Object Oriented Programming

Lab 5

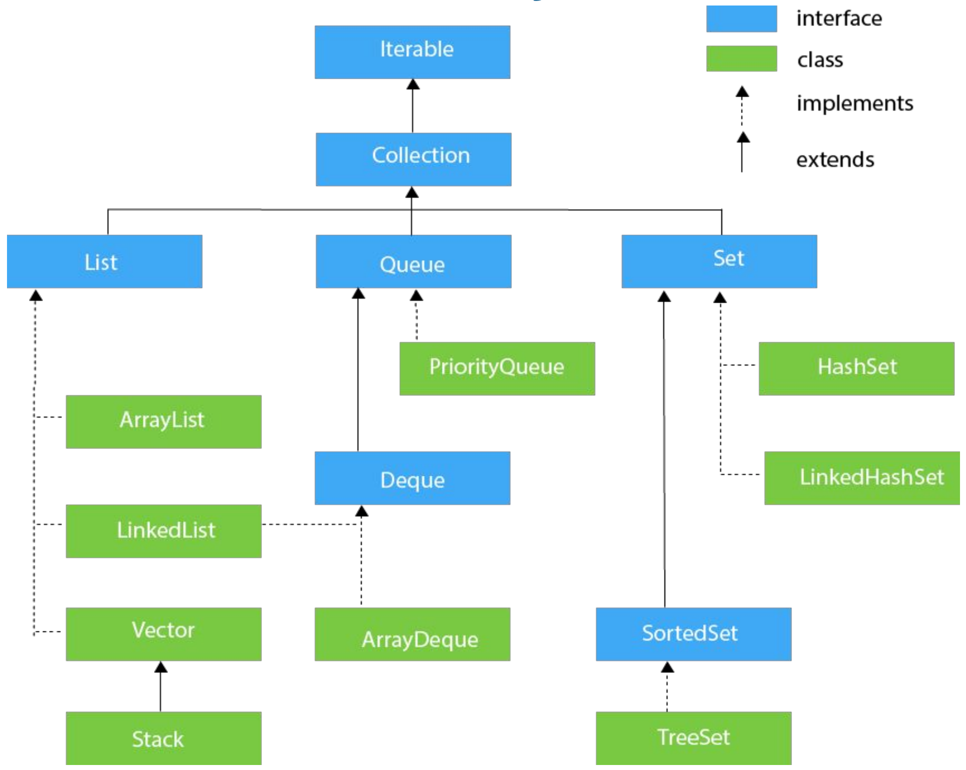
Announcements

- Lab 5 - special due date!
  - Due Tuesday, 02/22
- Project 1: Enigma
  - Checkpoint: 02/25
  - Due: 03/04

The Collection Interface

- Represents any collection of data
  - Most commonly used: `Set` or `List`
- Hierarchy of all Collection classes organized using interfaces
- Collections interface official documentation
- Most collections in `java.util` package
  - `import java.util.ArrayList;`
  - `import java.util.Linkedlist;`

Collections Hierarchy



Source: https://www.javatpoint.com/collections-in-java

The Set Interface

- A group of items with no duplicates
- Common methods supported by Sets:
  - `add(E e)`
  - `remove(Object o)`
  - `contains(Object o)`
  - `isEmpty()`
- Various kinds of set
  - `HashSet`, `TreeSet`, etc.
- Set is an **interface** that **extends** the Collection interface
  - Note: interfaces **extend** other interfaces, while classes **implement** interfaces

```
public interface Set extends Collection {
    ...
}
```

The List Interface

- An **ordered** group of items
- Common methods supported by List:
  - `add(E e)`
  - `add(int index, E e)`
  - `remove(Object o)`
  - `remove(int index)`
  - `contains(Object o)`, etc.
- Various kinds of list
  - `LinkedList`, `ArrayList`, etc.
- Similar to Set, List is an interface that **extends** Collection

```
public interface List extends Collection {
    ...
}
```

