S on condition (R.a = S.b)
FROM R x
FROM R AS x
  -Access table R and assign alias ‘x’
```

```
FROM R x(c,d)
FROM R AS x(c,d)
  -using aliases x for R and c,d for its attributes
FROM (R JOIN S t ON (R.a = t.b)), T
  -join R and S, and access T
FROM (R JOIN S ON (R.a = S.b)) JOIN T
  -join tables R and S and result with T
FROM create_sequence(1,100) AS seq(a)
  -call table function
```
FROM clause examples

FROM (SELECT count(*) FROM employee) AS empcnt(cnt)

- count number of employee in subquery

FROM clause examples

SELECT * FROM create_sequence(1,3) AS seq(a)

result

1
2
3

FROM clause examples

SELECT dep, headcnt
FROM (SELECT count(*) AS headcnt, dep
FROM employee
GROUP BY dep)
WHERE headcnt > 100

<table>
<thead>
<tr>
<th>employee</th>
<th>dep</th>
<th>headcnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>IT</td>
<td>103</td>
</tr>
<tr>
<td>Jim</td>
<td>Marketing</td>
<td>2506</td>
</tr>
</tbody>
</table>

result

FROM clause examples

SELECT name, chr
FROM employee AS e,
extrChar(e.name) AS c(chr)

<table>
<thead>
<tr>
<th>employee</th>
<th>name</th>
<th>chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>J</td>
<td>Joe</td>
</tr>
<tr>
<td>Jim</td>
<td>I</td>
<td>Jim</td>
</tr>
</tbody>
</table>

result

Correlation - Example

SELECT name, chr
FROM employee AS e,
extrChar(e.name) AS c(chr)

<table>
<thead>
<tr>
<th>employee</th>
<th>name</th>
<th>chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>J</td>
<td>Joe</td>
</tr>
<tr>
<td>Jim</td>
<td>I</td>
<td>Jim</td>
</tr>
</tbody>
</table>

result

Correlation - Example

SELECT name
FROM (SELECT max(salary) AS maxsal
FROM employee) AS m,
(SELECT name
FROM employee x
WHERE x.salary = m.maxsal) AS e

<table>
<thead>
<tr>
<th>employee</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>20,000</td>
</tr>
<tr>
<td>Jim</td>
<td>30,000</td>
</tr>
</tbody>
</table>

result

Correlation - Example

SELECT name
FROM (SELECT max(salary) AS maxsal
FROM employee) AS m,
(SELECT name
FROM employee x
WHERE x.salary = m.maxsal) AS e

<table>
<thead>
<tr>
<th>employee</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td></td>
</tr>
<tr>
<td>Jim</td>
<td></td>
</tr>
</tbody>
</table>
WHERE clause

- A condition
  - Attribute references
  - Constants
  - Operators (boolean)
  - Functions
  - Nested subquery expressions
- Result has to be boolean

WHERE clause examples

WHERE R.a = 3
- comparison between attribute and constant
WHERE (R.a > 5) AND (R.a < 10)
- range query using boolean AND
WHERE R.a = S.b
- comparison between two attributes
WHERE (R.a * 2) > (S.b - 3)
- using operators

Nested Subqueries

- Nesting a query within an expression
- Correlation allowed
  - Access FROM clause attributes
- Different types of nesting
  - Scalar subquery
  - Existential quantification
  - Universal quantification

Nested Subqueries Semantics

- For each tuple produced by the FROM clause execute the subquery
  - If correlated attributes replace them with tuple values

Scalar subquery

- Subquery that returns one result tuple
  - How to check?
    - -> Runtime error

```sql
SELECT *
FROM R
WHERE R.a = (SELECT count(*) FROM S)
```

Existential Quantification

- `<expr> IN <subquery>`
  - Evaluates to true if `<expr>` equal to at least one of the results of the subquery

```sql
SELECT *
FROM users
WHERE name IN (SELECT name FROM blacklist)
```
Existential Quantification

• \( \exists \) \(<\text{subquery}>\)
  – Evaluates to true if \(<\text{subquery}>\) returns at least one tuple

```
SELECT *
FROM users u
WHERE \exists \(<\text{select} \ast \text{from blacklist b WHERE b.name = u.name}>\n```

