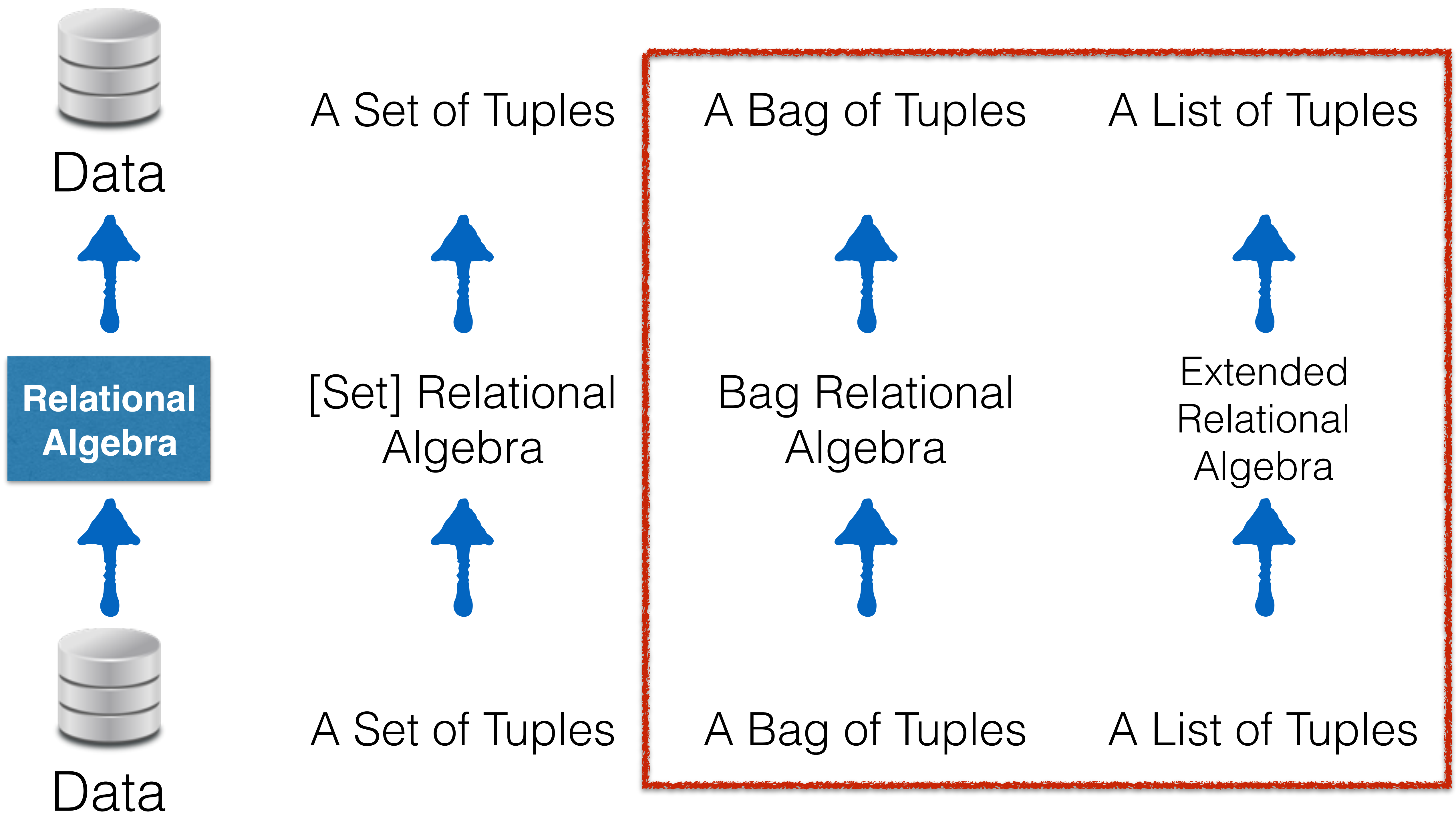
Extended RA

Database Systems: The Complete Book

Ch 5.1-5.2, 15.4

Relational Algebra



What's Missing?

<u>Set Relational Algebra</u>

Select (σ), Project (π), Join ($\bowtie$), Union ($\cup$)

<u>Bag-Relational Algebra</u>

Distinct (δ), Outer Joins

<u>List-Relational Algebra</u>

Sort (τ), Limit

<u>Arithmetic Expressions</u>

Extended Projection (π), Aggregation (Σ), Grouping (γ)

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

$\pi_{\textit{Total}:\texttt{Price*(1-Discount)},\ \textit{Profit}:\texttt{Cost-Price*(1-Discount)}}$ Lineitem

Sort, Limit

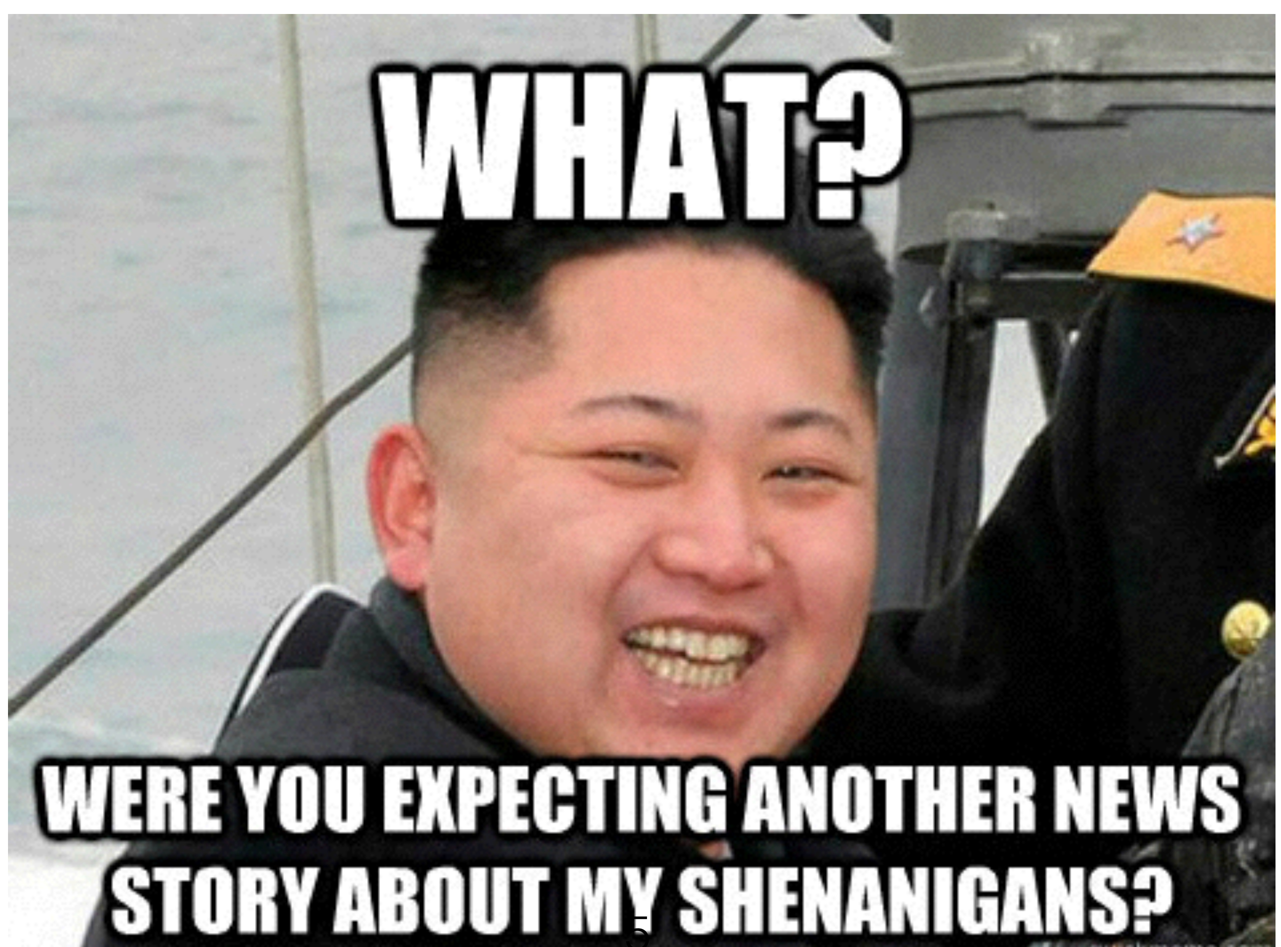
Sort a List

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()?

Sort

```
void open() {
  child.open()
  buffer = new List<Tuple>()
  while((next = child.getNext()) != null)
    buffer.add(next)
  Collections.sort(buffer)
}
```

Sort

```
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)
```

Single Column/Expression

Aggregation

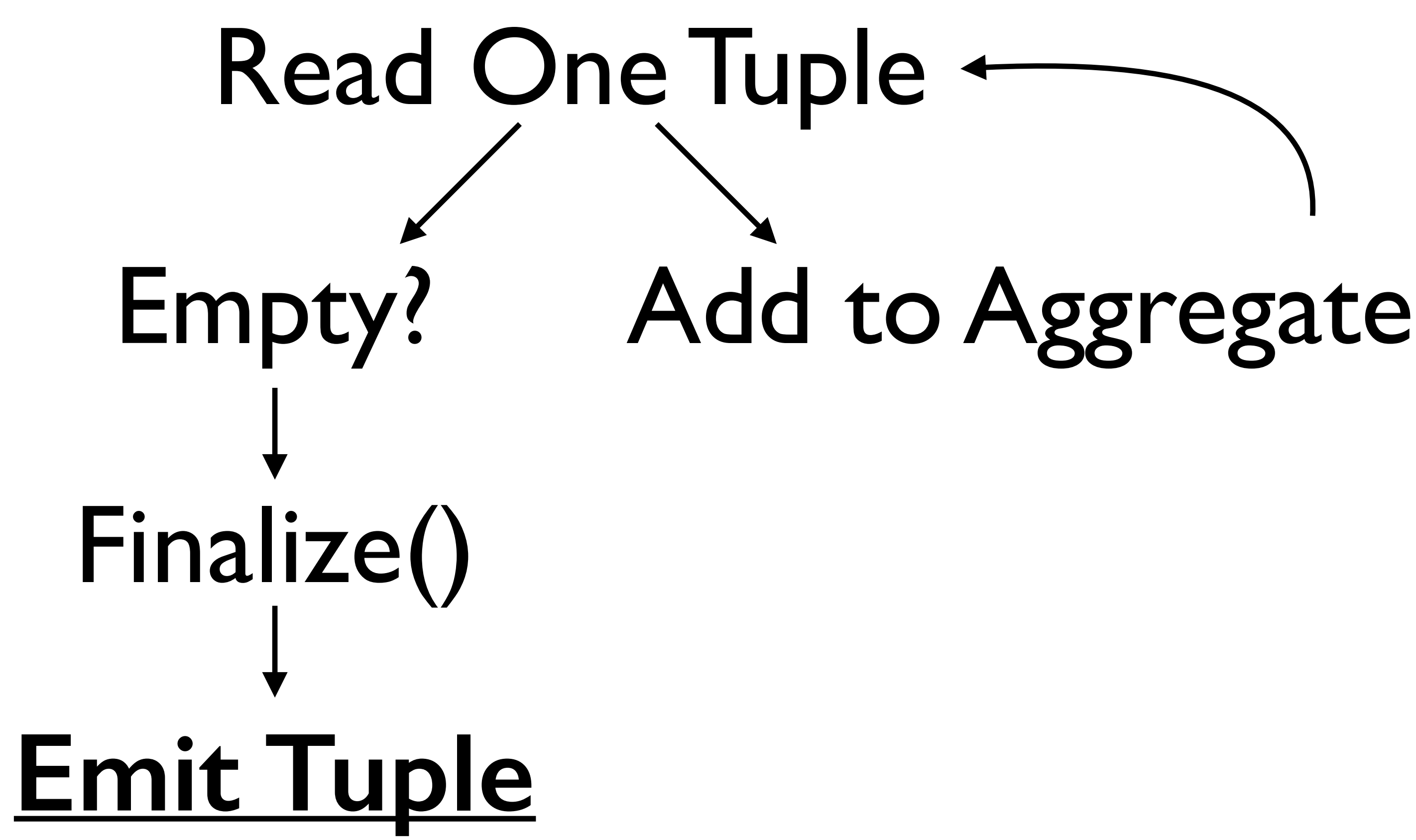
How do we implement these?