Iterators and Iterable

```
public interface Iterable<T> {
  Iterator<T> iterator();

  // some default methods...
}
```

```
for (String value : L) {
   System.out.print(value + " ");
}
```

```
public interface Iterator<E> {
  boolean hasNext();
  E next();
}
```

Iterators and Iterable

```
public class IntList implements Iterable<Integer> {
  // Rest of class not shown
  public Iterator<Integer> iterator() {
    return new IntListIterator();
  }

  class IntListIterator implements Iterator<Integer> {
    boolean hasNext() { /* ... */ }
    Integer next() { /* ... */ }

    IntListIterator() {
      // Often will have a constructor
    }
    // fields!
  }
}
```

From the last slide!

```
public interface Iterable<T> {
  Iterator<T> iterator();

  // some default methods...
}
```

```
public interface Iterator<E> {
  boolean hasNext();
  E next();
}
```

Iterators and Iterable

```
Iterable<Integer> myIterable = // something...
for (Integer i : myIterable) {
  // do stuff
}
```

```
Iterable<Integer> myIterable = // something...
Iterator<Integer> myIterator = myIterable.iterator();
while (myIterator.hasNext()) {
  Integer i = myIterator.next();
}
```

Table Join Demo

Table Join Demo

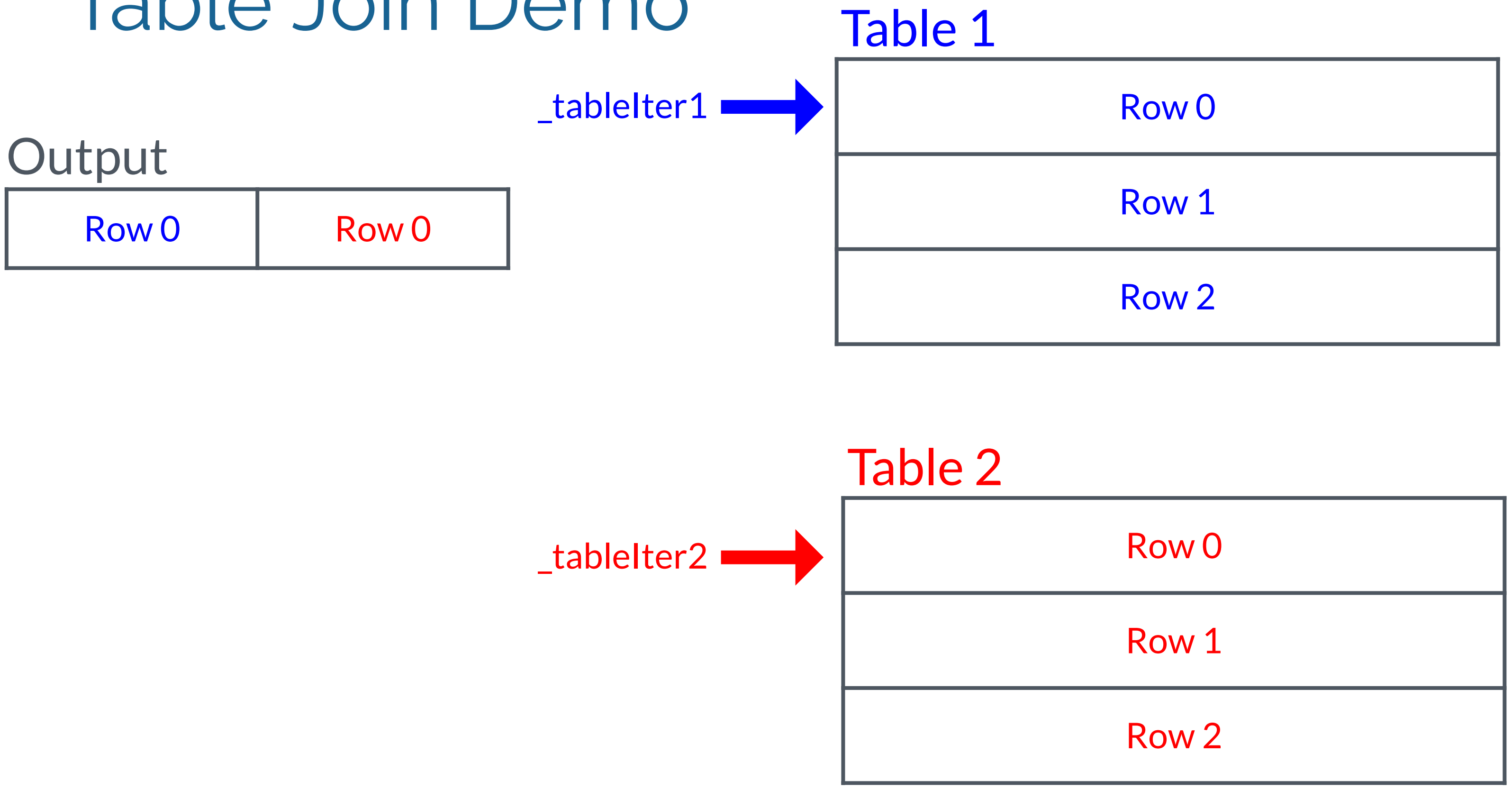


Table Join Demo

Output

Row 0	Row 0
Row 0	Row 1

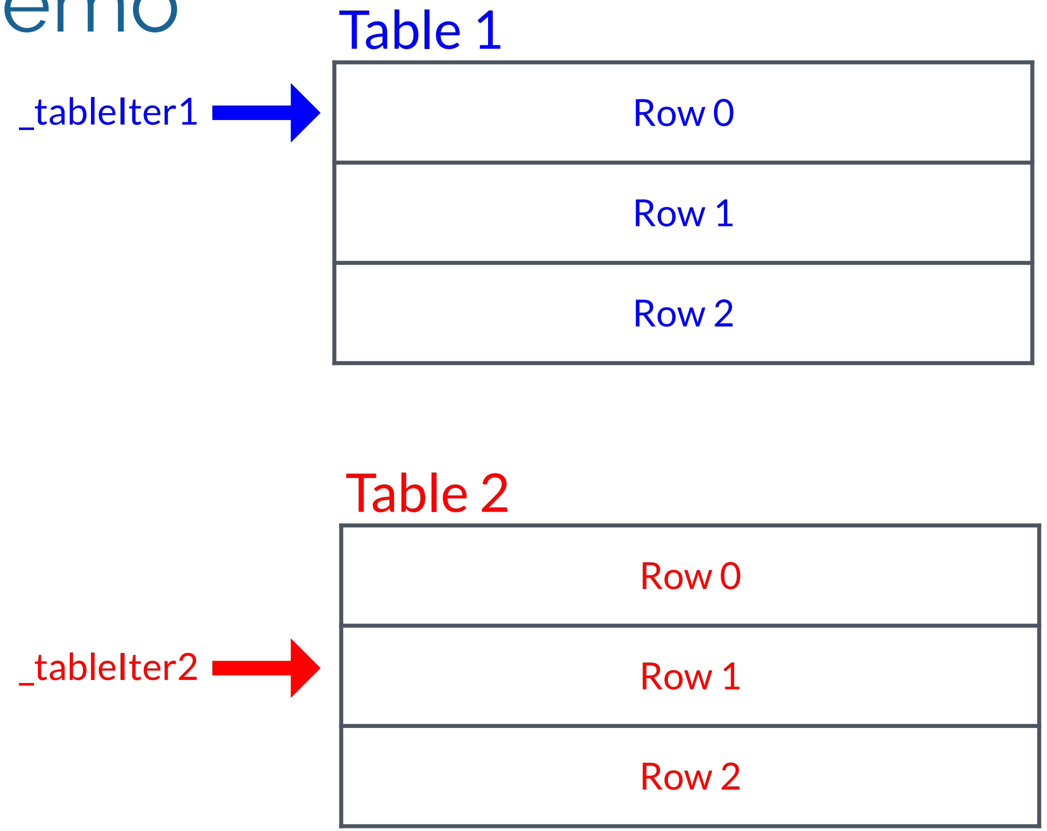


Table Join Demo

Output

Row 0	Row 0
Row 0	Row 1
Row 0	Row 2

