Extended RA

*Database Systems: The Complete Book*

Ch 5.1-5.2, 15.4
Relational Algebra

Data

A Set of Tuples

[Set] Relational Algebra

A Bag of Tuples

Bag Relational Algebra

A List of Tuples

Extended Relational Algebra
What’s Missing?

**Set Relational Algebra**
Select (σ), Project (π), Join (∝), Union (∪)

**Bag-Relational Algebra**
Distinct (δ), Outer Joins

**List-Relational Algebra**
Sort (τ), Limit

**Arithmetic Expressions**
Extended Projection (π), Aggregation (Σ), Grouping (ɣ)
What’s Missing?

Set Relational Algebra
Select (σ), Project (π), Join (⋈), Union (∪)

Bag-Relational Algebra
Distinct (δ), Outer Joins

List-Relational Algebra
Sort (τ), Limit

Arithmetic Expressions
Extended Projection (π), Aggregation (Σ), Grouping (ɣ)
Extended Projection

**Originally**: A List of Attributes

**Now**: A List of (Name,Expression) Pairs

\[ \pi_{Total: \text{Price} \times (1-\text{Discount}), \text{Profit: Cost} - \text{Price} \times (1-\text{Discount})} \text{Lineitem} \]
Sort, Limit

Sort a List

Pick the first N items from a List
Sort, Limit

Sort a List

Pick the first N items from a List
Sort, Limit

Sort a List
Pick the first N items from a List

What happens if you use Limit without Sort?
Sort

How do you implement Sort?
Sort

How do you implement Sort?

Can you do all of the work in GetNext()?
void open() {
    child.open()
    buffer = new List<Tuple>()
    while((next = child.getNext()) != null) {
        buffer.add(next)
    }
    Collections.sort(buffer)
}
void open() {
  child.open()
  buffer = new List<Tuple>()
  while((next = child.getNext()) != null)
    buffer.add(next)
  Collections.sort(buffer)
}

What are the potential problems of this approach?
Aggregation

COUNT(*)
COUNT(DISTINCT A[, B[, ...]])
SUM([DISTINCT] A)
AVG([DISTINCT] A)
MAX(A)
MIN(A)
Aggregation

How do we implement these?
void Init() {
    // prepare the aggregate
}

void Consume(float value) {
    // “add” value to the aggregate
}

float Finalize() {
    // return the final aggregate value
}
Iterators

Aggregate

Read One Tuple

Empty? Add to Aggregate

Finalize()

Emit Tuple
Iterators

Aggregate

Read One Tuple

Empty?  Add to Aggregate

Finalize()

Emmit Tuple

What is the Working Set Size?
Group Work

Design folds for any two of these aggregates

COUNT(*)
SUM(A)
AVG(A)
MAX(A)
Group Work

Design folds for each of these aggregates

\[
\text{COUNT} (\text{DISTINCT A})
\]
Group Work

Design folds for each of these aggregates

\[ \text{COUNT(DISTINCT A)} \]

What is the Working Set Size?
Grouping

∀A, B, …, Ccnt: COUNT(C), Dsum: SUM(D), …

For every unique value of <A, B, …>
Compute the Count of all Cs in <A, B, …, C, D,…>
Compute the SUM of all Ds in <A, B, …, C, D…>
Grouping

∀A, B, …, Ccnt: COUNT(C), Dsum: SUM(D), …

For every unique value of <A, B, …>
Compute the Count of all Cs in <A, B, …, C, D,…>
Compute the SUM of all Ds in <A, B, …, C, D,…>

What is the Output Schema?
Iterators

Group By Aggregate

Read One Tuple

Empty?
  \[\text{Finalize()}\] one group
  Emit Tuple

Find Group
  Add to Aggregate
Iterators

Group By Aggregate

Read One Tuple

Empty?

Finalize() one group

Emit Tuple

Find Group

Add to Aggregate

What is the Working Set Size?
Iterators

**Group By Aggregate**

- Read One Tuple
  - Empty?
    - Finalize() one group
      - Emit Tuple
  - Find Group
    - Add to Aggregate

What Data-Structures are required?
What is the Working Set Size?
Group Work

Use the Grouping operator to implement Distinct
**NULL Values**

- Field values can be *unknown* or *inapplicable*.
  - An officer not assigned to a ship.
  - Someone with no last name.
    - ‘Spock’ or ‘Data’ or ‘.’
- SQL provides a special **NULL** value for this.
- **NULL** makes things more complicated.
NULL Values

\[ \text{O.Rank} > 3.0 \]

What happens if O.Rank is null?
NULL Values

O.Rank > 3.0

What happens if O.Rank is NULL?