Aggregation - Fold

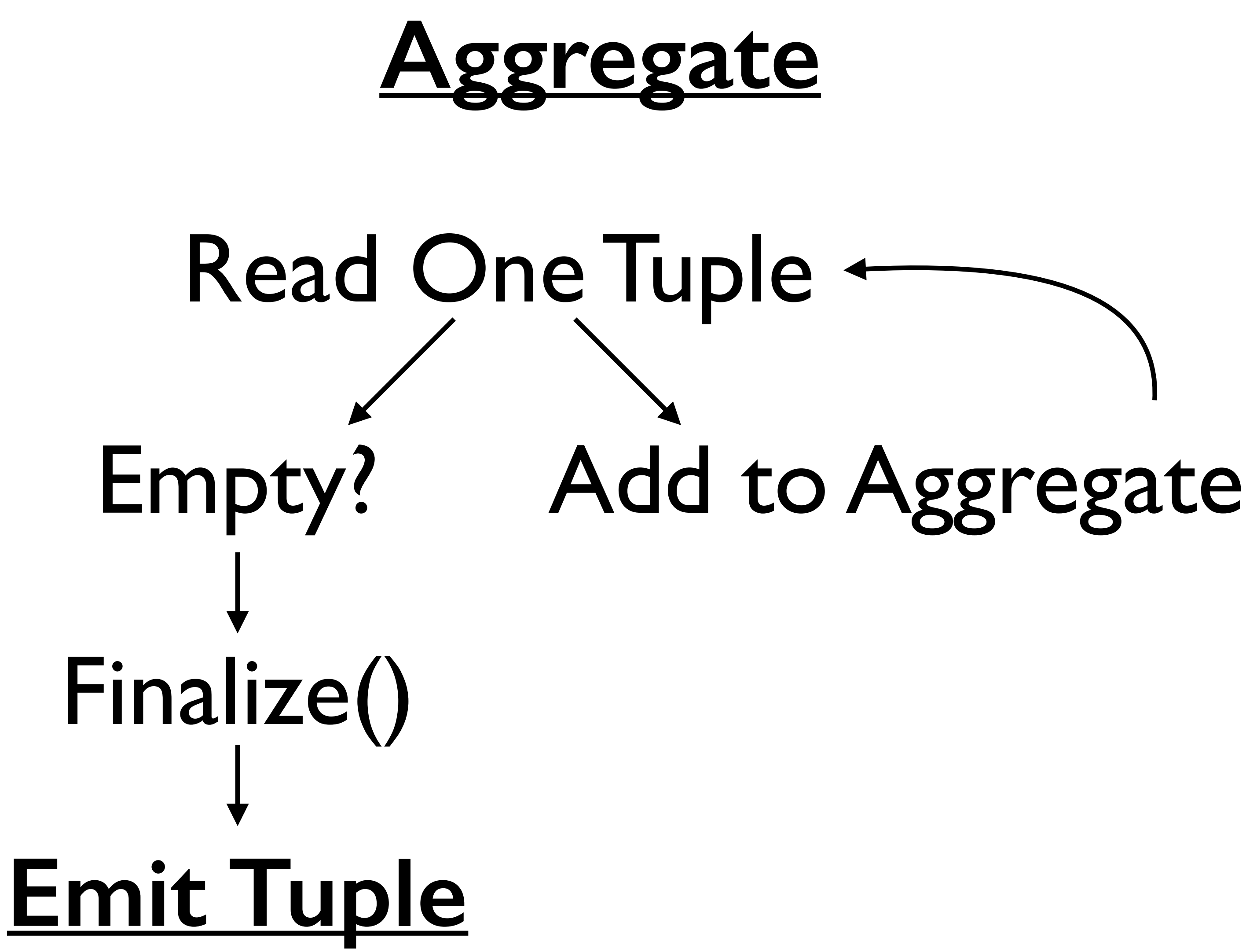
```
void Init() {
  // prepare the aggregate
}

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

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

Iterators

Aggregate



Iterators



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

```
COUNT(DISTINCT A)
```

Group Work

Design folds for each of these aggregates

`COUNT(DISTINCT A)`

What is the Working Set Size?

Grouping

$\gamma_{A,\ B,\ \ldots,\ Ccnt:\mathtt{COUNT(C)},\ Dsum:\mathtt{SUM(D)},\ \ldots}$

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

$\gamma_{A,\ B,\ \ldots,\ Ccnt:\mathtt{COUNT(C)},\ Dsum:\mathtt{SUM(D)},\ \ldots}$

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?

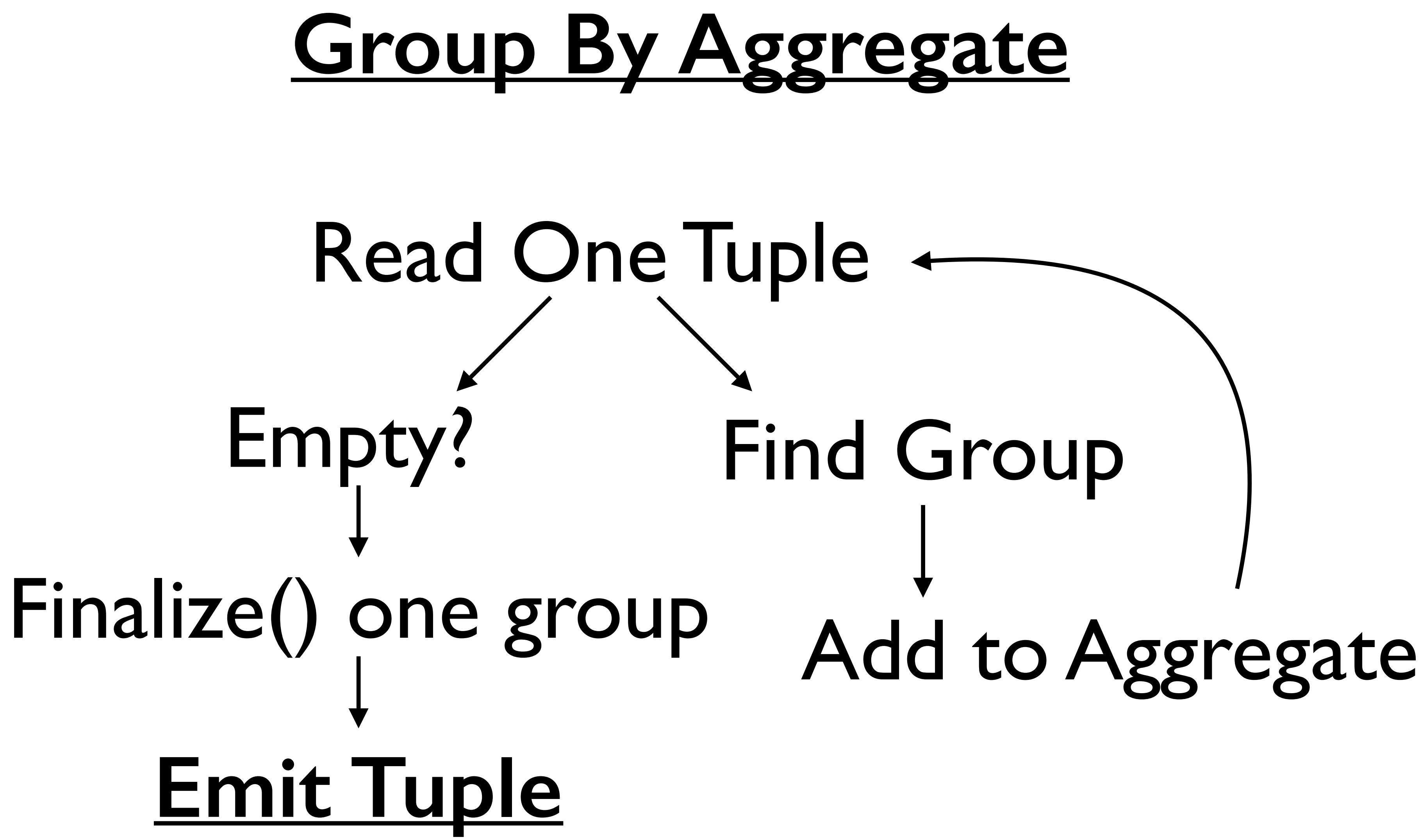
Finalize() one group

Emit Tuple

Find Group

Add to Aggregate

Iterators



What is the Working Set Size?

Iterators

<u>Group By Aggregate</u>

Read One Tuple

Empty?

Finalize() one group

<u>Emit Tuple</u>

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

```
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

```
 ID,       Name
[1701,    Enterprise  ]
[DS9,     Deep Space 9]
[74656,   Voyager     ]
[75633,   Defiant     ]
```

```
 Ship,    Location
[1701,    Subspace Anomaly]
[DS9,     Bajor           ]
[74656,   Gamma Quadrant  ]
[75633,   Risa            ]
```

Outer Joins

Ships

ID,	Name
[1701,	Enterprise]
[DS9,	Deep Space 9]
[74656,	Voyager]
[75633,	Defiant]

Locations

Ship,	Location
[DS9,	Bajor]
[74656,	Gamma Quadrant]
[75633,	Risa]

Outer Joins

Ships

ID,	Name
[1701,	Enterprise]
[DS9,	Deep Space 9]
[74656,	Voyager]
[75633,	Defiant]

Locations

Ship,	Location
[DS9,	Bajor]
[74656,	Gamma Quadrant]
[75633,	Risa]

What is the result of this query?

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

Outer Joins

Ships

ID,	Name
[1701,	Enterprise]
[DS9,	Deep Space 9]
[74656,	Voyager]
[75633,	Defiant]

Locations

Ship,	Location
[DS9,	Bajor]
[74656,	Gamma Quadrant]
[75633,	Risa]

What is the result of this query?

$$\pi_{Location}\ \sigma_{Name='Enterprise'}(Ships \bowtie_{Ship=ID} Locations)$$

Is an empty result what we're looking for?

Outer Joins

Ships

ID,	Name
[1701,	Enterprise]
[DS9,	Deep Space 9]
[74656,	Voyager]
[75633,	Defiant]

Locations

Ship,	Location
[DS9,	Bajor]
[74656,	Gamma Quadrant]
[75633,	Risa]

ID,	Name,	Ship,	Location
[1701,	Enterprise,	NULL,	NULL]
[DS9,	Deep Space 9,	DS9,	Bajor]
[74656,	Voyager,	74656,	Gamma Quadrant]
[75633,	Defiant,	75633,	Risa]

Outer Joins

Ships

ID,	Name
[1701,	Enterprise]
[DS9,	Deep Space 9]
[74656,	Voyager]
[75633,	Defiant]

Locations

Ship,	Location
[DS9,	Bajor]
[74656,	Gamma Quadrant]
[75633,	Risa]

Ships $⟕_{Ship=ID}$ Locations

ID,	Name,	Ship,	Location
[1701,	Enterprise,	NULL,	NULL]
[DS9,	Deep Space 9,	DS9,	Bajor]
[74656,	Voyager,	74656,	Gamma Quadrant]
[75633,	Defiant,	75633,	Risa]

Outer Joins

Join	Sym	Effect
`[INNER] JOIN`	$\bowtie$	Normal Join
`LEFT OUTER JOIN`	$⟕$	Keep dangling tuples from the left
`RIGHT OUTER JOIN`	$⟖$	Keep dangling tuples from the right
`[FULL] OUTER JOIN`	$⟗$	Keep all dangling tuples