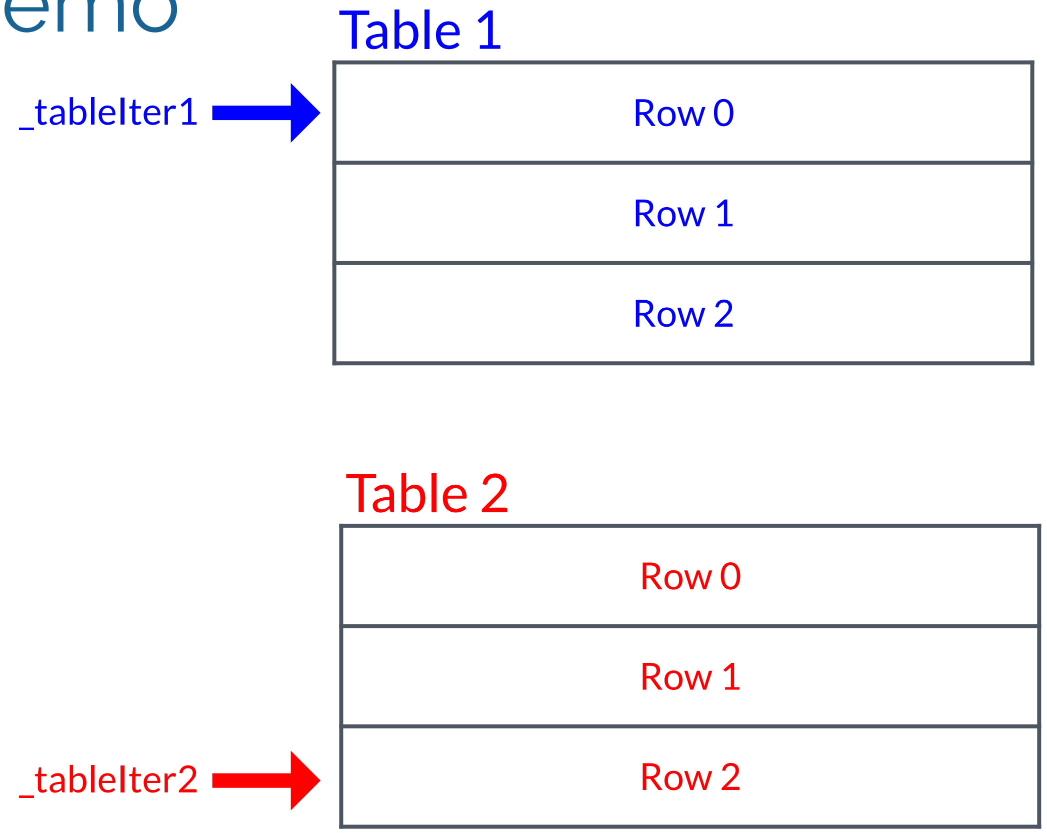


Table Join Demo

Output

Row 0	Row 0
Row 0	Row 1
Row 0	Row 2
Row 1	Row 0

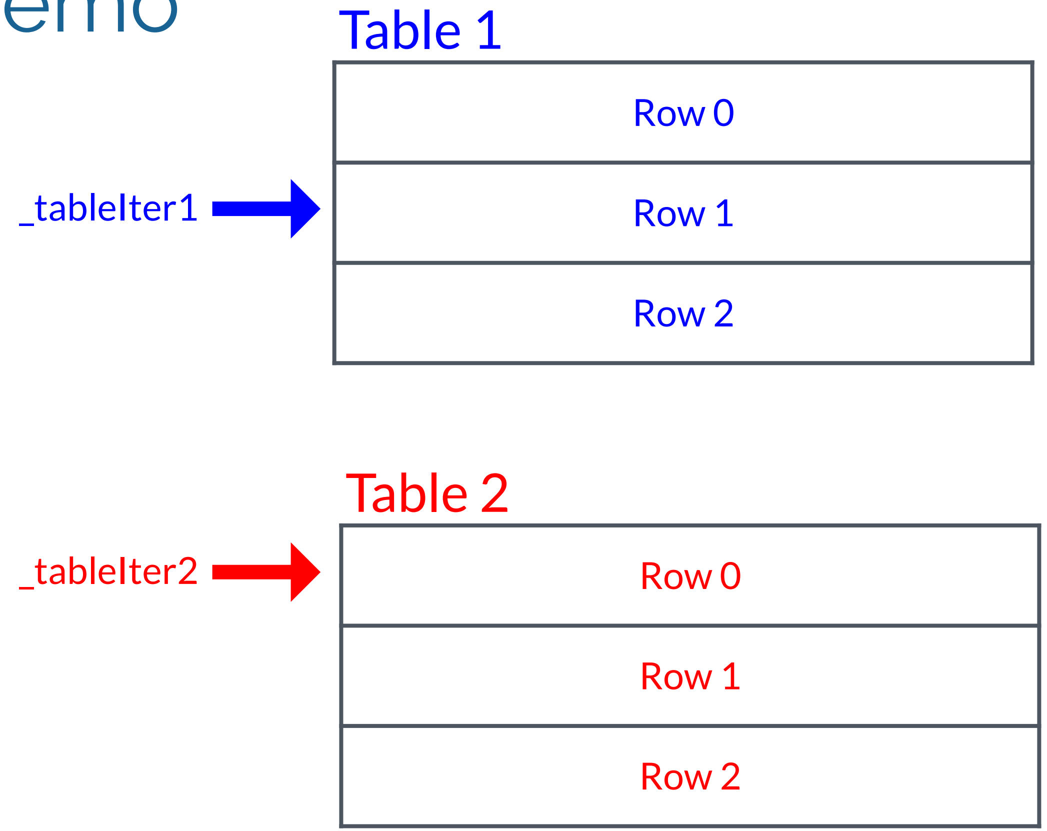


Table 1

Row 0
Row 1
Row 2

Table 2

Row 0
Row 1
Row 2

Table Join Demo

Output

Row 0	Row 0
Row 0	Row 1
Row 0	Row 2
Row 1	Row 0
Row 1	Row 1

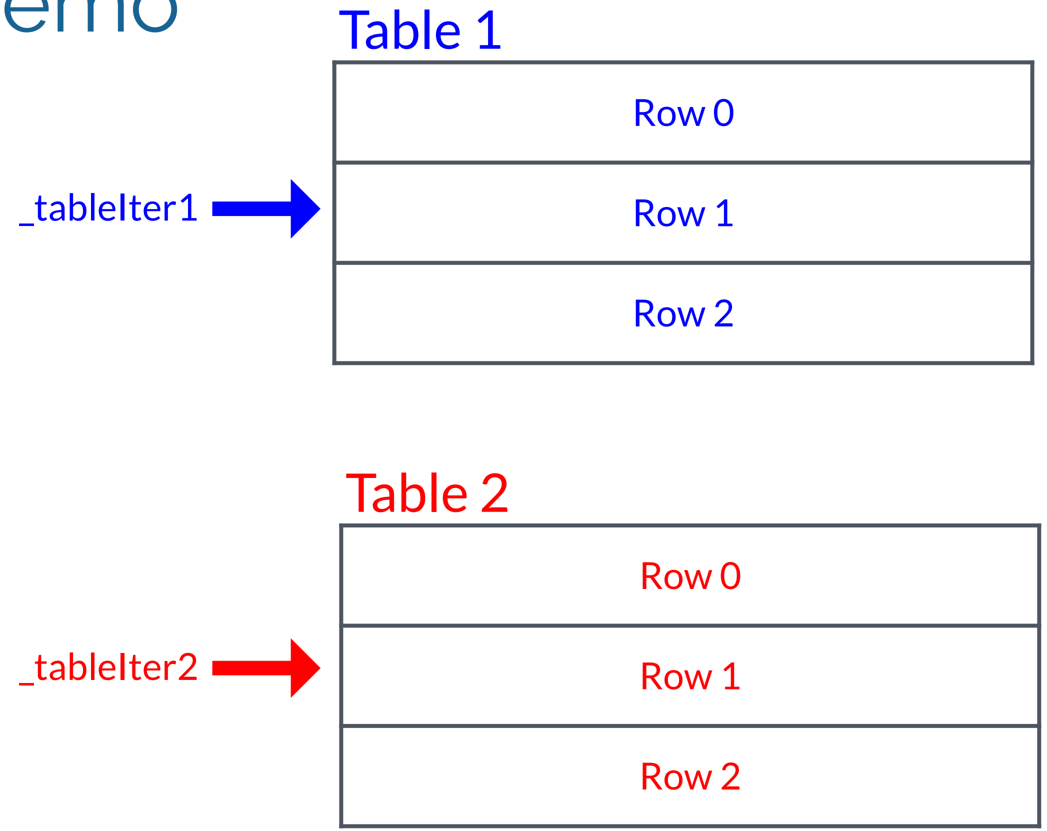


Table 1

Row 0
Row 1
Row 2

Table 2

Row 0
Row 1
Row 2

Table Join Demo

Output

Row 0	Row 0
Row 0	Row 1
Row 0	Row 2