Predicates can be True, False, or Unknown (3-valued logic)

WHERE clause eliminates all Non-True values
NULL Values

O.Rank > 3.0

What happens if O.Rank is NULL?

Predicates can be True, False, or Unknown (3-valued logic)

WHERE clause eliminates all Non-True values

How does this interact with AND, OR, NOT?
NULL Values

Unknown AND True = Unknown
Unknown AND False = False

Unknown OR True = True
Unknown OR False = Unknown

NOT Unknown = Unknown
Outer Joins

<table>
<thead>
<tr>
<th>ID,</th>
<th>Name</th>
<th>Ship,</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1701,</td>
<td>Enterprise</td>
<td>[1701,</td>
<td>Subspace Anomaly</td>
</tr>
<tr>
<td>DS9,</td>
<td>Deep Space 9</td>
<td>DS9,</td>
<td>Bajor</td>
</tr>
<tr>
<td>74656,</td>
<td>Voyager</td>
<td>74656,</td>
<td>Gamma Quadrant</td>
</tr>
<tr>
<td>75633,</td>
<td>Defiant</td>
<td>75633,</td>
<td>Risa</td>
</tr>
<tr>
<td>ID</td>
<td>Name</td>
<td>Ship</td>
<td>Location</td>
</tr>
<tr>
<td>--------</td>
<td>---------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>1701</td>
<td>Enterprise</td>
<td>DS9</td>
<td>Bajor</td>
</tr>
<tr>
<td>74656</td>
<td></td>
<td>Voyager</td>
<td>Gamma Quadrant</td>
</tr>
<tr>
<td>75633</td>
<td></td>
<td>Defiant</td>
<td>Risa</td>
</tr>
</tbody>
</table>

**Outer Joins**
Outer Joins

<table>
<thead>
<tr>
<th>Ships</th>
<th>Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID, Name</td>
<td>Ship, Location</td>
</tr>
<tr>
<td>[1701, Enterprise ]</td>
<td>[DS9, Bajor ]</td>
</tr>
<tr>
<td>[DS9, Deep Space 9]</td>
<td>[74656, Gamma Quadrant ]</td>
</tr>
<tr>
<td>[74656, Voyager ]</td>
<td>[75633, Risa ]</td>
</tr>
<tr>
<td>[75633, Defiant ]</td>
<td></td>
</tr>
</tbody>
</table>
Outer Joins

<table>
<thead>
<tr>
<th>ID,</th>
<th>Name</th>
<th>Ship,</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1701,</td>
<td>Enterprise</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS9,</td>
<td>Deep Space 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74656,</td>
<td>Voyager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75633,</td>
<td>Defiant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[DS9, Bajor]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[74656, Gamma Quadrant]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[75633, Risa]</td>
</tr>
</tbody>
</table>

What is the result of this query?

\[ \pi_{\text{Location}} \sigma_{\text{Name}='\text{Enterprise}'} (\text{Ships} \bowtie_{\text{Ship}=\text{ID}} \text{Locations}) \]
Outer Joins

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Ship</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1701, Enterprise ]</td>
<td>[DS9, Bajor ]</td>
<td>[74656, Gamma Quadrant ]</td>
<td></td>
</tr>
<tr>
<td>[DS9, Deep Space 9 ]</td>
<td></td>
<td>[75633, Risa ]</td>
<td></td>
</tr>
<tr>
<td>[74656, Voyager ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[75633, Defiant ]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What is the result of this query?

\[ \pi_{\text{Location}} \sigma_{\text{Name}=\text{Enterprise}} (\text{Ships} \bowtie_{\text{Ship}=\text{ID}} \text{Locations}) \]

Is an empty result what we’re looking for?
# Outer Joins

## Ships

<table>
<thead>
<tr>
<th>ID,</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1701,</td>
<td>Enterprise</td>
</tr>
<tr>
<td>DS9,</td>
<td>Deep Space 9</td>
</tr>
<tr>
<td>74656,</td>
<td>Voyager</td>
</tr>
<tr>
<td>75633,</td>
<td>Defiant</td>
</tr>
</tbody>
</table>