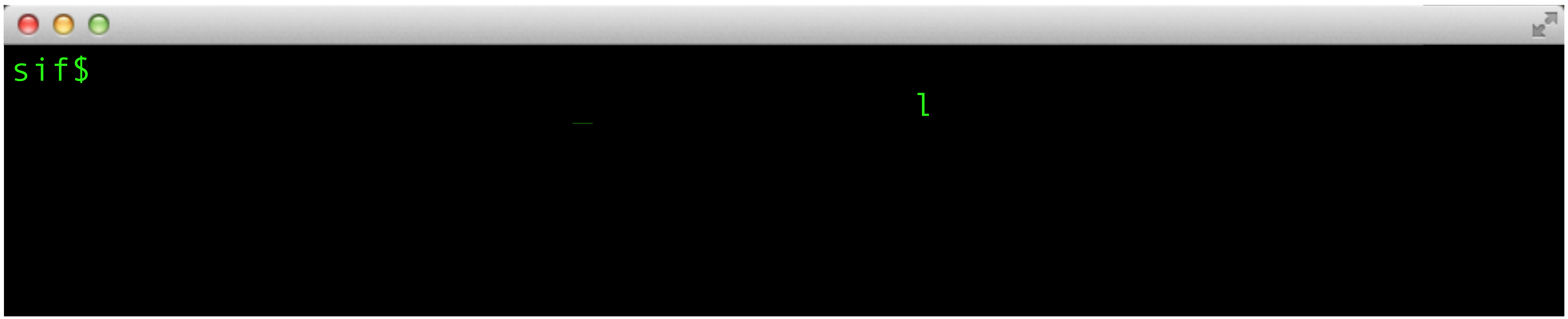
Project 1 Review

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

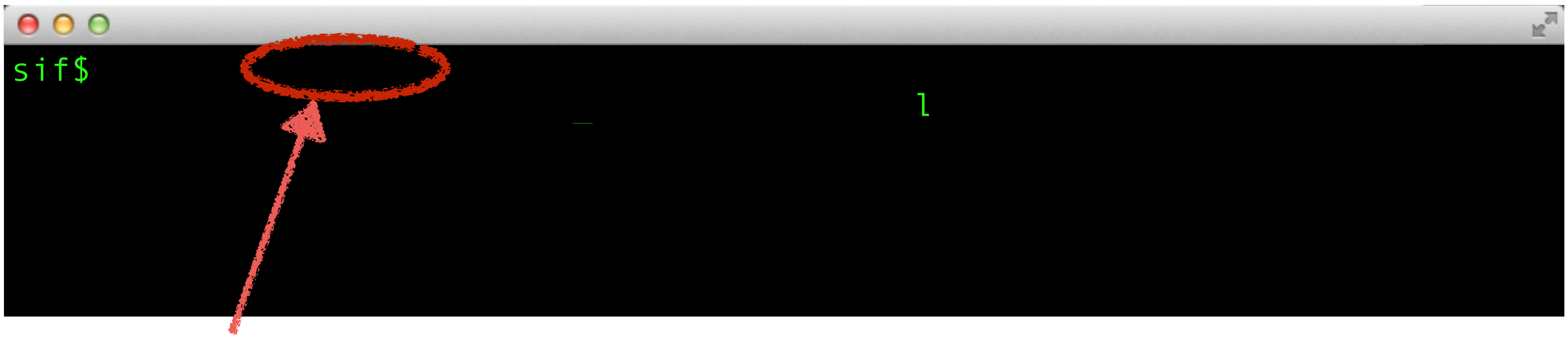
Project 1

```
sif$
```

Project 1



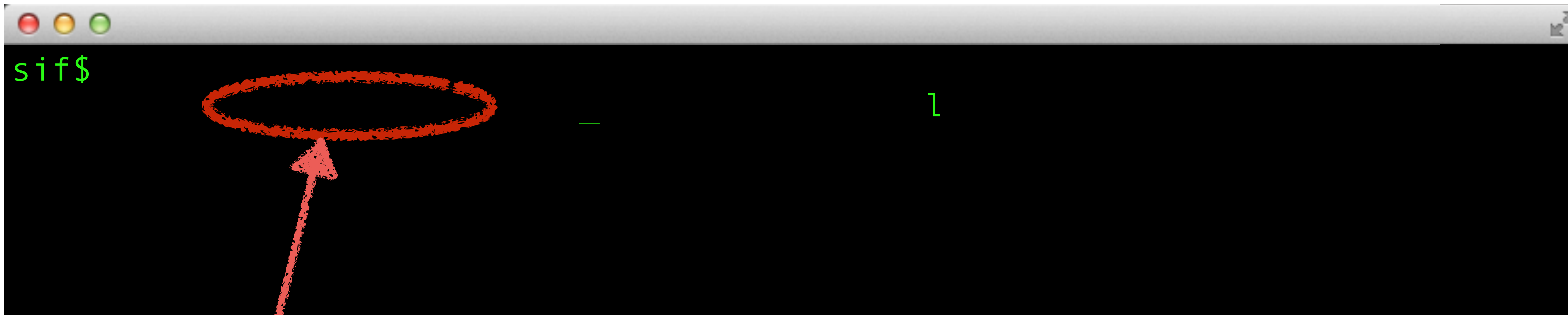
Project 1



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



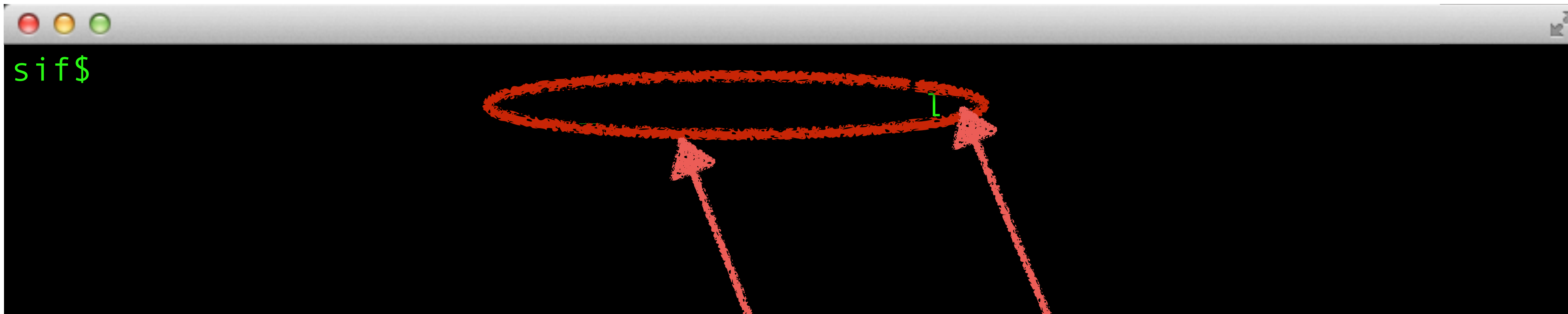
All java files in src compiled and put into classpath

```
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



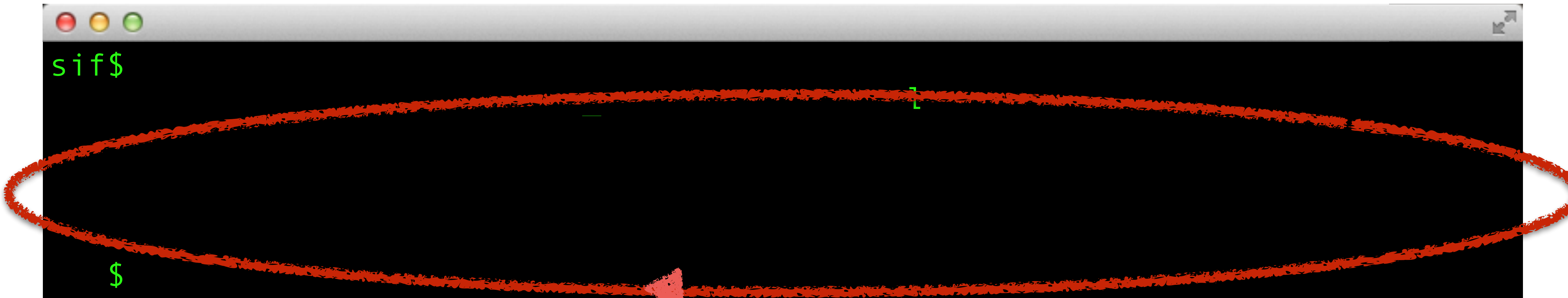
All java files in src compiled and put into classpath

```
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
```

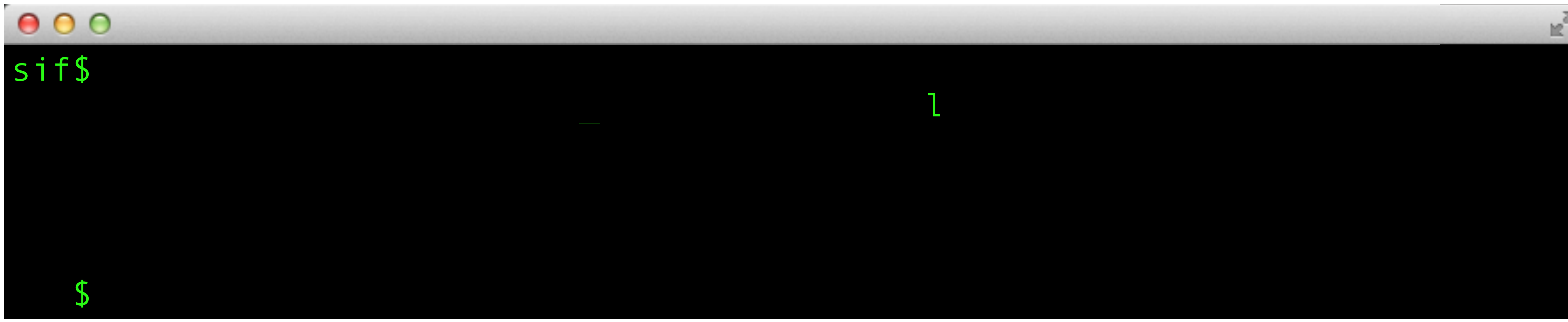
`--data [path]` specifies data directory

`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?

.sql

.sql

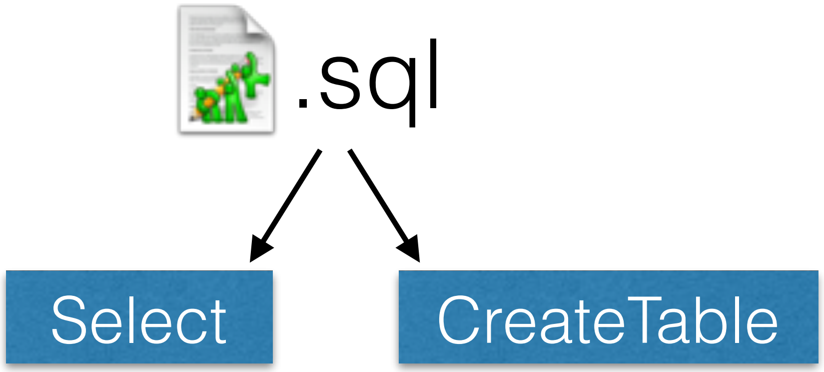
```
CREATE TABLE PLAYERS(
  ID string,
  FIRSTNAME string,
  LASTNAME string,
  FIRSTSEASON int,
  LASTSEASON int,
  WEIGHT int,
  BIRTHDATE date
);

SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

.sql (JSqlParser)

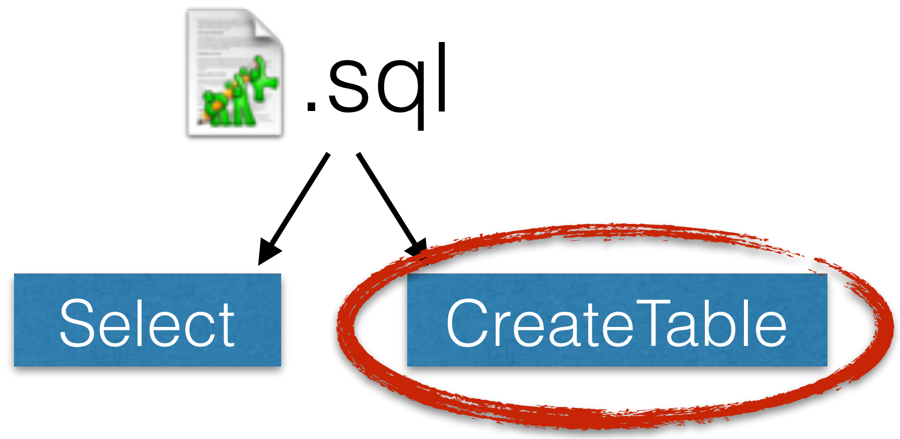
```
Statement stmt;
CCJSqlParser parser = …;

while((stmt = parser.Statement()) != null)
{
  // process stmt
}
```

.sql

```sql
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;
```



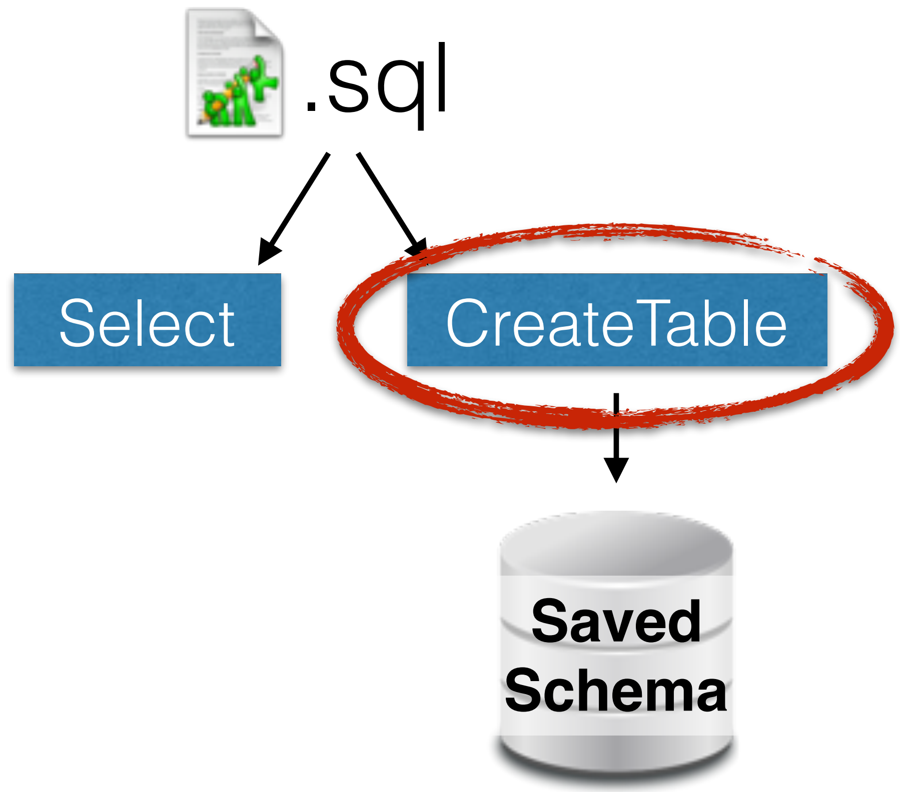

```
CREATE TABLE PLAYERS(
    ID string,
    FIRSTNAME string,
    LASTNAME string,
    FIRSTSEASON int,
    LASTSEASON int,
    WEIGHT int,
    BIRTHDATE date
);
```