Row 1	Row 0
Row 1	Row 1
Row 1	Row 2

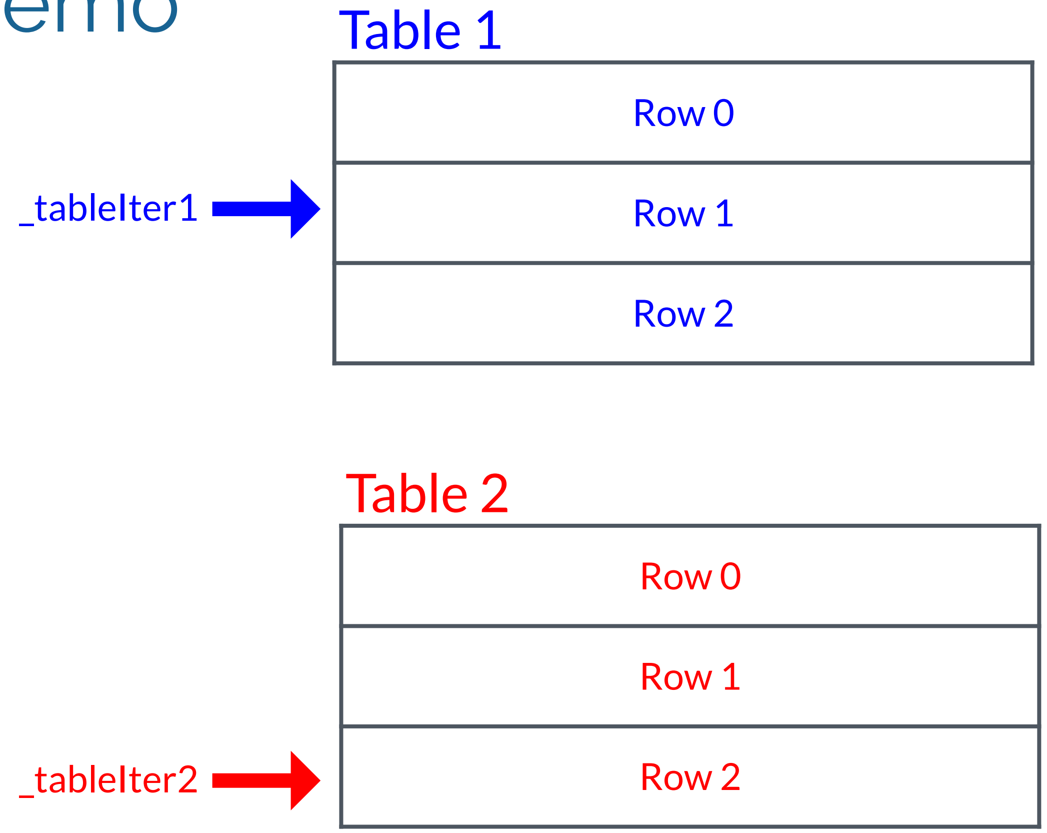


Table 1

Row 0
Row 1
Row 2

Table 2

Row 0
Row 1
Row 2

Table Join Demo

Output

Row 0	Row 0
Row 0	Row 1
Row 0	Row 2
Row 1	Row 0
Row 1	Row 1
Row 1	Row 2
Row 2	Row 0

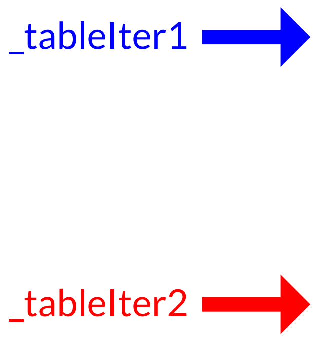


Table 1

Row 0
Row 1
Row 2

Table 2

Row 0
Row 1
Row 2

Table Join Demo

Output

Row 0	Row 0
Row 0	Row 1
Row 0	Row 2
Row 1	Row 0
Row 1	Row 1
Row 1	Row 2
Row 2	Row 0
Row 2	Row 1