## Locations

<table>
<thead>
<tr>
<th>Ship,</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS9,</td>
<td>Bajor</td>
</tr>
<tr>
<td>74656,</td>
<td>Gamma Quadrant</td>
</tr>
<tr>
<td>75633,</td>
<td>Risa</td>
</tr>
</tbody>
</table>

## Inner Joins

<table>
<thead>
<tr>
<th>ID,</th>
<th>Name,</th>
<th>Ship,</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1701,</td>
<td>Enterprise,</td>
<td>NULL,</td>
<td>NULL</td>
</tr>
<tr>
<td>DS9,</td>
<td>Deep Space 9,</td>
<td>DS9,</td>
<td>Bajor</td>
</tr>
<tr>
<td>74656,</td>
<td>Voyager,</td>
<td>74656,</td>
<td>Gamma Quadrant</td>
</tr>
<tr>
<td>75633,</td>
<td>Defiant,</td>
<td>75633,</td>
<td>Risa</td>
</tr>
</tbody>
</table>
## Outer Joins

### Ships

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1701</td>
<td>Enterprise</td>
</tr>
<tr>
<td>DS9</td>
<td>Deep Space 9</td>
</tr>
<tr>
<td>74656</td>
<td>Voyager</td>
</tr>
<tr>
<td>75633</td>
<td>Defiant</td>
</tr>
</tbody>
</table>

### Locations

<table>
<thead>
<tr>
<th>Ship</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS9</td>
<td>Bajor</td>
</tr>
<tr>
<td>74656</td>
<td>Gamma Quadrant</td>
</tr>
<tr>
<td>75633</td>
<td>Risa</td>
</tr>
</tbody>
</table>

#### Ships $\bowtie_{Ship=ID}$ Locations

<table>
<thead>
<tr>
<th>ID, Name, Ship, Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1701, Enterprise, NULL, NULL]</td>
</tr>
<tr>
<td>[DS9, Deep Space 9, DS9, Bajor]</td>
</tr>
<tr>
<td>[74656, Voyager, 74656, Gamma Quadrant]</td>
</tr>
<tr>
<td>[75633, Defiant, 75633, Risa]</td>
</tr>
</tbody>
</table>
## Outer Joins

<table>
<thead>
<tr>
<th>Join</th>
<th>Sym</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>[INNER] JOIN</td>
<td>⋈</td>
<td>Normal Join</td>
</tr>
<tr>
<td>LEFT OUTER JOIN</td>
<td>⩹</td>
<td>Keep dangling tuples from the left</td>
</tr>
<tr>
<td>RIGHT OUTER JOIN</td>
<td>⩹</td>
<td>Keep dangling tuples from the right</td>
</tr>
<tr>
<td>[FULL] OUTER JOIN</td>
<td>⩹</td>
<td>Keep all dangling tuples</td>
</tr>
</tbody>
</table>
Project 1 Review

Database Systems: The Complete Book
Ch. 5.1-5.2, 6.1-6.2, 6.4, 15.1-15.2
Project 1
Project 1
Project 1

All java files in src compiled and put into classpath

javac -cp jsqlparser.jar $(find src -name '*.java') -d build
jar -cf your_code.jar -C build
Project 1

All java files in src compiled and put into classpath
javac -cp jsqlparser.jar $(find src -name '*.java') -d build
jar -cf your_code.jar -C build

--data [path] specifies data directory
CREATE TABLE LINEITEM(...) stored in [path]/LINEITEM.dat
Project 1

All java files in src compiled and put into classpath
```
javac -cp jsqlparser.jar $(find src -name '*.java') -d build
jar -cf your_code.jar -C build
```

--data [path] specifies data directory

CREATE TABLE LINEITEM(...) stored in [path]/LINEITEM.dat

One or more SQL files with CREATE TABLE and SELECT statements
Project 1

All java files in src compiled and put into classpath
javac -cp jsqlparser.jar $(find src -name '*.java') -d build
jar -cf your_code.jar -C build

--data [path] specifies data directory
CREATE TABLE LINEITEM(...) stored in [path]/LINEITEM.dat

One or more SQL files with CREATE TABLE and SELECT statements

Evaluate the SELECT statements and print to stdout in ‘|’-delimited form
Project 1