`stmt instanceof CreateTable`

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
);
```

↓

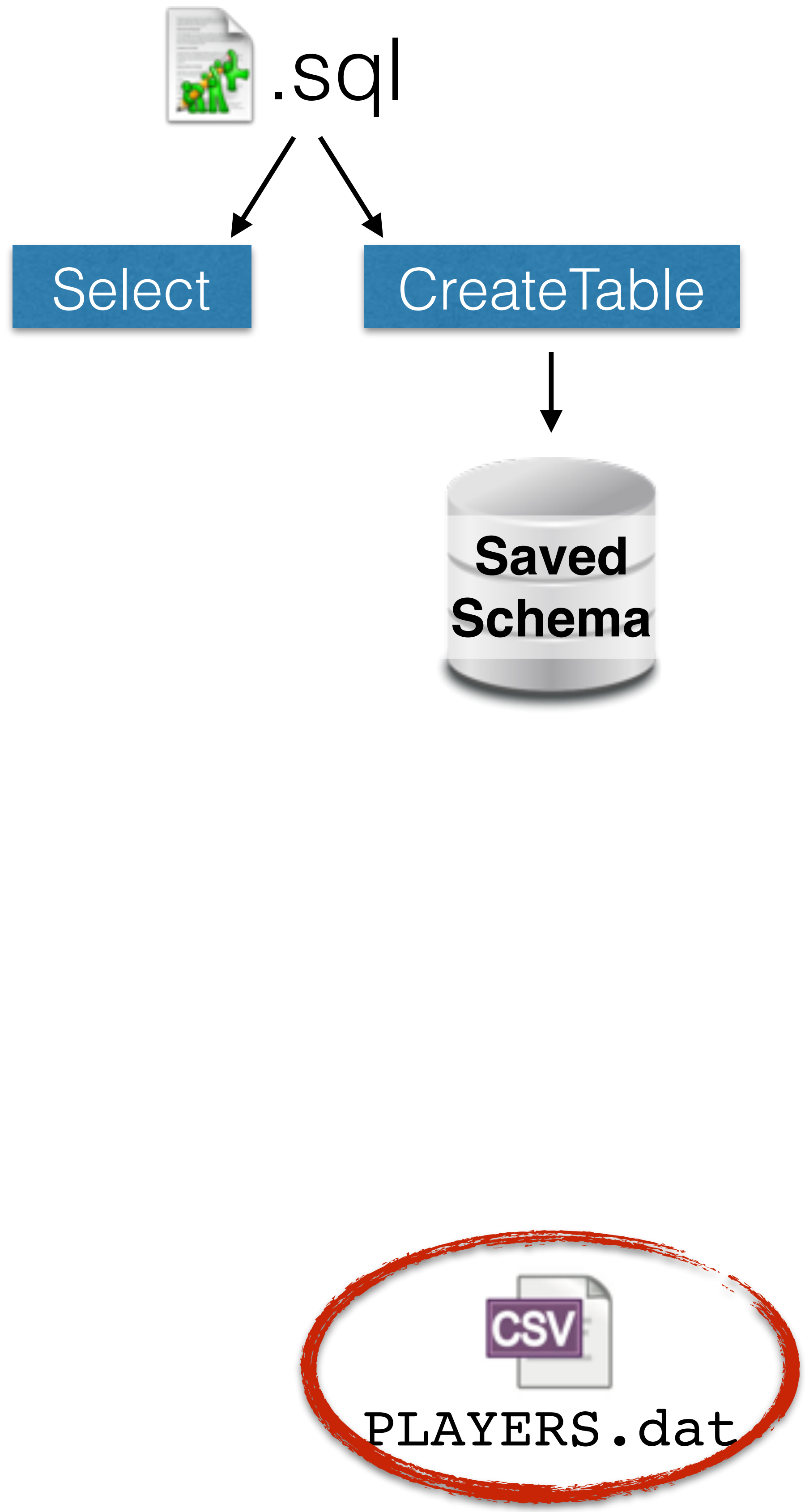
`stmt instanceof` **`CreateTable`**

↓

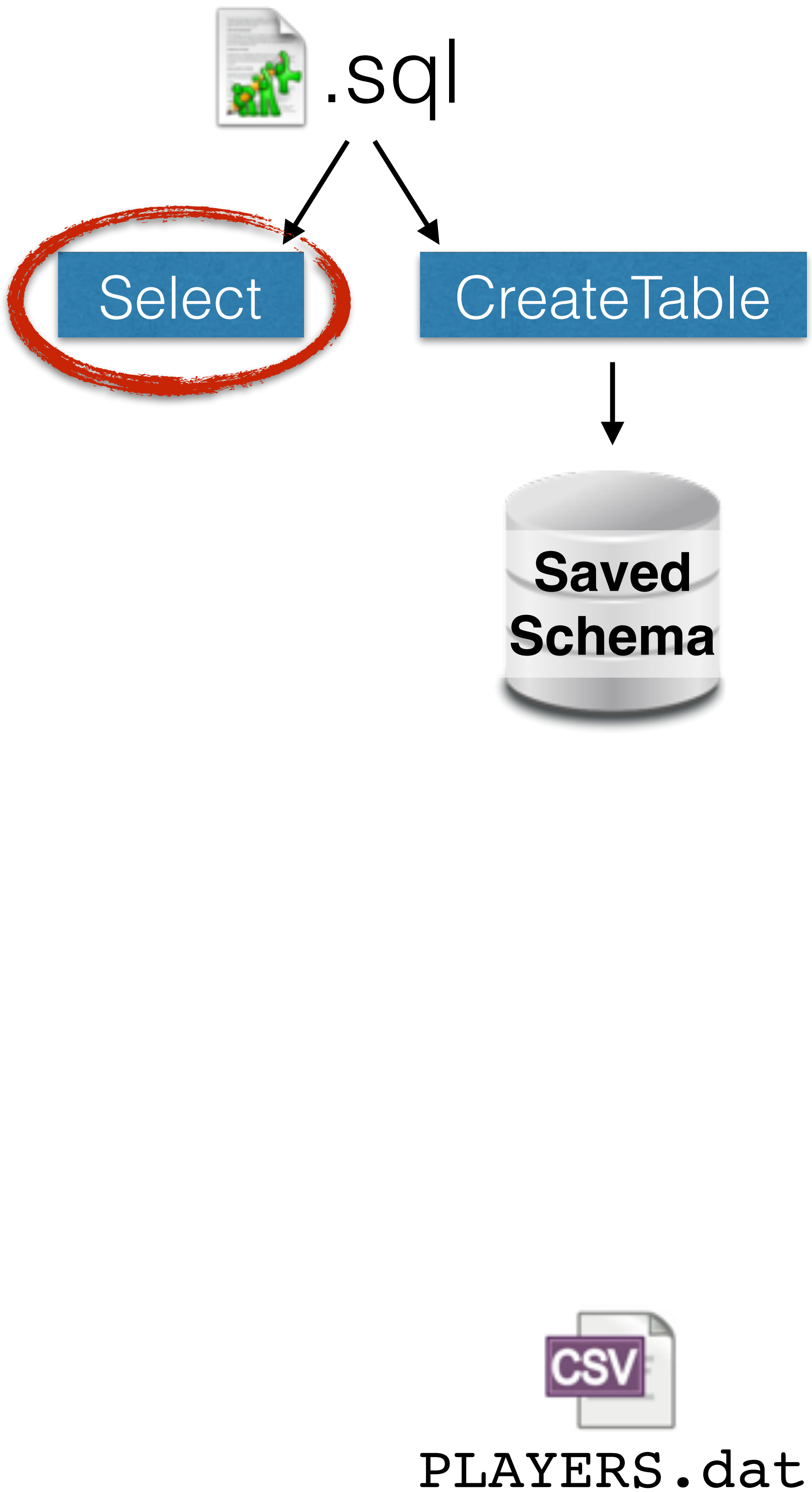
There is a table named `Players`

- with 7 attributes.
- with the given schema
- with a data file “PLAYERS.dat”

(Save this information for later)

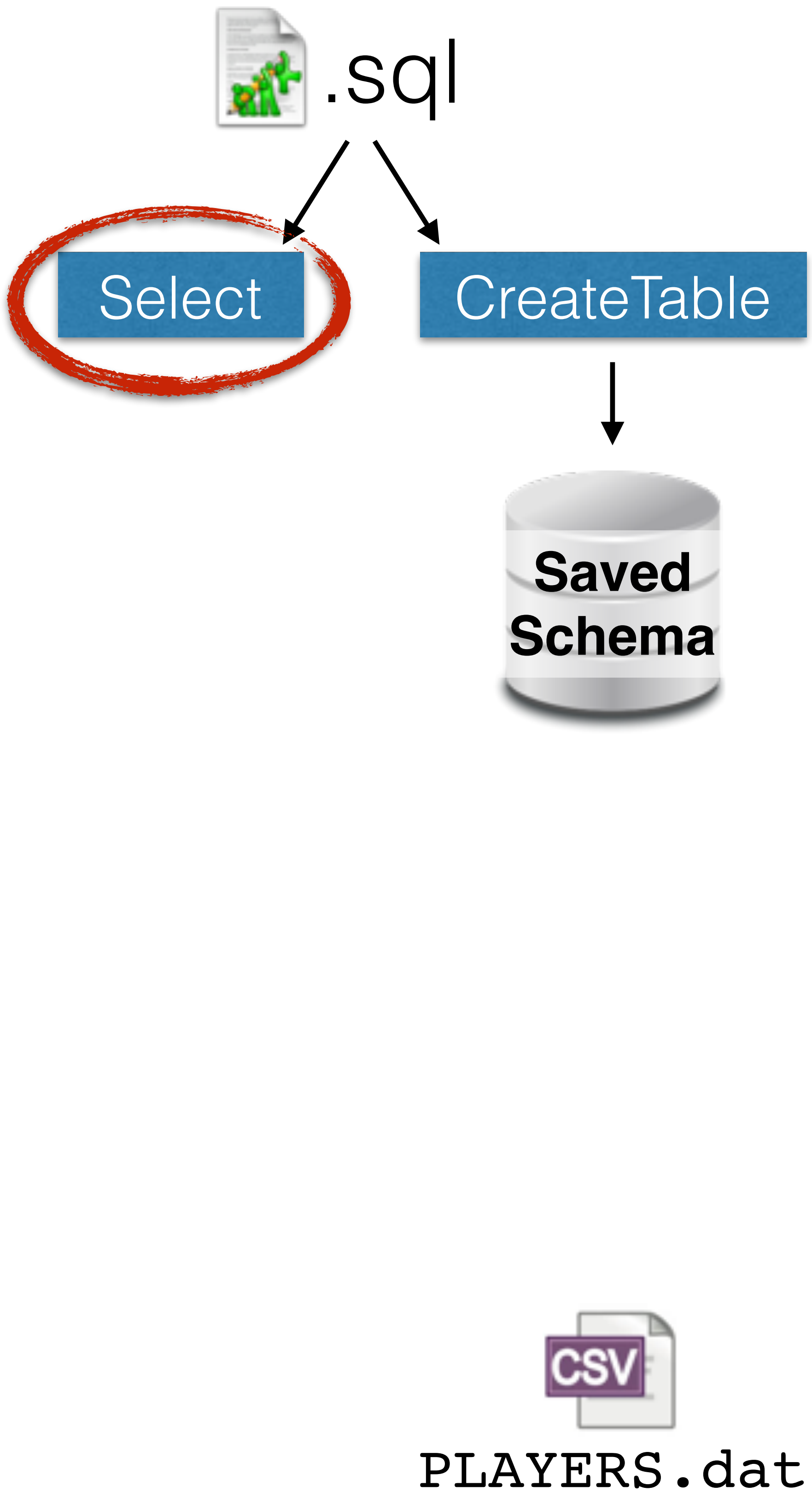



```
ABDELAL01|Alaa|Abdelnaby|1990|1994|240|1968-06-24
ABDULKA01|Kareem|Abdul-jabbar|1969|1988|225|1947-04-16
ABDULMA01|Mahmo|Abdul-rauf|1990|2000|162|1969-03-09
ABDULTA01|Tariq|Abdul-wahad|1997|2002|223|1974-11-03
ABDURSH01|Shareef|Abdur-rahim|1996|2007|225|1976-12-11
ABERNTO01|Tom|Abernethy|1976|1980|220|1954-05-06
ABLEFO01|Forest|Able|1956|1956|180|1932-07-27
ABRAMJO01|John|Abramovic|1946|1947|195|1919-02-09
ACKERAL01|Alex|Acker|2005|2008|185|1983-01-21
ACKERDO01|Donald|Ackerman|1953|1953|183|1930-09-04
ACRESMA01|Mark|Acres|1987|1992|220|1962-11-15
ACTONCH01|Charles|Acton|1967|1967|210|1942-01-11
ADAMSAL01|Alvan|Adams|1975|1987|210|1954-07-19
ADAMSDO01|Don|Adams|1970|1976|210|1947-11-27
ADAMSGE01|George|Adams|1972|1974|210|1949-05-15
ADAMSMI01|Michael|Adams|1985|1995|162|1963-01-19
AdamsHa01|Hassan|Adams|2006|2008|220|1984-06-20
ADDISRA01|Rafael|Addison|1986|1996|215|1964-07-22
ADELMRI01|Rick|Adelman|1968|1974|175|1946-06-16
AfflaAr01|Arron|Afflalo|2007|2009|215|1985-11-15
AgerMa01|Maurice|Ager|2006|2008|202|1984-02-09
AGUIRMA01|Mark|Aguirre|1981|1993|232|1959-12-10
AhearBl01|Blake|Ahearn|2007|2008|190|1984-05-27
AINGEDA01|Danny|Ainge|1981|1994|175|1959-03-17
AITCHMA01|Matt|Aitch|1967|1967|230|1944-09-21
AjincAl01|Alexis|Ajinca|2008|2009|220|1988-05-06
AKINHE01|Henry|Akin|1966|1968|225|1944-07-31
ALARIMA01|Mark|Alarie|1986|1990|217|1963-12-11
ALCORGA01|Gary|Alcorn|1959|1960|225|1936-10-08
AldriLa01|LaMarcus|Aldridge|2006|2009|240|1985-07-19
ALEKSCH01|Chuck|Aleksinas|1984|1984|260|1959-02-26
ALEXACO01|Cory|Alexander|1995|2004|185|1973-06-22
ALEXACO02|Courtney|Alexander|2000|2002|205|1977-04-27
ALEXAGA01|Gary|Alexander|1993|1993|240|1969-11-01
ALEXAVI01|Victor|Alexander|1991|2001|265|1969-08-31
AlexaJo01|Joe|Alexander|2008|2009|230|1986-12-26
ALFORST01|Steve|Alford|1987|1990|183|1964-11-23
ALLENBI01|Bill|Allen|1967|1967|205|1945-01-01
ALLENBO01|Bob|Allen|1968|1968|205|1946-07-17
ALLENJE01|Jerome|Allen|1995|1995|184|1973-01-28
ALLENLU01|Lucius|Allen|1969|1978|175|1947-09-26
ALLENMA01|Malik|Allen|2001|2009|225|1978-06-27
ALLENRA01|Randy|Allen|1988|1989|220|1965-01-26
ALLENRA02|Ray|Allen|1996|2009|205|1975-07-20
ALLENWI01|Willie|Allen|1971|1971|230|1949-02-08
ALLENTO01|Tony|Allen|2004|2009|214|1982-01-11
ALLISOD01|Odis|Allison|1971|1971|195|1949-10-02
...
```