Universal Quantification

• \(<\text{expr}>\) \(\forall\) \(<\text{subquery}>\)
  – Evaluates to true if \(<\text{expr}>\) \(\forall\) \(<\text{tuple}>\)
evaluates to true for all result tuples
  – \(\forall\) is any comparison operator: \(=, <, >, \ldots\)

```
SELECT *
FROM nation
WHERE nname = ALL \(<\text{select} \text{nation FROM blacklist}>\n```

Nested Subqueries Example

```
SELECT dep, name
FROM employee e
WHERE salary >= ALL \(<\text{select} \text{salary FROM employee d WHERE e.dep = d.dep}>\n```

<table>
<thead>
<tr>
<th>name</th>
<th>dep</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>IT</td>
<td>2000</td>
</tr>
<tr>
<td>Jim</td>
<td>IT</td>
<td>300</td>
</tr>
<tr>
<td>Bob</td>
<td>HR</td>
<td>100</td>
</tr>
<tr>
<td>Alice</td>
<td>HR</td>
<td>10000</td>
</tr>
<tr>
<td>Patrice</td>
<td>HR</td>
<td>10000</td>
</tr>
</tbody>
</table>

GROUP BY clause

• A list of expressions
  – Same as WHERE
  – No restriction to boolean
  – DBMS has to know how to compare \(=\) for data type
• Results are grouped by values of the expressions
• \(\rightarrow\) usually used for aggregation

GROUP BY restrictions

• If group-by is used then
  – SELECT clause can only use group by expressions or aggregation functions
**GROUP BY clause examples**

- **GROUP BY R.a**
  - group on single attribute
- **GROUP BY (1+2)**
  - allowed but useless (single group)
- **GROUP BY salary / 1000**
  - groups of salary values in buckets of 1000
- **GROUP BY R.a, R.b**
  - group on two attributes

**HAVING clause**

- A boolean expression
- Applied after grouping and aggregation
  - Only references aggregation expressions and group by expressions

**ORDER BY clause**

- A list of expressions
- Semantics: Order the result on these expressions

**TABLE**

<table>
<thead>
<tr>
<th>numP</th>
<th>numF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**SELECT**

```
SELECT count(*) AS numP,
       (SELECT count(*)
        FROM friends o
        WHERE o.with = f.name) AS numF
FROM (SELECT DISTINCT name FROM friends) f
GROUP BY (SELECT count(*)
          FROM friends o
          WHERE o.with = f.name)
```

**HAVING clause examples**

- **HAVING sum(R.a) > 100**
  - only return tuples with sum bigger than 100

- **GROUP BY dep**
  - HAVING dep = 'IT' AND sum(salary) > 1000000
  - only return group 'IT' and sum threshold

**ORDER BY clause examples**

- **ORDER BY R.a ASC**
- order ascending on R.a
- **ORDER BY R.a DESC**
- order descending on R.a
- **ORDER BY salary + bonus**
- order by sum of salary and bonus
New and Non-standard SQL features (excerpt)

• LIMIT / OFFSET
  – Only return a fix maximum number of rows
  – FETCH FIRST n ROWS ONLY (DB2)
  – row_number() (Oracle)

• Window functions
  – More flexible grouping
  – Return both aggregated results and input values

Parsing, Analysis, Conversion

1. Parsing
2. Analysis
3. Conversion

Analysis Goals

• Semantic checks
  – Table column exists
  – Operator, function exists
  – Determine type casts
  – Scope checks

• Rewriting
  – Unfolding views

Semantic checks

SELECT *
FROM R
WHERE R.a + 3 > 5
• Table R exists?
• Expand *=: which attributes in R?
• R.a is a column?
• Type of constants 3, 5?
• Operator + for types of R.a and 3 exists?
• Operator > for types of result of + and 5 exists?

Database Catalog

• Stores information about database objects

• Aliases:
  – Information Schema
  – System tables
  – Data Dictionary

Typical Catalog Information

• Tables
  – Name, attributes + data types, constraints

• Schema, DB
  – Hierarchical structuring of data

• Data types
  – Comparison operators
  – physical representation
  – Functions to (de)serialize to string
Typical Catalog Information

- Functions (including aggregate/set)
  - Build-in
  - User defined (UDF)
- Triggers
- Stored Procedures
- ...