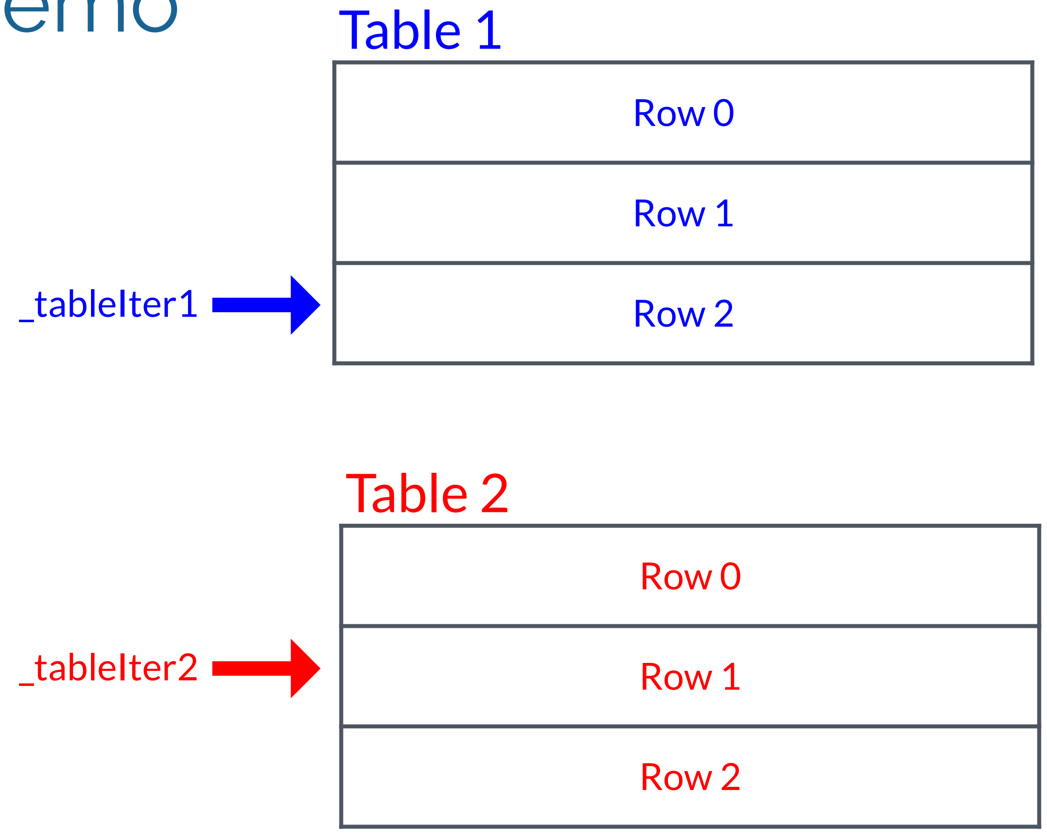


Table 1

Row 0
Row 1
Row 2

Table 2

Row 0
Row 1
Row 2

Table Join Demo

Output

Row 0	Row 0
Row 0	Row 1
Row 0	Row 2