```
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

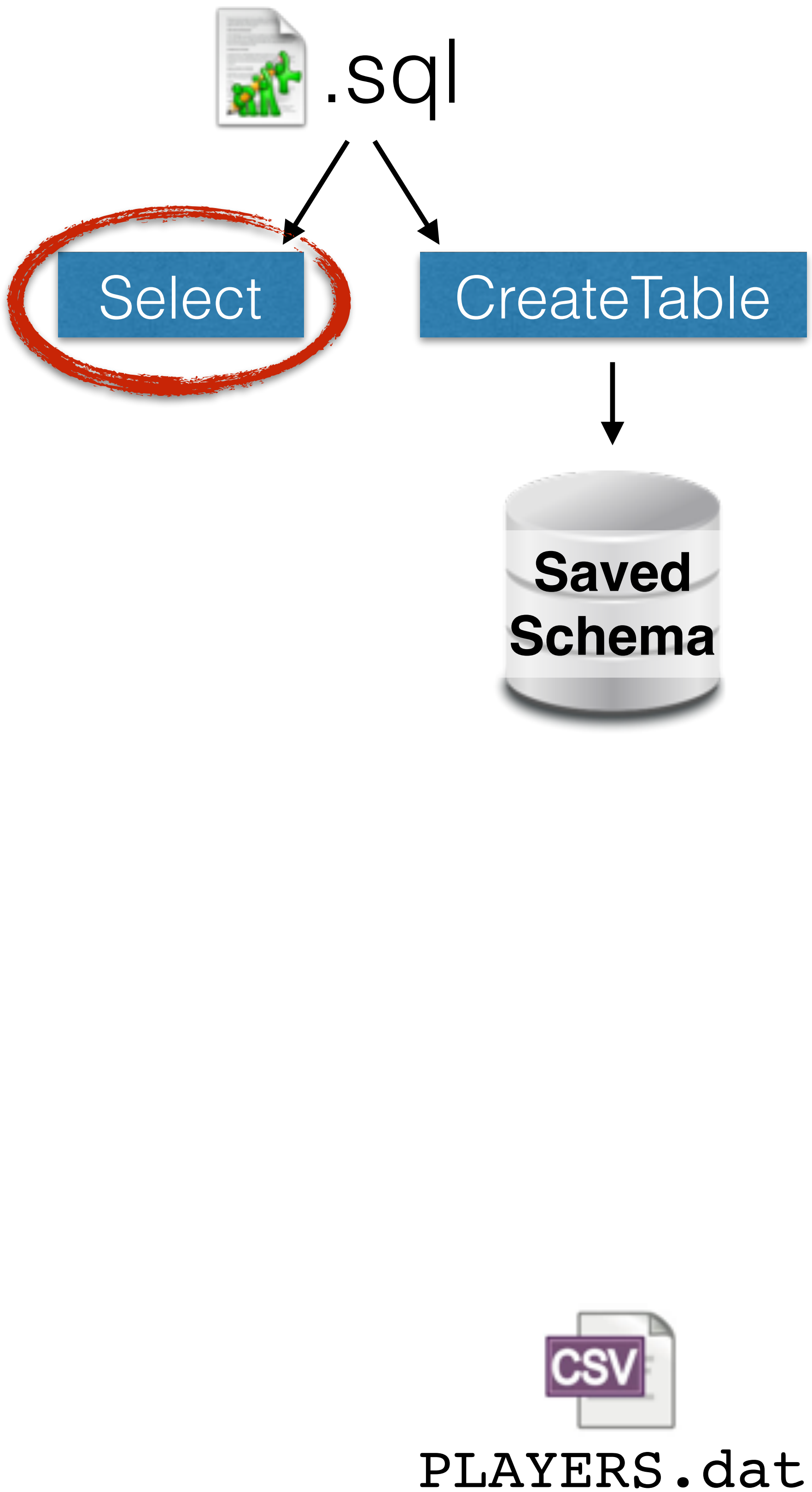
```
stmt instanceof Select
```




```
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
```

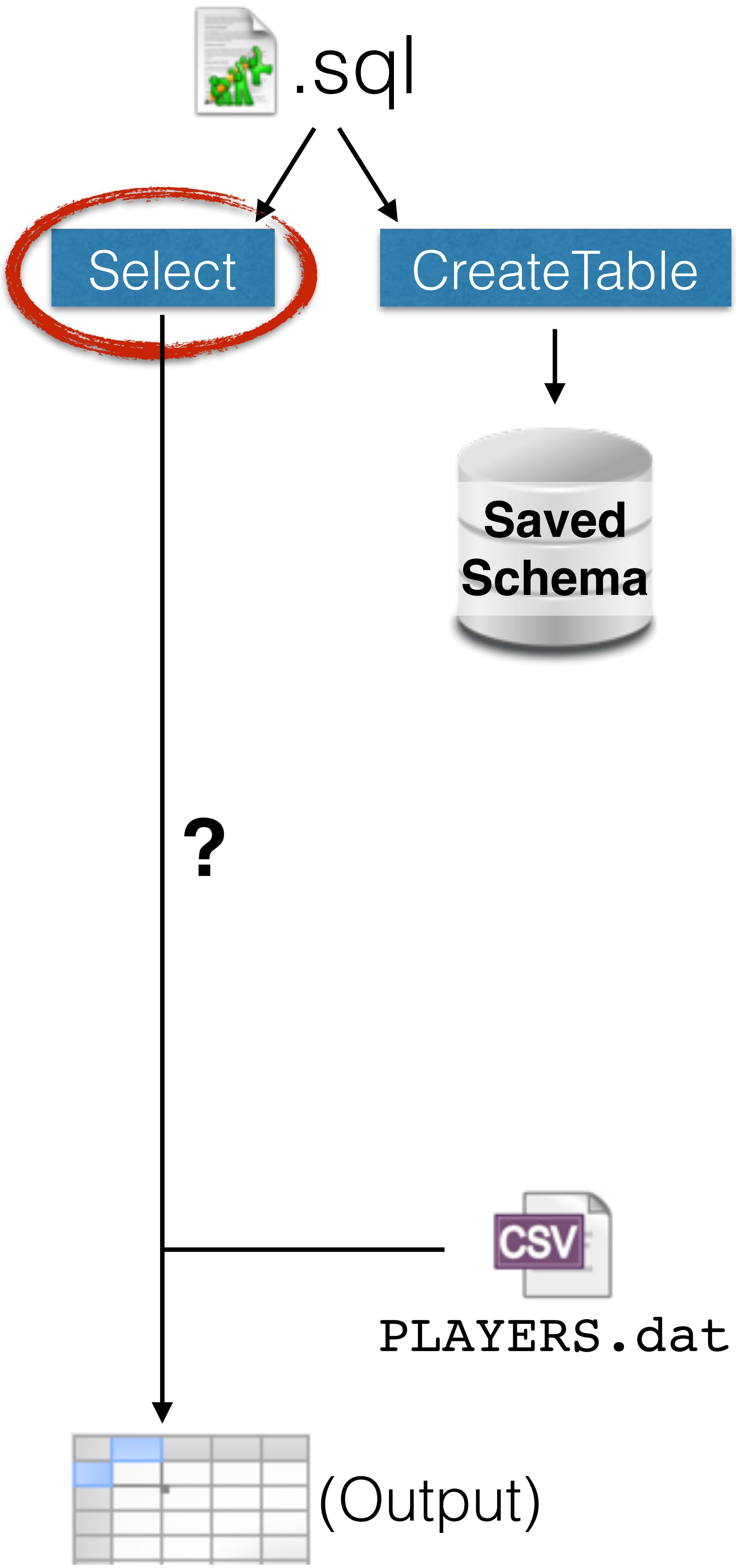



```sql
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

```java
stmt instanceof Select
```

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

```java
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

<u>Set Relational Algebra</u>

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

<u>Bag-Relational Algebra</u>

Distinct (δ), Outer Joins

<u>List-Relational Algebra</u>

Sort (τ), Limit

<u>Arithmetic Expressions</u>

Extended Project (π), Aggregation (Σ), Grouping (ɣ)

Relational Algebra

<u>Set Relational Algebra</u>

Select (σ), ~~Project (π)~~, Join (⋈), Union (∪), Relation (**R**)

<u>Bag-Relational Algebra</u>

~~Distinct (δ)~~, ~~Outer Joins~~

<u>List-Relational Algebra</u>

Sort (τ), Limit

<u>Arithmetic Expressions</u>

Extended Project (π), ~~Aggregation (Σ)~~, Grouping (ɣ)


```
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;
```

`body.getFromItem()`

↓

query = Players

`body.getWhere()`

`body.getSelectItems()`


```
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

`body.getFromItem()`

↓

query = Players

`body.getWhere()`

↓

query = σ(where, query)

`body.getSelectItems()`


```
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

`body.getFromItem()`

↓

query = Players

`body.getWhere()`

↓

query = σ(where, query)

`body.getSelectItems()`

↓

query = π(items, query)


```sql
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

`body.getFromItem()`

$$\downarrow$$

query = Players

`body.getWhere()`

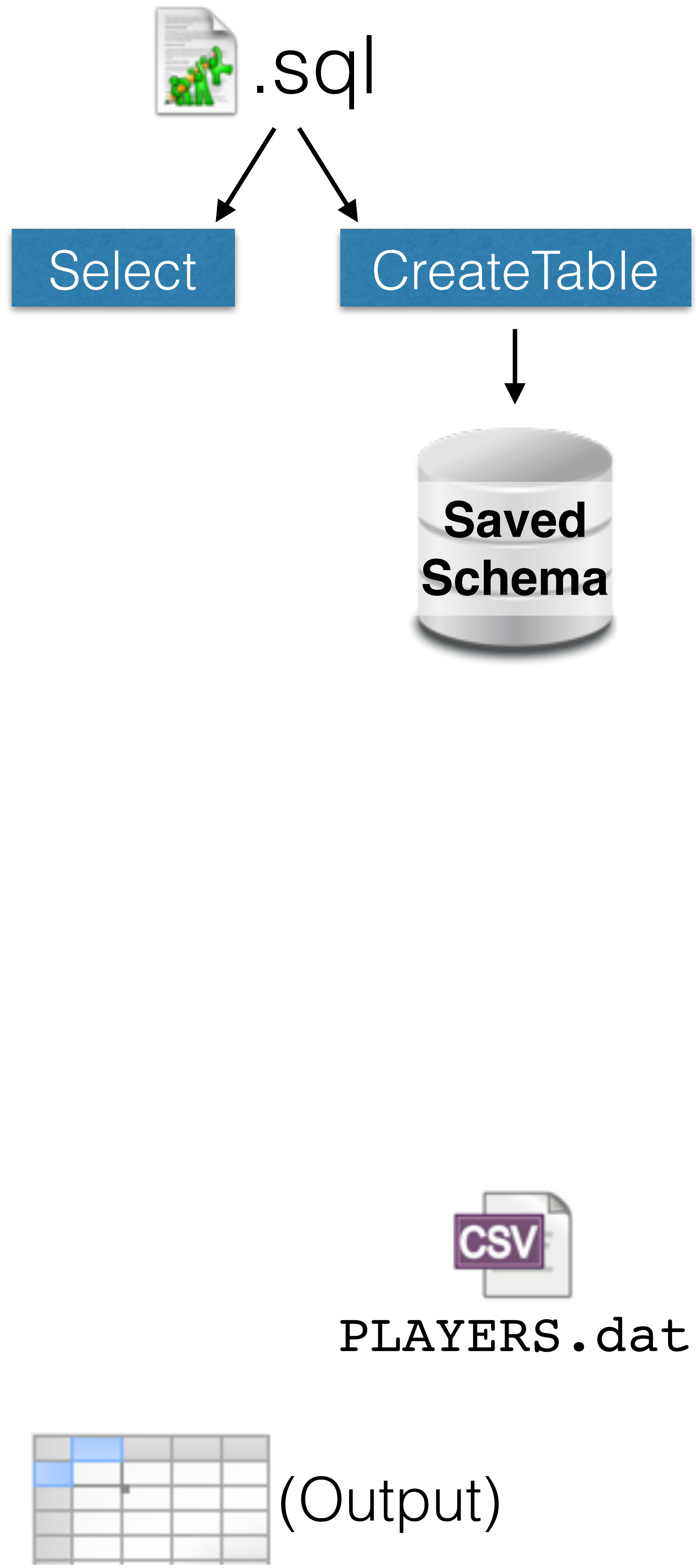
$$\downarrow$$

query = σ(where, **query**)

`body.getSelectItems()`

$$\downarrow$$

query = π(items, **query**)