Type Casts

- Similar to automatic type conversion in programming languages
- Expression: R.a + 3.0
  - Say R.a is of type integer
  - Search for a function +(int, float)
  - Does not exist?
    - Try to find a way to cast R.a, 3.0 or both to new data type
    - So that a function + exists for new types

Scope checks

- Check that references are in correct scope
- E.g., if GROUP BY is present then SELECT clause expression can only reference group by expressions or aggregated values

View Unfolding

- SQL allows for stored queries using CREATE VIEW
- Afterwards a view can be used in queries
- If view is not materialized, then need to replace view with its definition

View Unfolding Example

CREATE VIEW totalSalary AS
SELECT name, salary + bonus AS total
FROM employee
SELECT *
FROM totalSalary
WHERE total > 10000
Analysis Summary

- Perform semantic checks
  - Catalog lookups (tables, functions, types)
  - Scope checks
- View unfolding
- Generate internal representation during analysis

Conversion

- Create an internal representation
  - Should be useful for analysis
  - Should be useful optimization
- Internal representation
  - Relational algebra
  - Query tree/graph models
    - E.g., QGM (Query Graph Model) in Starburst

Relational Algebra

- Formal language
- Good for studying logical optimization and query equivalence (containment)
- Not informative enough for analysis
  - No datatype representation in algebra expressions
  - No meta-data

Other Internal Representations

- Practical implementations
  - Mostly following structure of SQL query blocks
  - Store data type and meta-data (where necessary)

Canonical Translation to Relational Algebra

- TEXTBOOK version of conversion
- Given an SQL query
- Return an equivalent relational algebra expression
Relational Algebra Recap

- Formal query language
- Consists of operators
  - Input(s): relation
  - Output: relation
  - \( \rightarrow \) Composable
- Set and Bag semantics version

Relation Schema
- A set of attribute name-datatype pairs

Relation (instance)
- A (multi-)set of tuples with the same schema

Tuple
- List of attribute value pairs (or function from attribute name to value)

Set- vs. Bag semantics

- Set semantics:
  - Relations are Sets
  - Used in most theoretical work

- Bag semantics
  - Relations are Multi-Sets
  - Each element (tuple) can appear more than once
  - SQL uses bag semantics

Operators

- Selection
- Renaming
- Projection
- Joins
  - Theta, natural, cross-product, outer, anti
- Aggregation
- Duplicate removal
- Set operations

Bag semantics notation

- We use \( t^m \) to denote tuple \( t \) appears with multiplicity \( m \)

Set- vs. Bag semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>Guitar</td>
</tr>
<tr>
<td>Joe</td>
<td>Drum</td>
</tr>
<tr>
<td>Alice</td>
<td>Bass</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>Guitar</td>
</tr>
<tr>
<td>Alice</td>
<td>Bass</td>
</tr>
<tr>
<td>Alice</td>
<td>Bass</td>
</tr>
</tbody>
</table>
### Selection

- **Syntax:** \( \sigma_c(R) \)
  - \( R \) is input
  - \( C \) is a condition

- **Semantics:**
  - Return all tuples that match condition \( C \)
  - Set: \( \{ t | t \in R \text{ AND } t \text{ fulfills } C \} \)
  - Bag: \( \{ t^n | t^n \in R \text{ AND } t \text{ fulfills } C \} \)

### Selection Example

\[ \sigma_{a>5}(R) \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
</tr>
</tbody>
</table>

### Renaming

- **Syntax:** \( \rho_A(R) \)
  - \( R \) is input
  - \( A \) is list of attribute renamings

- **Semantics:**
  - Applies renaming from \( A \) to inputs
  - Set: \( \{ t.A | t \in R \} \)
  - Bag: \( \{ (t.A)^n | t^n \in R \} \)

### Renaming Example

\[ \rho_{c \leftarrow a}(R) \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
</tr>
</tbody>
</table>

### Projection

- **Syntax:** \( \Pi_A(R) \)
  - \( R \) is input
  - \( A \) is list of projection expressions
  - Standard: only attributes in \( A \)

- **Semantics:**
  - Project all inputs on projection expressions
  - Set: \( \{ t.A | t \in R \} \)
  - Bag: \( \{ (t.A)^n | t^n \in R \} \)