Row 1	Row 0
Row 1	Row 1
Row 1	Row 2
Row 2	Row 0
Row 2	Row 1
Row 2	Row 2

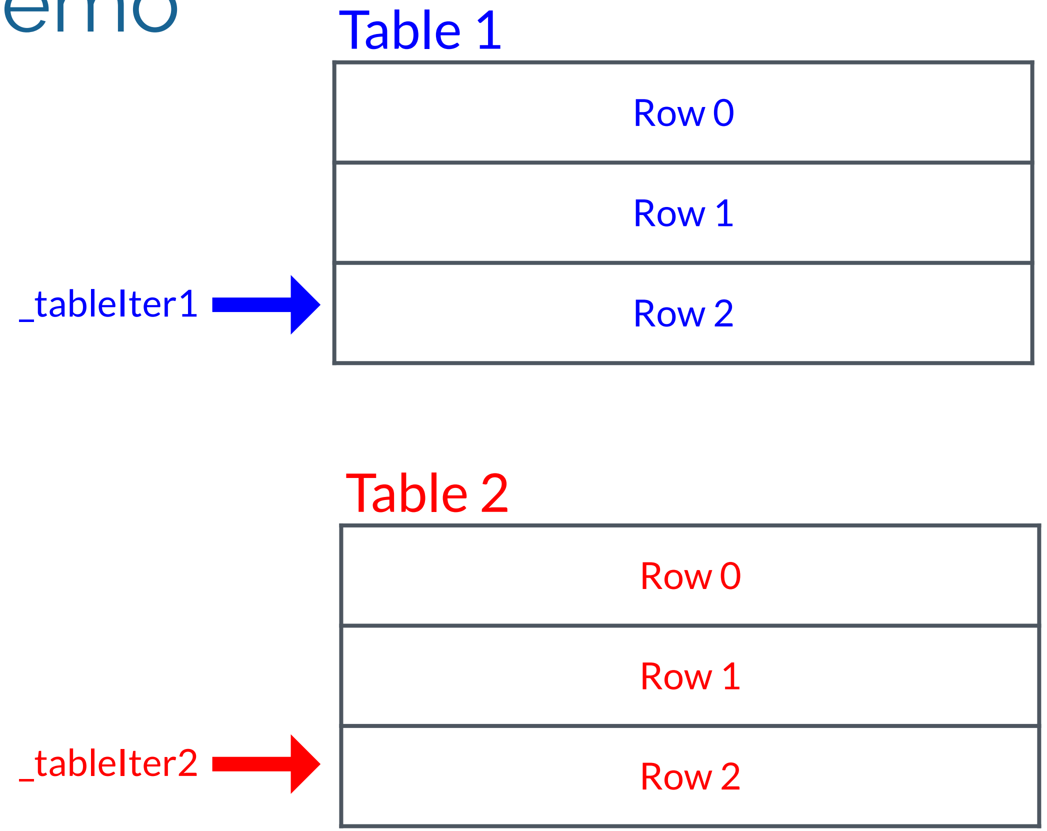


Table 1

Row 0
Row 1
Row 2

Table 2

Row 0
Row 1
Row 2