```sql
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

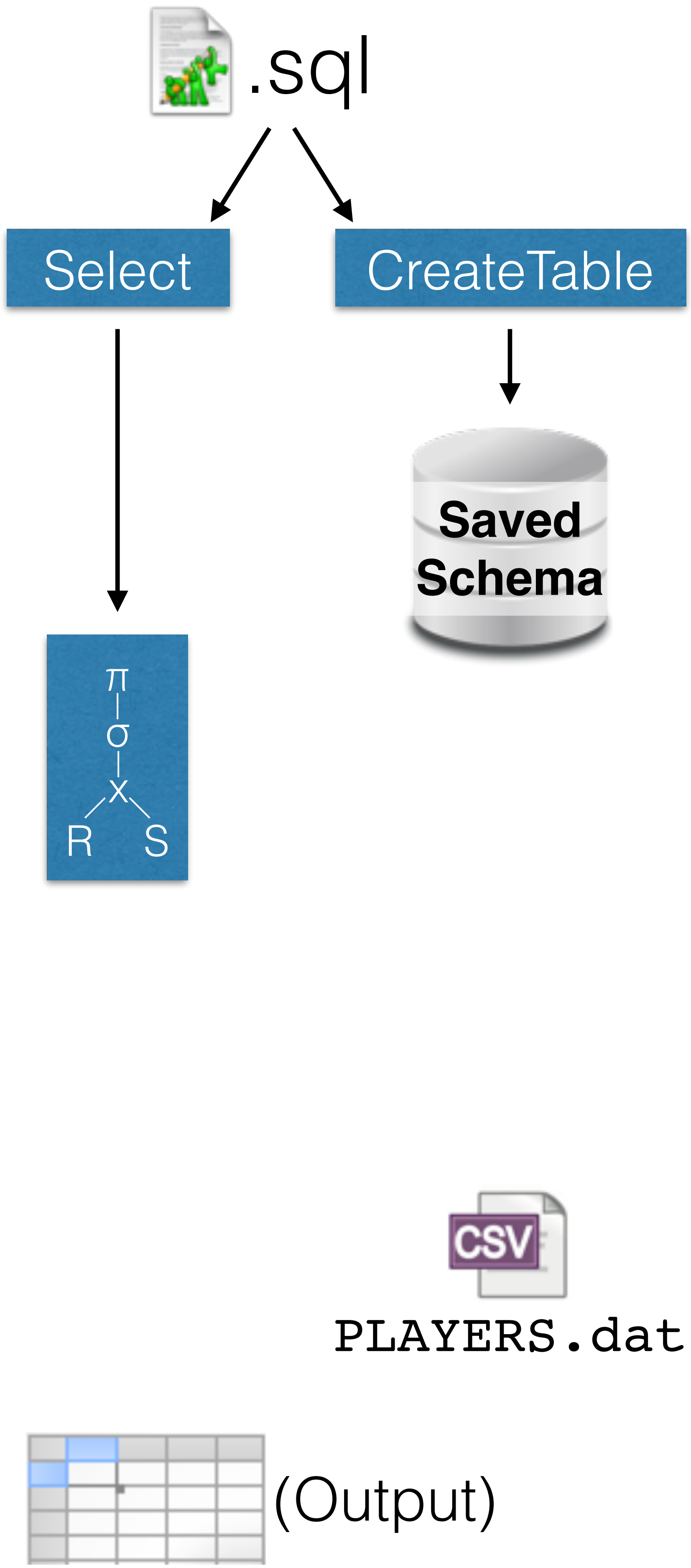
π(items)

|

σ(where)

|

Players




```sql
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

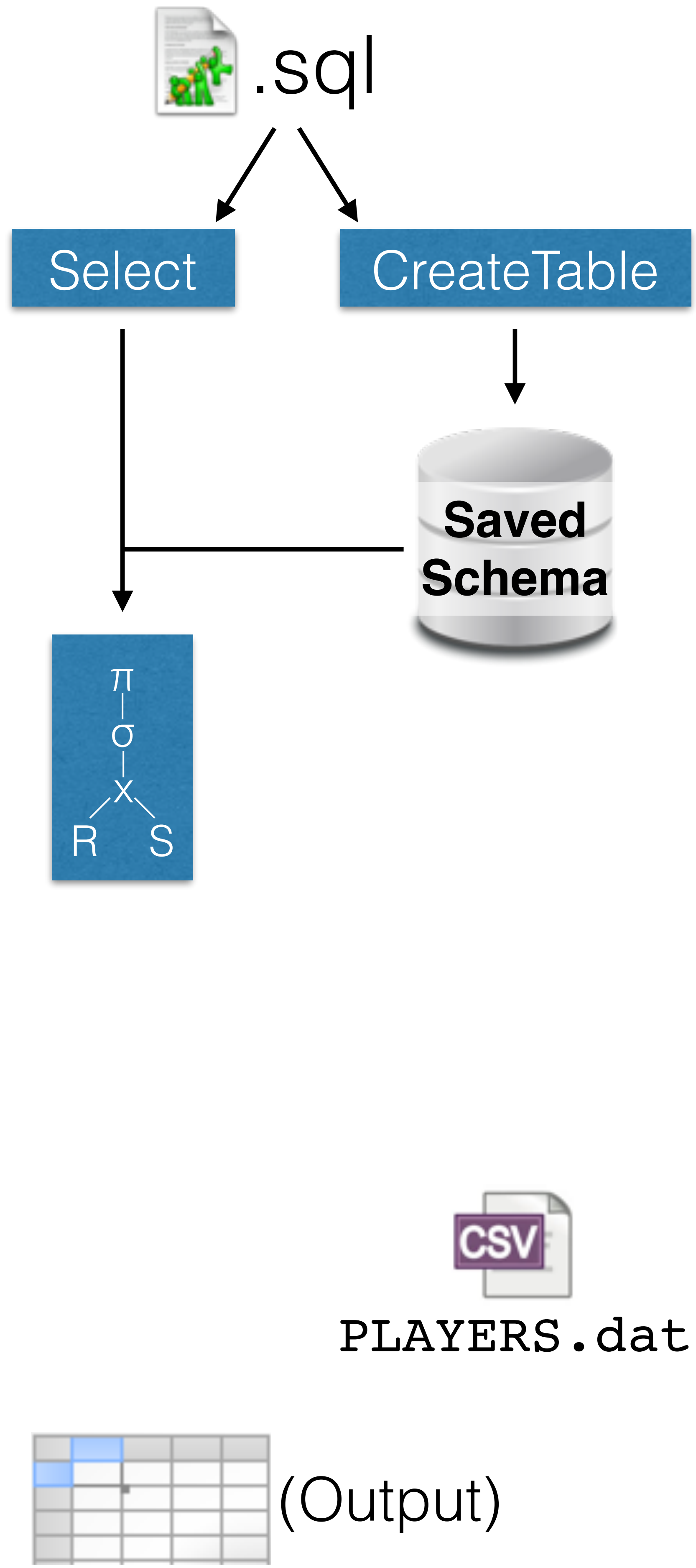
π(items)

|

σ(where)

|

Players




```
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

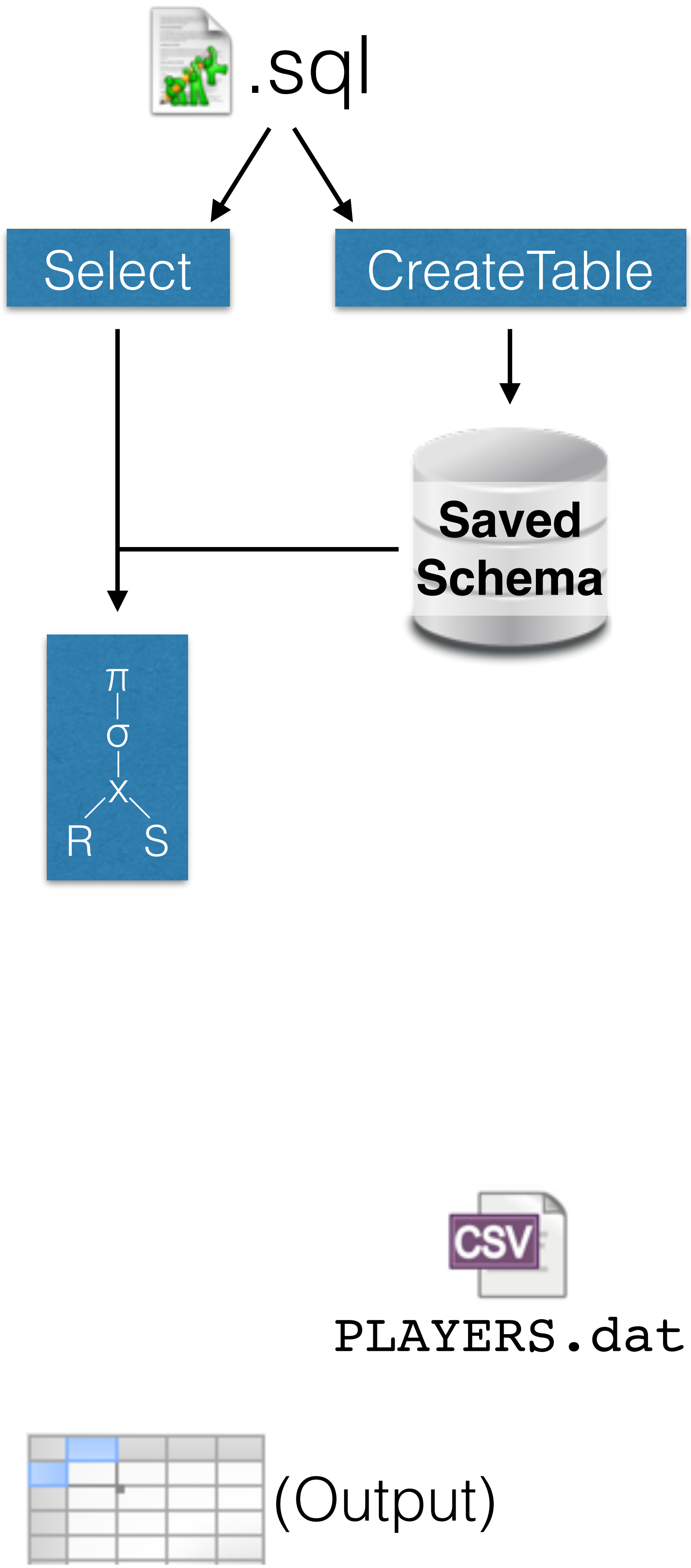
π(items)

|

σ(where)

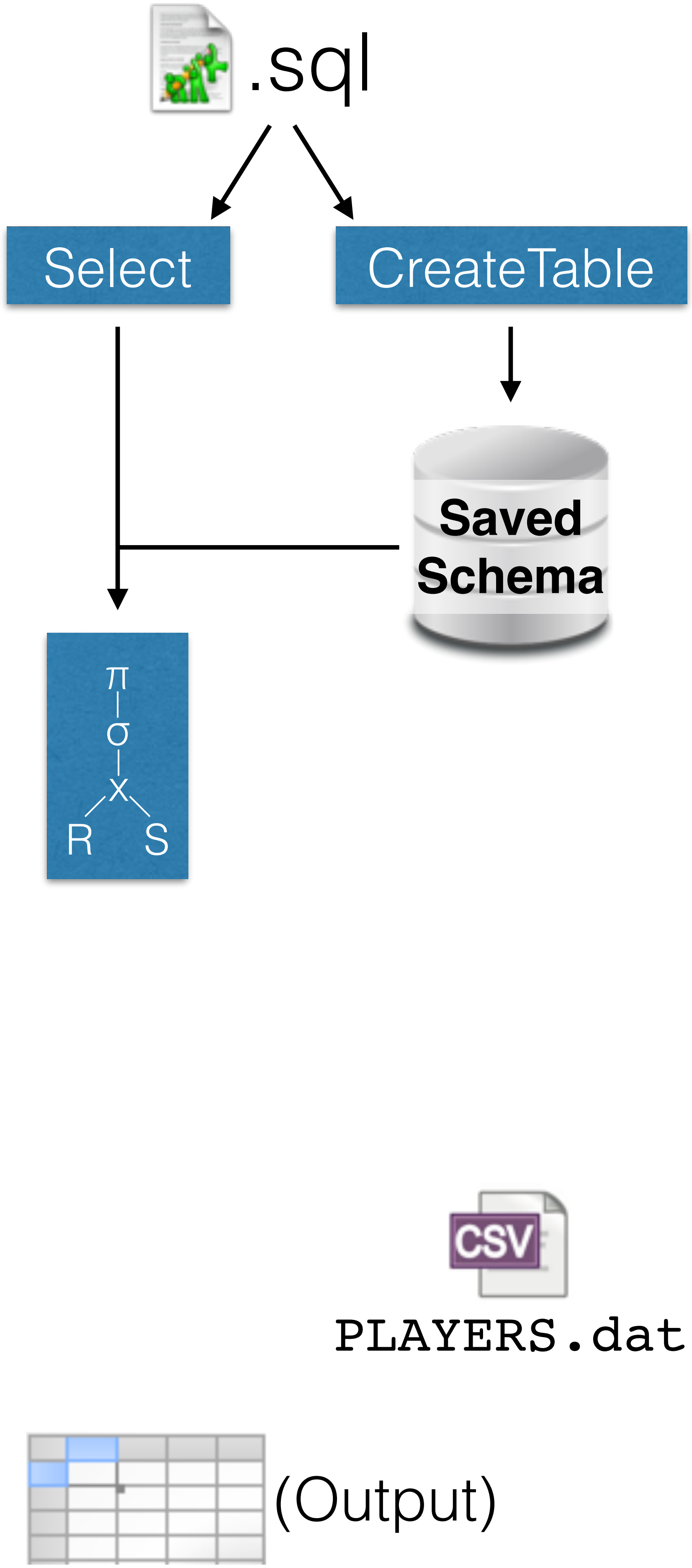
|

Players




```
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

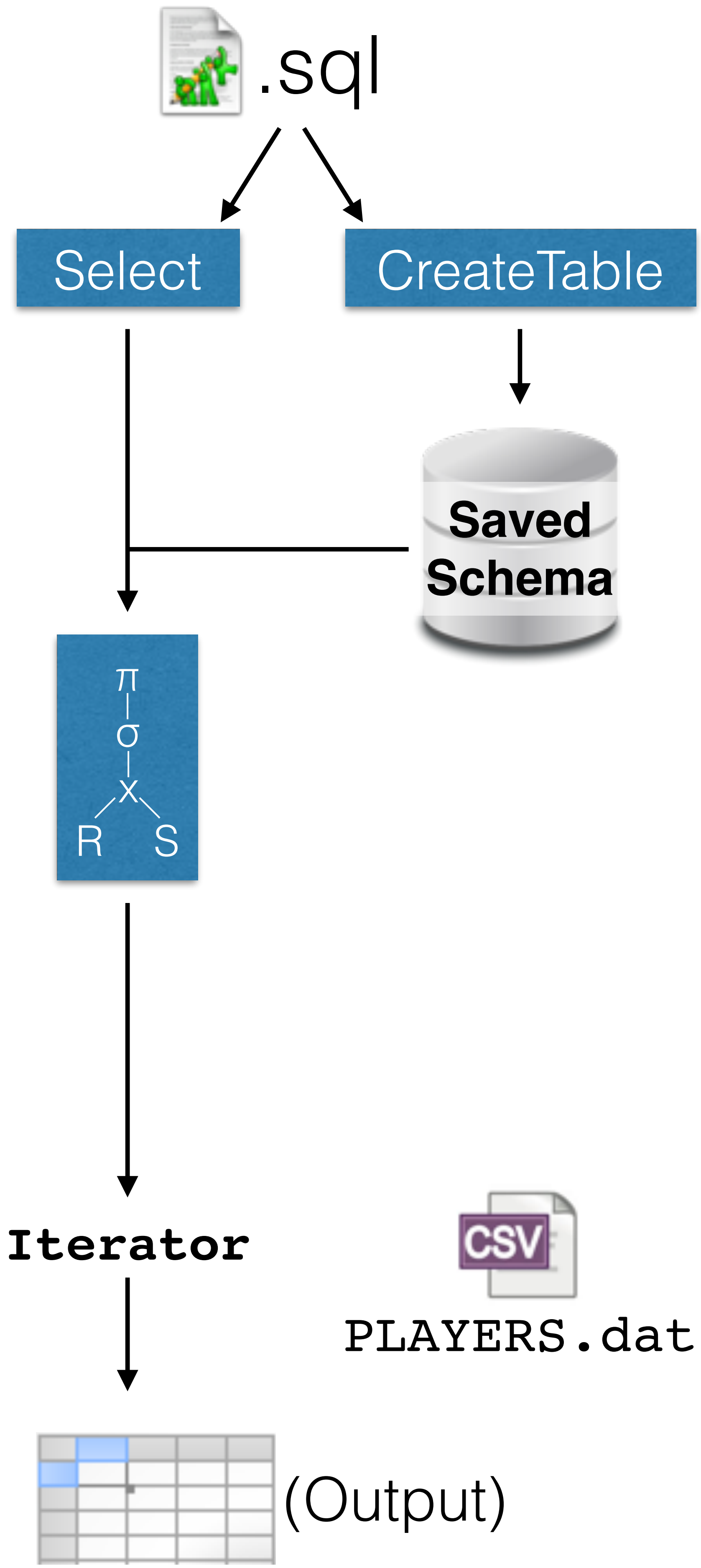
π(items)
|
σ(where)
|
Players