### Projection Example

\[ \Pi_b(R) \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
</tr>
</tbody>
</table>
Cross Product

- Syntax: \( R \times S \)
  - \( R \) and \( S \) are inputs
- Semantics:
  - All combinations of tuples from \( R \) and \( S \)
  - Set: \( \{ (t,s) \mid t \in R \text{ AND } s \in S \} \)
  - Bag: \( \{ (t,s)^{n\times m} \mid t \in R \text{ AND } s \in S \} \)

Cross Product Example

- \( R \times S \)
- \( R \)
- \( S \)
- Result

Join

- Syntax: \( R \bowtie_{C} S \)
  - \( R \) and \( S \) are inputs
  - \( C \) is a condition
- Semantics:
  - All combinations of tuples from \( R \) and \( S \) that match \( C \)
  - Set: \( \{ (t,s) \mid t \in R \text{ AND } s \in S \text{ AND } (t,s) \text{ matches } C \} \)
  - Bag: \( \{ (t,s)^{n\times m} \mid t \in R \text{ AND } s \in S \text{ AND } (t,s) \text{ matches } C \} \)

Join Example

- \( R \bowtie_{a=d} S \)
- \( R \)
- \( S \)
- Result

Natural Join

- Syntax: \( R \bowtie S \)
  - \( R \) and \( S \) are inputs
- Semantics:
  - All combinations of tuples from \( R \) and \( S \) that match on common attributes
  - \( A = \) common attributes of \( R \) and \( S \)
  - \( C = \) exclusive attributes of \( S \)
  - Set: \( \{ (t,s,C) \mid t \in R \text{ AND } s \in S \text{ AND } t.A = s.A \} \)
  - Bag: \( \{ (t,s,C)^{n\times m} \mid t \in R \text{ AND } s \in S \text{ AND } t.A = s.A \} \)

Natural Join Example

- \( R \bowtie S \)
- \( R \)
- \( S \)
- Result
Left-outer Join

- Syntax: $R \bowtie_C S$
  - $R$ and $S$ are inputs
  - $C$ is condition
- Semantics:
  - $R$ join $S$
  - $t \in R$ without match, fill $S$ attributes with NULL
    - $\{ (t, s) | t \in R \text{ AND } s \in S \text{ AND } (t, s) \text{ matches } C \}$
  - union
    - $\{ (t, \text{NULL}(s)) | t \in R \text{ AND NOT exists } s \in S : (t, s) \text{ matches } C \}$

Left-outer Join Example

- $R \bowtie_{a=d} S$

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>b</td>
</tr>
</tbody>
</table>

Right-outer Join

- Syntax: $R \bowtie_C S$
  - $R$ and $S$ are inputs
  - $C$ is condition
- Semantics:
  - $R$ join $S$
  - $s \in S$ without match, fill $R$ attributes with NULL
    - $\{ (t, s) | t \in R \text{ AND } s \in S \text{ AND } (t, s) \text{ matches } C \}$
  - union
    - $\{ (\text{NULL}(R), s) | s \in S \text{ AND NOT exists } t \in R : (t, s) \text{ matches } C \}$

Right-outer Join Example

- $R \bowtie_{a=d} S$

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>b</td>
</tr>
</tbody>
</table>

Full-outer Join

- Syntax: $R \bowtie_C S$
  - $R$ and $S$ are inputs
  - $C$ is condition
- Semantics:
  - $\{ (t, s) | t \in R \text{ AND } s \in S \text{ AND } (t, s) \text{ matches } C \}$
  - union
    - $\{ (\text{NULL}(R), s) | s \in S \text{ AND NOT exists } t \in R : (t, s) \text{ matches } C \}$
  - union
    - $\{ (t, \text{NULL}(s)) | t \in R \text{ AND NOT exists } s \in S : (t, s) \text{ matches } C \}$

Full-outer Join Example

- $R \bowtie_{a=d} S$

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>b</td>
</tr>
</tbody>
</table>
Semijoin

- Syntax: $R \bowtie S$ and $R \bowtie S$
- $R$ and $S$ are inputs
- Semantics:
  - All tuples from $R$ that have a matching tuple from relation $S$ on the common attributes $A$
    \[ \{ t \mid t \in R \text{ AND exists } s \in S : t.A = s.A \} \]

