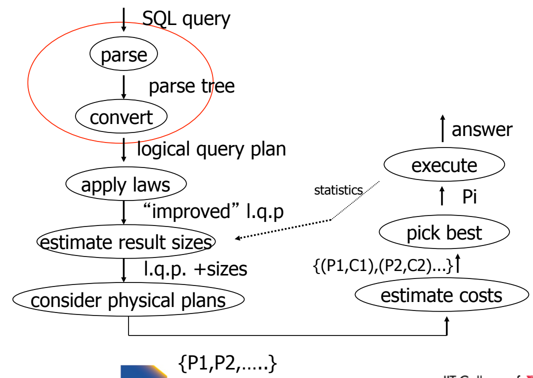
# CS 525: Advanced Database Organisation

## 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, Analysis, Conversion

1. Parsing
2. Analysis
3. Conversion



## 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



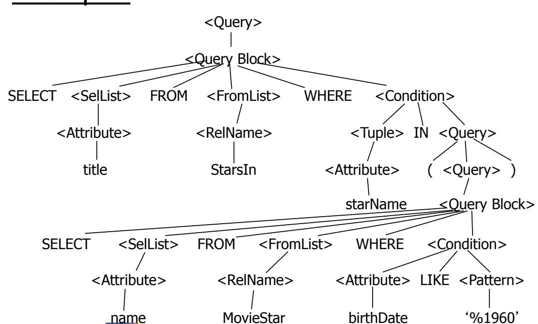
## Example: SQL query

```
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






# 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**

**SELECT** (1 + 2) AS result

| result |
| --- |
| 3 |



# **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
```

**person**

| name | gender |
|---|---|
| Joe | male |
| Jim | male |

**result**

| initial | name |
|---|---|
| J | Joe |
| J | Jim |



# SELECT clause – set functions

- Function `extrChar(string)`

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

**person**

| name | gender |
|---|---|
| Joe | male |
| Jim | male |

**result**

| n |
|---|
| J |
| o |
| e |
| J |
| i |
| m |



# SELECT clause – DISTINCT

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

**person**

| name | gender |
|---|---|
| Joe | male |
| Jim | male |

**result**

| gender |
|---|
| male |



# 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 clause examples

`FROM R x(c,d)`

`FROM R AS x(c,d)`

-using aliases x for R and c,d for its attribues

`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**

| a |
|---|
| 1 |
| 2 |
| 3 |



## **FROM** clause examples

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

**employee**

| name | dep |
|---|---|
| Joe | IT |
| Jim | Marketing |
| … | … |

**result**

| dep | headcnt |
|---|---|
| IT | 103 |
| Support | 2506 |
| … | … |



## **FROM** clause - correlation

- Correlation
  - Reference attributes from other FROM clause item
  - Attributes of $i^{th}$ entry only available in j > I
  - Semantics:
    - For each row in result of $i^{th}$ entry:
    - Substitute correlated attributes with value from current row and evaluate query



## Correlation - Example

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

**employee**

| name | dep |
|---|---|
| Joe | IT |
| Jim | Marketing |
| … | … |

**result**

| name | chr |
|---|---|
| Joe | J |
| Joe | o |
| Joe | e |
| Jim | J |
| Jim | i |
| … | … |



## Correlation - Example

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

**employee**

| name | salary |
|---|---|
| Joe | 20,000 |
| Jim | 30,000 |
| … | … |

**result**

| name |
|---|
| Jim |

# **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

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



# Existential Quantification

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

```
SELECT *
FROM users
WHERE name IN (SELECT name FROM
              blacklist)
```

## Existential Quantification

- `EXISTS <subquery>`
  - Evaluates to true if <subquery> returns at least one tuple

```
SELECT *
FROM users u
WHERE EXISTS (SELECT * FROM
              blacklist
             WHERE b.name = u.name)
```



## Existential Quantification

- `<expr> <op> ANY <subquery>`
  - Evaluates to true if <expr> <op> <tuple> evaluates to true for **at least one** result tuple
  - Op is any comparison operator: =, <, >, …

```
SELECT *
FROM users
WHERE name = ANY (SELECT name FROM
                 blacklist)
```