```
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

π(items)
|
σ(where)
|
Players

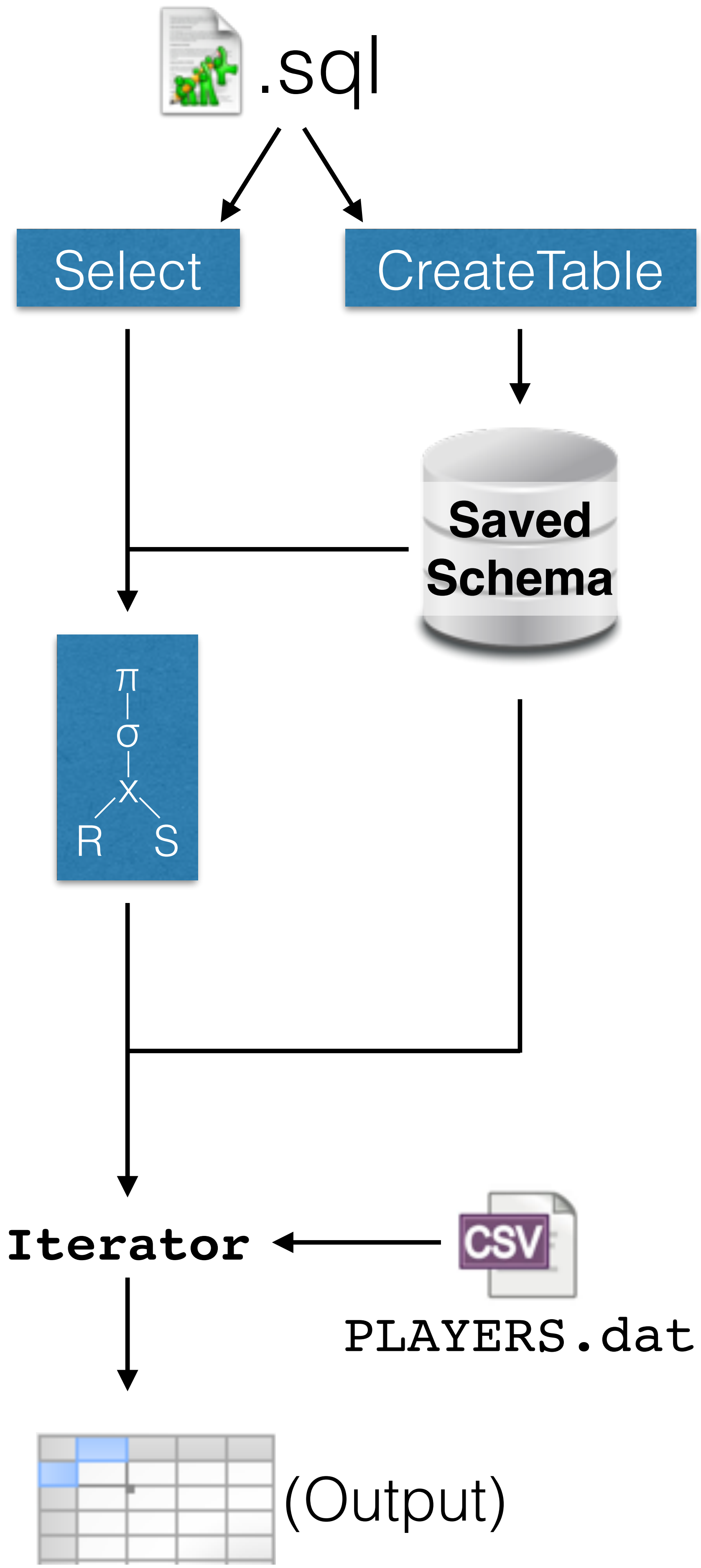
Iterators: Read 1 row at a time




```sql
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

π(items)
|
σ(where)
|
Players

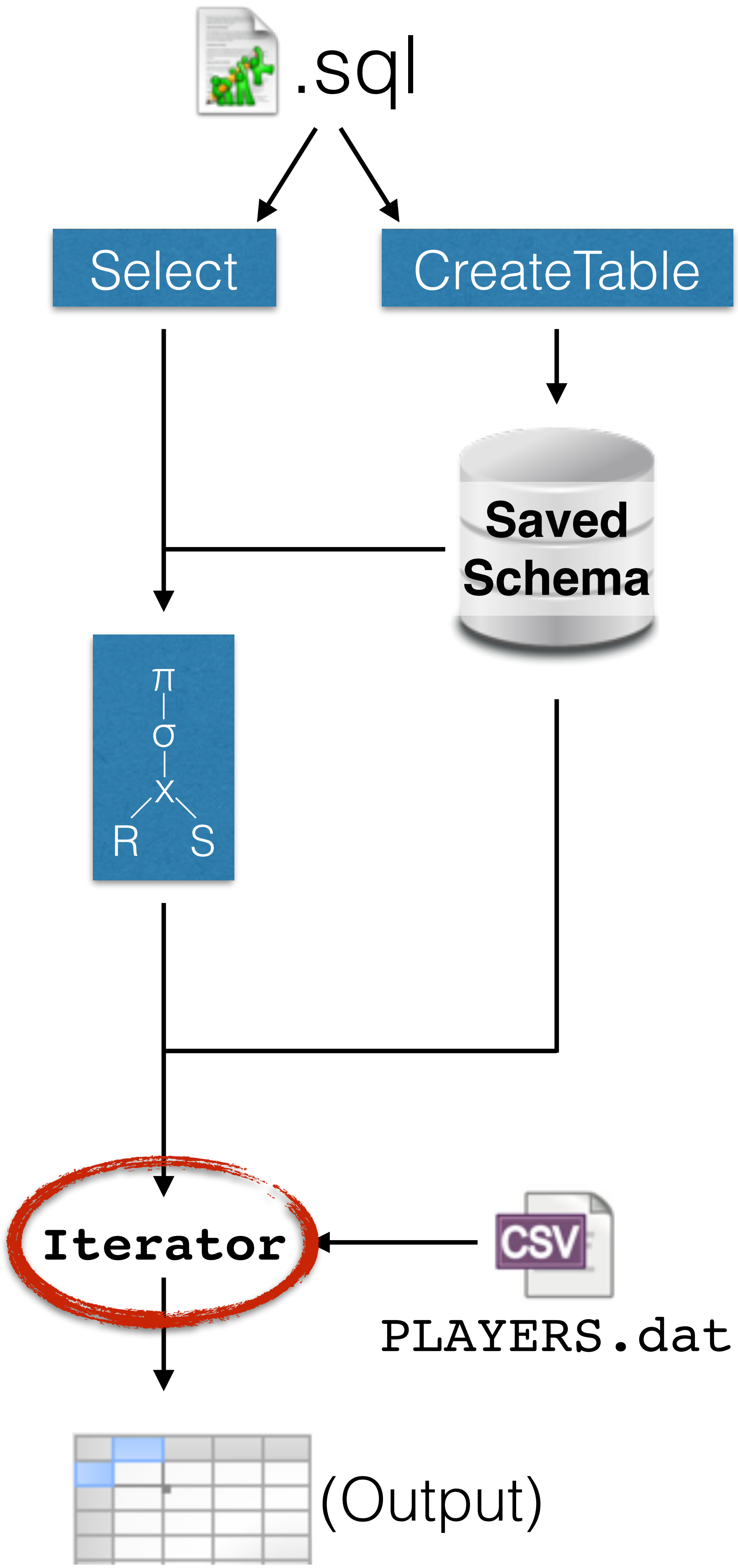
Iterators: Read 1 row at a time




```
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

π(items)
|
σ(where)
|
Players

Iterators: Read 1 row at a time




```
SELECT FIRSTNAME, LASTNAME,
       WEIGHT, BIRTHDATE
FROM PLAYERS
WHERE WEIGHT>200;
```

π(items)
|
σ(where)
|
Players

```
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`

<u>Implementations</u>

BooleanValue
LongValue
DoubleValue
StringValue
DateValue
~~TimestampValue~~
~~TimeValue~~

<u>Methods</u>

```
 double toDouble()
   long toLong()
 String toString()
boolean toBool()
```

`net.sf.jsqlparser.eval.Eval`

`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()`

$\downarrow$

query = R

`body.getJoins()`

`body.getWhere()`

$\downarrow$

query = σ(where, **query**)

`body.getSelectItems()`

$\downarrow$

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()`

↓

query = R

`body.getJoins()`

↓

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

`body.getWhere()`

↓

query = σ(where, **query**)

`body.getSelectItems()`

↓

query = π(items, **query**)