Semijoin Example

- $R \bowtie S$

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>13</td>
<td>b 5</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>b 3</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Antijoin

- Syntax: $R \triangleright S$
- $R$ and $S$ are inputs
- Semantics:
  - All tuples from $R$ that have no matching tuple from relation $S$ on the common attributes $A$
    \[ \{ t \mid t \in R \text{ AND NOT exists } s \in S : t.A = s.A \} \]

Antijoin Example

- $R \triangleright S$

<table>
<thead>
<tr>
<th>$R$</th>
<th>$S$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>13</td>
<td>b 5</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>b 3</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Aggregation

- Syntax: $\alpha A (R)$
- $A$ is list of aggregation functions
- $G$ is list of group by attributes
- Semantics:
  - Build groups of tuples according $G$ and compute the aggregation functions from each group
  - $\{ (t' G, \text{agg}(G(t)) \mid t \in R \}$
  - $G(t) = \{ t'' \mid t'' \in R \text{ AND } t''.G = t.G \}$

Aggregation Example

- $\sum (\text{a sum(a)} (R))$

<table>
<thead>
<tr>
<th>$R$</th>
<th>$\text{sum(a)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Duplicate Removal

- Syntax: \( \delta(R) \)
  - \( R \) is input

- Semantics:
  - Remove duplicates from input
  - Set: N/A
  - Bag: \( \{ t^i | t^i \in R \} \)

Duplicate Removal Example

\[ \delta (R) \]

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

Set operations

- Input: \( R \) and \( S \)
  - Have to have the same schema
    - Union compatible
  - Modulo attribute names

- Types
  - Union
  - Intersection
  - Set difference

Union

- Syntax: \( R \cup S \)
  - \( R \) and \( S \) are union-compatible inputs

- Semantics:
  - Set: \( \{ (t) | t \in R \lor t \in S \} \)
  - Bag: \( \{ (t,s)^{\min(n,m)} | t \in R \land s \in S \} \)
    - Assumption \( t^n \) with \( n < 1 \) for tuple not in relation

Union Example

\[ R \cup S \]

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Intersection

- Syntax: \( R \cap S \)
  - \( R \) and \( S \) are union-compatible inputs

- Semantics:
  - Set: \( \{ (t) | t \in R \land t \in S \} \)
  - Bag: \( \{ (t,s)^{\min(n,m)} | t \in R \land s \in S \} \)
Intersection Example

\[ R \cap S \]

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>c</td>
</tr>
</tbody>
</table>

Set Difference

- Syntax: \( R - S \)
- \( R \) and \( S \) are union-compatible inputs
- Semantics:
  - Set: \( \{ t \mid t \in R \text{ AND NOT } t \in S \} \)
  - Bag: \( \{ (t,s) \mid n - m \mid t \in R \text{ AND } s \in S \} \)

Set Difference Example

\[ R - S \]

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>c</td>
</tr>
</tbody>
</table>

Canonical Translation to Relational Algebra

- **TEXTBOOK** version of conversion
- Given an SQL query
- Return an equivalent relational algebra expression

Canonical Translation

- **FROM** clause into joins and cross-products
  - Cross-product between list items
  - Joins into their algebra counter-part
- **WHERE** clause into selection
- **SELECT** clause into projection and renaming
  - If it has aggregation functions use aggregation
  - **DISTINCT** into duplicate removal
- **GROUP BY** clause into aggregation
- **HAVING** clause into selection
- **ORDER BY** – no counter-part
- Then turn joins into crossproducts and selections
Set Operations

- **UNION ALL** into union
- **UNION** duplicate removal over union
- **INTERSECT ALL** into intersection
- **INTERSECT** add duplicate removal
- **EXCEPT ALL** into set difference
- **EXCEPT** apply duplicate removal to inputs and then apply set difference

Example: Relational Algebra Translation

SELECT sum(a) 
FROM R 
GROUP BY b

Example: Relational Algebra Translation

```
SELECT dep, headcnt
FROM (SELECT count(*) AS headcnt, dep
      FROM employee
      GROUP BY dep)
WHERE headcnt > 100
```

Example: Relational Algebra Translation

SELECT * 
FROM R JOIN S ON (R.a = S.b)

Parsing and Analysis Summary

- SQL text -> Internal representation
- Semantic checks
- Database catalog
- View unfolding