## Universal Quantification

- `<expr> <op> ALL <subquery>`
  - Evaluates to true if <expr> <op> <tuple> evaluates to true for **all** result tuples
  - Op is any comparison operator: =, <, >, …

```
SELECT *
FROM nation
WHERE nname = ALL (SELECT nation FROM
                  blacklist)
```



## Nested Subqueries Example

```
SELECT dep,name
FROM employee e
WHERE salary >= ALL (SELECT salary
                    FROM employee d
                    WHERE e.dep = d.dep)
```

**employee**

| name | dep | salary |
|---|---|---|
| Joe | IT | 2000 |
| Jim | IT | 300 |
| Bob | HR | 100 |
| Alice | HR | 10000 |
| Patrice | HR | 10000 |

**result**

| dep | Name |
|---|---|
| IT | Joe |
| HR | Alice |
| HR | Patrice |



## **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
- -> 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



```
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)
```

**result**

| numP | numF |
|---|---|
| 1 | 1 |
| 2 | 2 |

**friends**

| name | with |
|---|---|
| Joe | Jim |
| Joe | Peter |
| Jim | Joe |
| Jim | Peter |
| Peter | Joe |



# HAVING clause

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



# 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

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



# ORDER BY clause examples

```
ORDER BY R.a ASC
ORDER BY R.a
```
-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 than 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
```



## View Unfolding Example

```
CREATE VIEW totalSalary AS
SELECT name, salary + bonus AS total
FROM employee