```sql
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.getJoins()`

↓

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

`body.getWhere()`

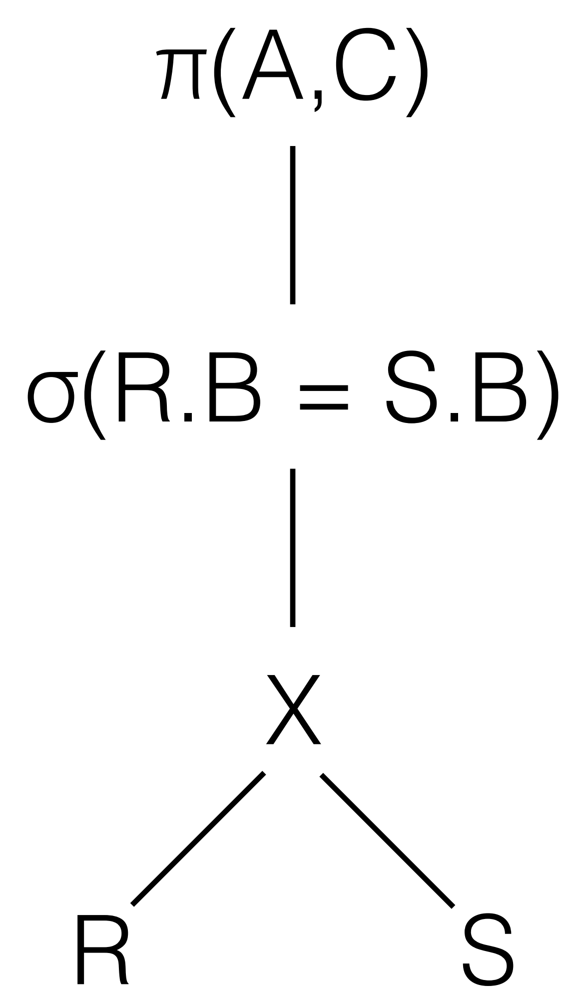
↓

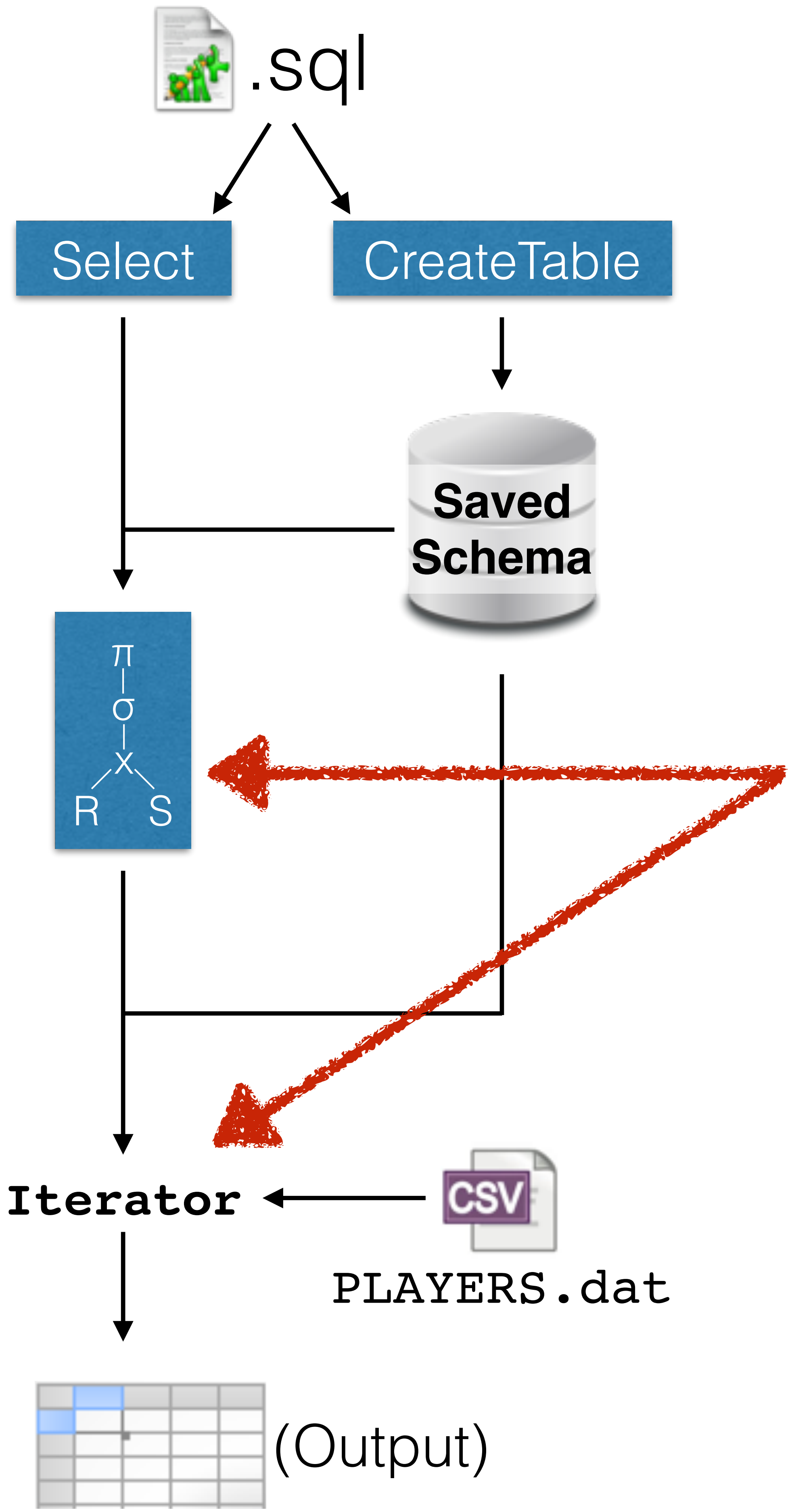
query = σ(where, **query**)

`body.getSelectItems()`

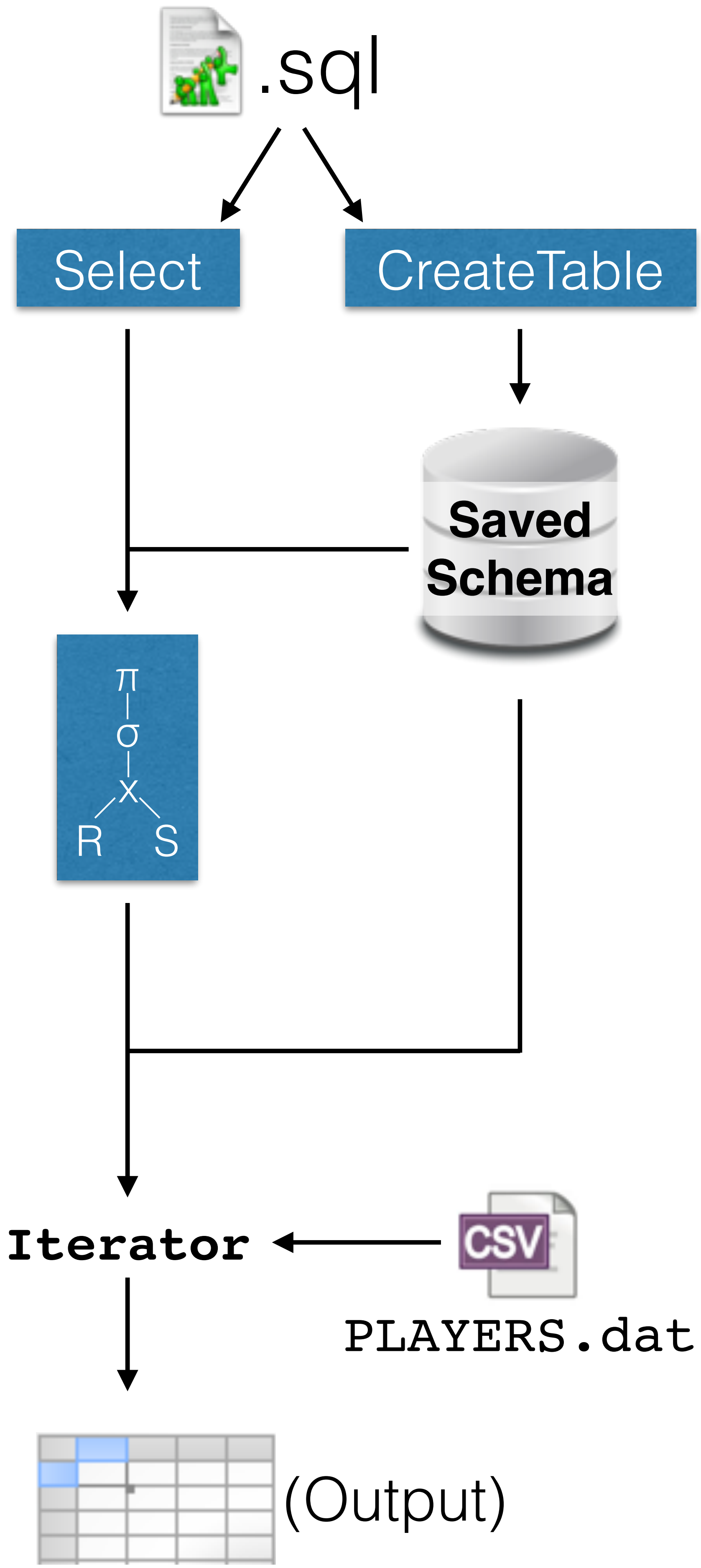
↓

query = π(items, **query**)





Why have both Relational Algebra AND Iterators?



Why have both Relational Algebra AND Iterators?

Not strictly necessary…
… but convenient

Optimizers need…

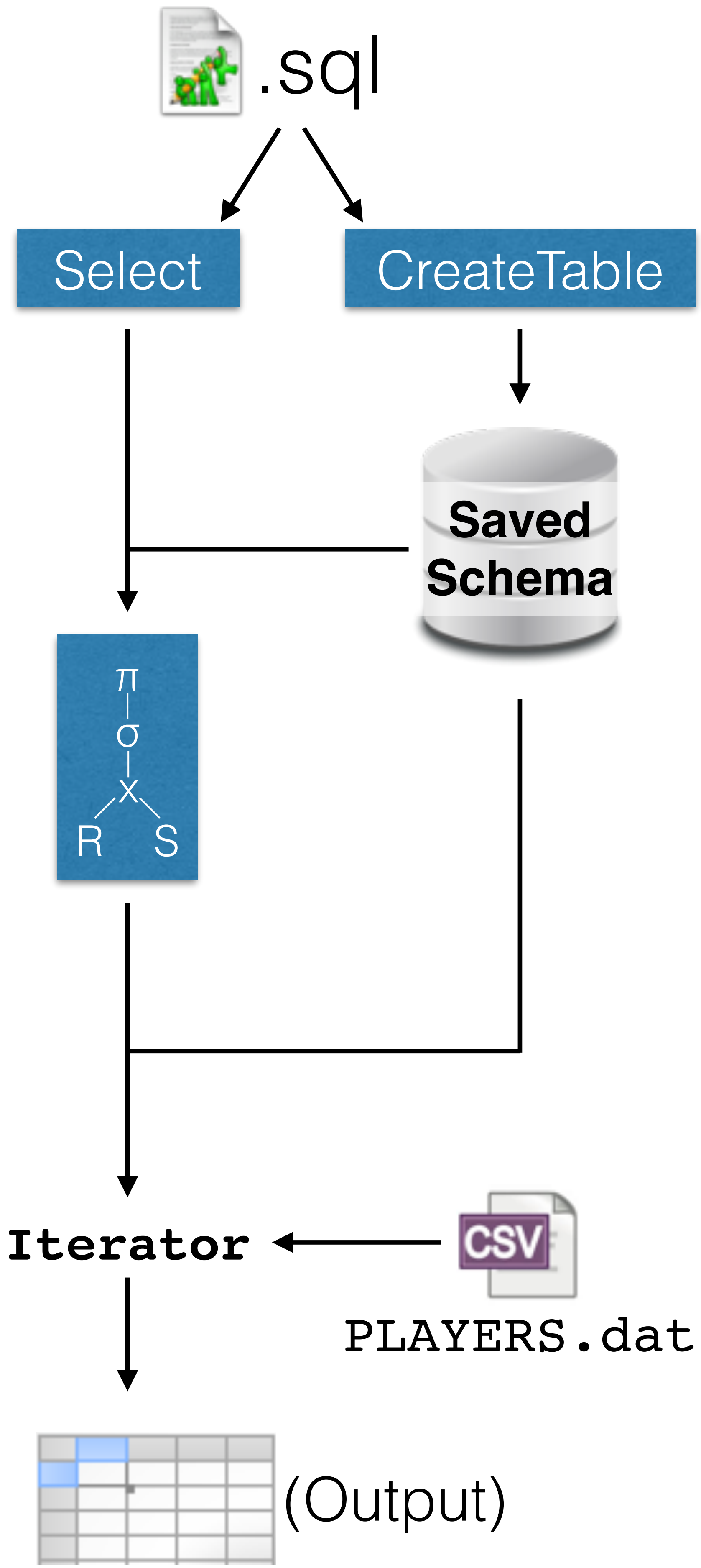
- flexibility
- to easily rearrange subtrees
- evolving schemas

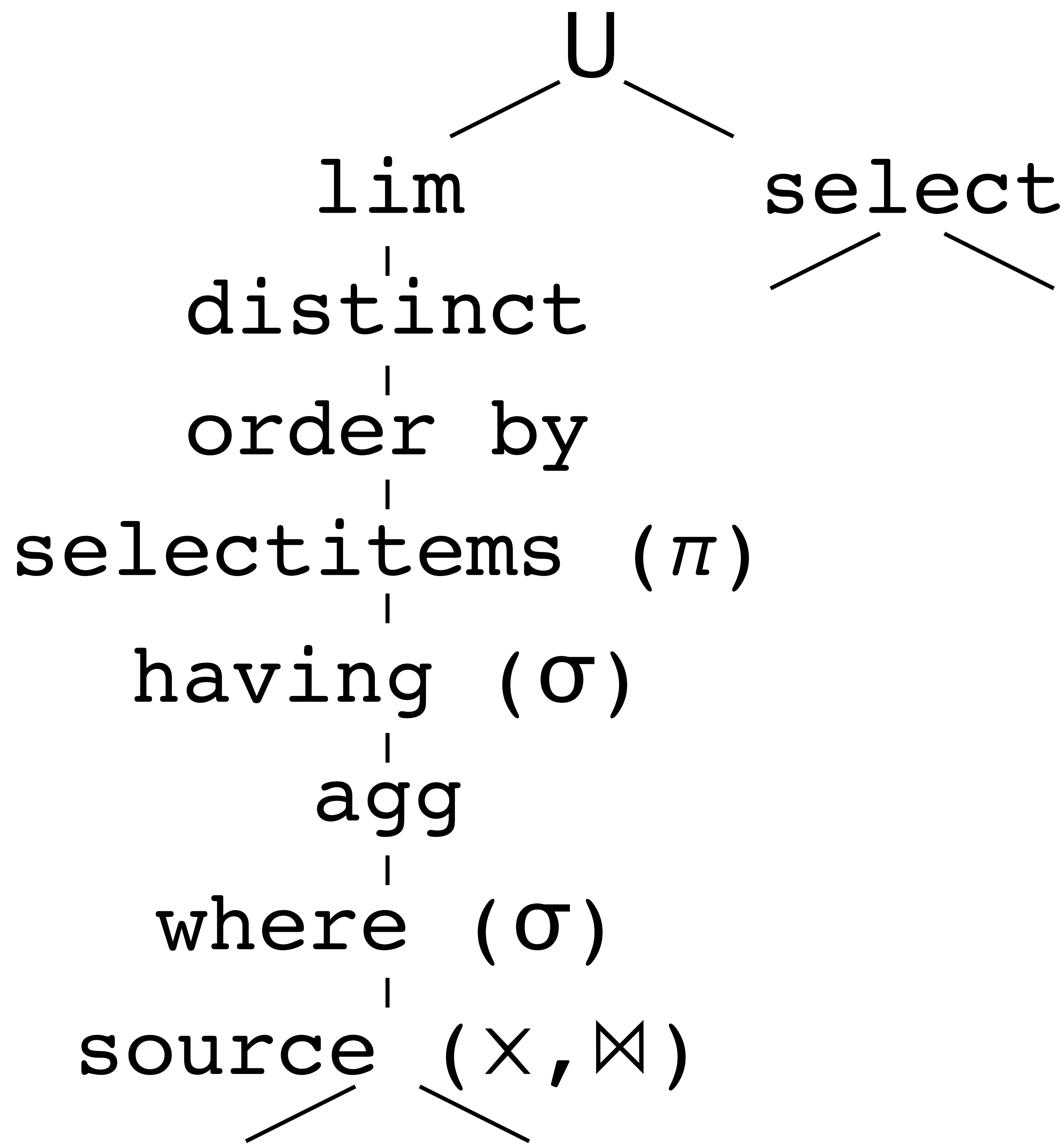
Iterators need…

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

Separate representations
for optimization and execution

Extra Reference Slides


.sql
Select
CreateTable
Saved Schema
π
σ
X
R
S
Iterator
CSV
PLAYERS.dat
(Output)


U
lim
select
distinct
order by
selectitems (π)
having (σ)
agg
where (σ)
source (×,⋈)

Iterators

```
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
}
```