All java files in src compiled and put into classpath
javac -cp jsqlparser.jar $(find src -name '*.java') -d build
jar -cf your_code.jar -C build

--data [path] specifies data directory
CREATE TABLE LINEITEM(...) stored in [path]/LINEITEM.dat

One or more SQL files with CREATE TABLE and SELECT statements

Evaluate the SELECT statements and print to stdout in ‘|’-delimited form

Any questions about the expected interface?
CREATE TABLE PLAYERS(
    ID string,
    FIRSTNAME string,
    LASTNAME string,
    FIRSTSEASON int,
    LASTSEASON int,
    WEIGHT int,
    BIRTHDATE date
);

SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
Statement stmt;
CCJSqlParser parser = ...;

while((stmt = parser.Statement()) != null) {
    // process stmt
}
```
CREATE TABLE PLAYERS(
    ID string,
    FIRSTNAME string,
    LASTNAME string,
    FIRSTSEASON int,
    LASTSEASON int,
    WEIGHT int,
    BIRTHDATE date
);

SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```
CREATE TABLE PLAYERS(
  ID string,
  FIRSTNAME string,
  LASTNAME string,
  FIRSTSEASON int,
  LASTSEASON int,
  WEIGHT int,
  BIRTHDATE date
);

SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
CREATE TABLE PLAYERS(
    ID string,
    FIRSTNAME string,
    LASTNAME string,
    FIRSTSEASON int,
    LASTSEASON int,
    WEIGHT int,
    BIRTHDATE date
);

There is a table named Players - with 7 attributes.
- with the given schema
- with a data file “PLAYERS.dat”
(Save this information for later)
CREATE TABLE PLAYERS(
    ID string,
    FIRSTNAME string,
    LASTNAME string,
    FIRSTSEASON int,
    LASTSEASON int,
    WEIGHT int,
    BIRTHDATE date
);

There is a table named Players
- with 7 attributes.
- with the given schema
- with a data file "PLAYERS.dat"
(Save this information for later)
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
stmt instanceof Select

Saved Schema
PLAYERS.dat
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;

stmt instanceof Select

SelectBody body = stmt.getSelectBody()
// body is either an instanceof
// Union or PlainSelect
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;

stmt instanceof Select

SelectBody body = stmt.getSelectBody()
// body is either an instanceof
// Union or PlainSelect

body.getFromItem()

body.getWhere()

body.getSelectItems()
```
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;
```

```
stmt instanceof Select

SelectBody body = stmt.getSelectBody()
// body is either an instanceof
// Union or PlainSelect

body.getFromItem()

body.getWhere()

body.getSelectItems()
```
Trust me on this one…

Don’t try to evaluate SQL directly
Relational Algebra

**Set Relational Algebra**
Select (σ), Project (π), Join (⋈), Union (∪), Relation (R)

**Bag-Relational Algebra**
Distinct (δ), Outer Joins

**List-Relational Algebra**
Sort (τ), Limit

**Arithmetic Expressions**
*Extended Project* (π), Aggregation (Σ), Grouping (ɣ)
Relational Algebra

Set Relational Algebra
Select (σ), Project (π), Join (⋈), Union (∪), Relation (R)

Bag-Relational Algebra
Distinct (δ), Outer Joins

List-Relational Algebra
Sort (τ), Limit

Arithmetic Expressions
Extended Project (π), Aggregation (Σ), Grouping (γ)
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;

body.getFromItem()

body.getWhere()

body.getSelectItems()
```sql
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;
```

```python
query = Players
body.getFromItem()

body.getWhere()

body.getSelectItems()
```
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;

query = Players

query = σ(where, query)

body.getSelectItems()
\textbf{SELECT} FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE 
\textbf{FROM} PLAYERS 
\textbf{WHERE} WEIGHT > 200;

body.getFromItem() 
\rightarrow query = \text{Players} 

body.getWhere() 
\rightarrow query = \sigma(\text{where}, \text{query}) 

body.getSelectItems() 
\rightarrow query = \pi(\text{items}, \text{query})
```
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;
```