SELECT *
FROM (SELECT name,
             salary + bonus AS total
      FROM employee) AS totalSalary
WHERE total > 10000
```

# Analysis Summary

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



## Parsing, Analysis, Conversion

1. Parsing
2. Analysis
3. Conversion



# 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 Alegbra

- 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
  - -> 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



## Bag semantics notation

- We use $\mathbf{t}^{\mathbf{m}}$ to denote tuple t appears with multiplicity **m**



## Set- vs. Bag semantics

Set

| Name | Purchase |
|---|---|
| Peter | Guitar |
| Joe | Drum |
| Alice | Bass |

Bag

| Name | Purchase |
|---|---|
| Peter | Guitar |
| Peter | Guitar |
| Joe | Drum |
| Alice | Bass |
| Alice | Bass |



## Operators

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

## Selection

– Syntax: $\sigma_c$ (R)
  - R is input
  - C is a condition
– Semantics:
  - Return all tuples that match condition C
  - Set: { t | t ε R AND t fulfills C }
  - Bag: { $t^n$ | $t^n$ ε R AND t fulfills C }



## Selection Example

- $\sigma_{a<5}$ (R)

R

| a | b |
|---|---|
| 1 | 13 |
| 3 | 12 |
| 6 | 14 |

Result

| a | b |
|---|---|
| 6 | 14 |



## Renaming

– Syntax: $\rho_A$ (R)
  - R is input
  - A is list of attribute renamings b ← a
– Semantics:
  - Applies renaming from A to inputs
  - Set: { t.A | t ε R }
  - Bag: { $(t.A)^n$ | $t^n$ ε R }



## Renaming Example

- $\rho_{c \leftarrow a}$ (R)

R

| a | b |
|---|---|
| 1 | 13 |
| 3 | 12 |
| 6 | 14 |

Result

| c | b |
|---|---|
| 1 | 13 |
| 3 | 12 |
| 6 | 14 |



## Projection

– Syntax: $\Pi_A$ (R)
  - R is input
  - A is list of projection expressions
  - Standard: only attribute in A
– Semantics:
  - Project all inputs on projection expressions
  - Set: { t.A | t ε R }
  - Bag: { $(t.A)^n$ | $t^n$ ε R }



## Projection Example

- $\Pi_b$ (R)

R

| a | b |
|---|---|
| 1 | 13 |
| 3 | 12 |
| 6 | 14 |

Result

| b |
|---|
| 13 |
| 12 |
| 14 |

# Cross Product

– Syntax: R X S
  - R and S are inputs
– Semantics:
  - All combinations of tuples from R and S
  - = mathematical definition of cross product
  - Set: { (t,s) | t ε R AND s ε S }
  - Bag: { $(t,s)^{n*m}$ | $t^n$ ε R AND $s^m$ ε S }



# Cross Product Example

- R X S

R

| a | b |
|---|---|
| 1 | 13 |
| 3 | 12 |

S

| c | d |
|---|---|
| a | 5 |
| b | 3 |
| c | 4 |

Result

| a | b | c | d |
|---|---|---|---|
| 1 | 13 | a | 5 |
| 1 | 13 | b | 3 |
| 1 | 13 | c | 4 |
| 3 | 12 | a | 5 |
| 3 | 12 | b | 3 |
| 3 | 12 | c | 4 |



# 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) | t ε R AND s ε S AND (t,s) matches C}
  - Bag: { $(t,s)^{n*m}$ | $t^n$ ε R AND $s^m$ ε S AND (t,s) matches C}



# Join Example

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

R

| a | b |
|---|---|
| 1 | 13 |
| 3 | 12 |

S

| c | d |
|---|---|
| a | 5 |
| b | 3 |
| c | 4 |

Result

| a | b | c | d |
|---|---|---|---|
| 3 | 12 | b | 3 |



# 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) | t ε R AND s ε S AND t.A=s.A}
  - Bag: { $(t,s.C)^{n*m}$ | $t^n$ ε R AND $s^m$ ε S AND t.A=s.A}



# Natural Join Example

- R $\bowtie$ S

R

| a | b |
|---|---|
| 1 | 13 |
| 3 | 12 |

S

| c | a |
|---|---|
| a | 5 |
| b | 3 |
| c | 4 |

Result

| a | b | c |
|---|---|---|
| 3 | 12 | b |

# Left-outer Join

- Syntax: $R ⟕_{C} S$
  - R and S are inputs
  - C is condition
- Semantics:
  - R join S
  - t ε R without matches fill S attributes with NULL

{ (t,s) | t ε R AND s ε S AND (t,s) matches C}
union
{ (t, NULL(S)) | t ε R AND NOT exists s ε S: (t,s) matches C }



# Left-outer Join Example

- $R ⟕_{a=d} S$

R

| a | b |
|---|---|
| 1 | 13 |
| 3 | 12 |

S

| c | d |
|---|---|
| a | 5 |
| b | 3 |
| c | 4 |

Result

| a | b | c | d |
|---|---|---|---|
| 1 | 13 | NULL | NULL |
| 3 | 12 | b | 3 |



# Right-outer Join

- Syntax: $R ⟖_{C} S$
  - R and S are inputs
  - C is condition
- Semantics:
  - R join S