```
body.getFromItem()  
\[
query = \text{Players}
\]

body.getWhere()  
\[
query = \sigma(\text{where}, \ query)
\]

body.getSelectItems()  
\[
query = \pi(\text{items}, \ query)
\]
```
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;

\[ \pi(\text{items}) \big\upharpoonright \sigma(\text{where}) \]
\[ \text{Players} \]
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;

\[ \pi(\text{items}) \mid \sigma(\text{where}) \mid \text{Players} \]
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE 
FROM PLAYERS 
WHERE WEIGHT > 200;

π(items) 
σ(where) 
Players
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;
```
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;
```

**Iterators:** Read 1 row at a time

```
π(items)
σ(where)
Players
```
```
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;
```

Iterators: Read 1 row at a time
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;

Iterators: Read 1 row at a time
```sql
SELECT FIRSTNAME, LASTNAME, WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT > 200;
```

```java
public class ProjectIterator implements SqlIterator {
    SqlIterator source;
    Column[] inputSchema;
    Expression[] outputExpressions;

    LeafValue[] getNext() {
        // read one row from source
        // use ExpressionLib for output
        // return computed output
    }
}
```
public class SelectBody {

    ...  
    Expression getWhere(){ ...  }
    Expression getHaving(){ ...  }

    ...
}

public class SelectExpressionItem {

    ...
    Expression getExpression(){ ...  }

    ...
}
**net.sf.jsqlparser.expression.PrimitiveValue**

**Implementations**

- BooleanValue
- LongValue
- DoubleValue
- StringValue
- DateValue
- TimestampValue
- TimeValue

**Methods**

- double toDouble()
- long toLong()
- String toString()
- boolean toBool()
LeafValue eval(Column c)

Implement this
(The reference implementation caches this for each row)

LeafValue eval(Expression e)

Then call this
CREATE TABLE R ( A int, B int );
CREATE TABLE S ( B int, C int );

SELECT R.A, S.C FROM R, S WHERE R.B = S.B;

body.getFromItem()

query = R

body.getWhere()

query = σ(where, query)

body.getSelectItems()

query = π(items, query)
CREATE TABLE R ( A int, B int );
CREATE TABLE S ( B int, C int );

SELECT R.A, S.C FROM R, S WHERE R.B = S.B;

body.getFromItem()

\[ \text{query} = R \]

body.getJoins()

body.getWhere()

\[ \text{query} = \sigma(\text{where, query}) \]

body.getSelectItems()

\[ \text{query} = \pi(\text{items, query}) \]
CREATE TABLE R ( A int, B int );
CREATE TABLE S ( B int, C int );

SELECT R.A, S.C FROM R, S WHERE R.B = S.B;

body.getFromItem()
  \[\textbf{query} = R\]

body.getJoins()
  \[\textbf{for}(j : \text{joins}) \{ \textbf{query} = \text{x}(\textbf{query}, j.\text{table}) \}\]

body.getWhere()
  \[\textbf{query} = \sigma(\text{where}, \textbf{query})\]

body.getSelectItems()
  \[\textbf{query} = \pi(\text{items}, \textbf{query})\]
CREATE TABLE R ( A int, B int );
CREATE TABLE S ( B int, C int );

SELECT R.A, S.C FROM R, S WHERE R.B = S.B;

body.getFromItem()

\[
\text{query} = R
\]

body.getJoins()

for(j : joins) { \text{query} = \text{x(query, j.table)} }

body.getWhere()

\[
\text{query} = \sigma(\text{where, query})
\]

body.getSelectItems()

\[
\text{query} = \pi(\text{items, query})
\]
Why have both Relational Algebra AND Iterators?
Why have both Relational Algebra AND Iterators?

Not strictly necessary…
… but convenient

**Optimizers** need…
- flexibility
- to easily rearrange subtrees
- evolving schemas

**Iterators** need…
- to be fast and efficient
- precomputed schemas (for Eval)
- state (Sort, Aggregate, Join)

Separate representations for optimization and execution
Extra Reference Slides
Select

CreateTable

Saved Schema

π σ θ R S

Selectitems (π)

having (σ)

agg

where (σ)

source (X, ⊓)

select

distinct

order by

lim

U

(Output)

PLAYERS.dat

Iterator

CSV
Iterators

```c
void open() {
    // call open() on child iterators
    // prepare the iterator
}

Tuple getNext() {
    // read, process, and return a tuple
}

void close() {
    // clean-up the iterator
    // call close() on child iterators
}
```