  - s ε S without matches fill R attributes with NULL

{ (t,s) | t ε R AND s ε S AND (t,s) matches C}
union
{ (NULL(R),s) | s ε S AND NOT exists t ε R: (t,s) matches C }



# Right-outer Join Example

- $R ⟖_{a=d} S$

R

| a | b |
|---|---|
| 1 | 13 |
| 3 | 12 |

S

| c | d |
|---|---|
| a | 5 |
| b | 3 |
| c | 4 |

Result

| a | b | c | d |
|---|---|---|---|
| NULL | NULL | a | 5 |
| 3 | 12 | b | 3 |
| NULL | NULL | c | 4 |



# Full-outer Join

- Syntax: $R ⟗_{C} S$
  - R and S are inputs and C is condition
- Semantics:

{ (t,s) | t ε R AND s ε S AND (t,s) matches C}
union
{ (NULL(R),s) | s ε S AND NOT exists t ε R: (t,s) matches C }
union
{ (t, NULL(S)) | t ε R AND NOT exists s ε S: (t,s) matches C }



# Full-outer Join Example

- $R ⟗_{a=d} S$

R

| a | b |
|---|---|
| 1 | 13 |
| 3 | 12 |

S

| c | d |
|---|---|
| a | 5 |
| b | 3 |
| c | 4 |

Result

| a | b | c | d |
|---|---|---|---|
| 1 | 13 | NULL | NULL |
| NULL | NULL | a | 5 |
| 3 | 12 | b | 3 |
| NULL | NULL | c | 4 |

# Semijoin

– Syntax: R ⋉ S and R ⋊ 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 | t ε R AND exists s ε S: t.A = s.A}



# Semijoin Example

- R ⋉ S

R

| a | b |
|---|---|
| 1 | 13 |
| 3 | 12 |

S

| c | a |
|---|---|
| a | 5 |
| b | 3 |
| c | 4 |

Result

| a | b |
|---|---|
| 3 | 12 |



# Antijoin

– Syntax: R ▷ 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 | t ε R AND NOT exists s ε S: t.A = s.A}



# Antijoin Example

- R ▷ S

R

| a | b |
|---|---|
| 1 | 13 |
| 3 | 12 |

S

| c | a |
|---|---|
| a | 5 |
| b | 3 |
| c | 4 |

Result

| a | b |
|---|---|
| 1 | 13 |



# Aggregation

– Syntax: $_{A}\alpha_{G}$ (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, agg(G(t)) | t ε R }
  - G(t) = { t' | t' ε R AND t'.G = t.G }



# Aggregation Example

- $_{b}\alpha_{sum(a)}$ (R)

R

| a | b |
|---|---|
| 1 | 1 |
| 3 | 1 |
| 6 | 2 |
| 3 | 2 |

Result

| sum(a) | b |
|---|---|
| 4 | 1 |
| 9 | 2 |

## Duplicate Removal

– Syntax: δ (R)
  - R is input
– Semantics:
  - Remove duplicates from input
  - Set: N/A
  - Bag: { $t^1$ | $t^n$ ε R }



## Duplicate Removal Example

- δ (R)

R

| a | b |
|---|---|
| 1 | 13 |
| 1 | 13 |
| 6 | 14 |

Result

| a | b |
|---|---|
| 1 | 13 |
| 6 | 14 |



## 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 ∪ S
  - R and S are union-compatible inputs
– Semantics:
  - Set: { (t) | t ε R OR t ε S}
  - Bag: { $(t,s)^{n+m}$ | $t^n$ ε R AND $s^m$ ε S }
    – Assumption $t^n$ with n < 1 for tuple not in relation



## Union Example

- R ∪ S

R

| a |
|---|
| 1 |
| 3 |

S

| b |
|---|
| 1 |
| 2 |
| 3 |

Result

| a |
|---|
| 1 |
| 2 |
| 3 |
| 1 |
| 3 |



## Intersection

– Syntax: R ∩ S
  - R and S are union-compatible inputs
– Semantics:
  - Set: { (t) | t ε R AND t ε S}
  - Bag: { $(t,s)^{min(n,m)}$ | $t^n$ ε R AND $s^m$ ε S }

## Intersection Example

- R ∩ S

R

| a |
|---|
| 1 |
| 3 |

S

| b |
|---|
| 1 |
| 2 |
| 3 |

Result

| a |
|---|
| 1 |
| 3 |



## Set Difference

- Syntax: R - S
  - R and S are union-compatible inputs
- Semantics:
  - Set: { (t) | t ε R AND NOT t ε S}
  - Bag: { $(t,s)^{n-m}$ | $t^n$ ε R AND $s^m$ ε S }



## Set Difference Example

- R - S

R

| a |
|---|
| 1 |
| 5 |

S

| b |
|---|
| 1 |
| 2 |
| 3 |

Result

| a |
|---|
| 5 |



## 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



## Canonical Translation

- **GROUP BY** clause into aggregation
- **HAVING** clause into selection
- **ORDER BY** – no counter-part

- Then turn joins into crossproducts and selections

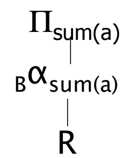
## Set Operations

- **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

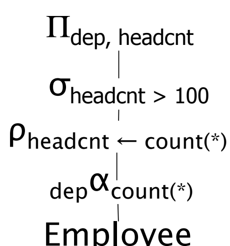
```
SELECT sum(R.a)
FROM R
GROUP BY b
```






## Example: Relational Algebra Translation

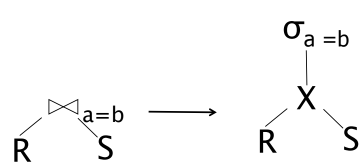
```
SELECT dep, headcnt
FROM (SELECT count(*) AS headcnt, dep
     FROM employee
     GROUP BY dep)
WHERE headcnt > 100
```






## 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