# Using Hive

```
import org.apache.spark.sql.hive.HiveContext
val sqlContext = new HiveContext(sc)
sqlContext.sql("set spark.sql.shuffle.partitions=10");

val joinAggData = sqlContext.sql("select o.order_date,
round(sum(oi.order_item_subtotal), 2), count(distinct o.order_id) from orders o
join order_items oi on o.order_id = oi.order_item_order_id group by o.order_date
order by o.order_date")

joinAggData.collect().foreach(println)
```

# Using spark native sql

```
import org.apache.spark.sql.SQLContext, org.apache.spark.sql.Row
val sqlContext = new SQLContext(sc)
sqlContext.sql("set spark.sql.shuffle.partitions=10");

val ordersRDD = sc.textFile("/user/cloudera/sqoop_import/orders")
val ordersMap = ordersRDD.map(o => o.split(","))

case class Orders(order_id: Int, order_date: String, order_customer_id: Int,
order_status: String)
val orders = ordersMap.map(o => Orders(o(0).toInt, o(1), o(2).toInt, o(3)))

import sqlContext.createSchemaRDD
orders.registerTempTable("orders")

val orderItemsRDD = sc.textFile("/user/cloudera/sqoop_import/order_items")
val orderItemsMap = orderItemsRDD.map(oi => oi.split(","))

case class OrderItems
  (order_item_id: Int,
   order_item_order_id: Int,
   order_item_product_id: Int,
   order_item_quantity: Int,
   order_item_subtotal: Float,
   order_item_product_price: Float
  )

val orderItems = sc.textFile("/user/cloudera/sqoop_import/order_items").
  map(rec => rec.split(",")).
  map(oi => OrderItems(oi(0).toInt, oi(1).toInt, oi(2).toInt, oi(3).toInt,
oi(4).toFloat, oi(5).toFloat))

orderItems.registerTempTable("order_items")

val joinAggData = sqlContext.sql("select o.order_date,
sum(oi.order_item_subtotal), " +
  "count(distinct o.order_id) from orders o join order_items oi " +
  "on o.order_id = oi.order_item_order_id " +
  "group by o.order_date order by o.order_date")

joinAggData.collect().foreach(println)
```
===

# Using Hive Context

```
import org.apache.spark.sql.hive.HiveContext
val hiveContext = new HiveContext(sc)
hiveContext.sql("set spark.sql.shuffle.partitions=10");

hiveContext.sql("select o.order_date, sum(oi.order_item_subtotal)/count(distinct
oi.order_item_order_id) from orders o join order_items oi on o.order_id =
oi.order_item_order_id group by o.order_date order by
o.order_date").collect().foreach(println)
```

# This query works in hive

```
select * from (select q.order_date, q.order_customer_id, q.order_item_subtotal,
max(q.order_item_subtotal) over (partition by q.order_date)
max_order_item_subtotal
from (select o.order_date, o.order_customer_id, sum(oi.order_item_subtotal)
order_item_subtotal
from orders o join order_items oi
on o.order_id = oi.order_item_order_id
group by o.order_date, o.order_customer_id) q) s
where s.order_item_subtotal = s.max_order_item_subtotal
```

```
order by s.order_date;

select * from (
select o.order_date, o.order_customer_id, sum(oi.order_item_subtotal)
order_item_subtotal
from orders o join order_items oi
on o.order_id = oi.order_item_order_id
group by o.order_date, o.order_customer_id) q1
join
(select q.order_date, max(q.order_item_subtotal) order_item_subtotal
from (select o.order_date, o.order_customer_id, sum(oi.order_item_subtotal)
order_item_subtotal
from orders o join order_items oi
on o.order_id = oi.order_item_order_id
group by o.order_date, o.order_customer_id) q
group by q.order_date) q2
on q1.order_date = q2.order_date and q1.order_item_subtotal =
q2.order_item_subtotal
order by q1.order_date;

#####################################################################
```

# **Using data frames (only works with spark 1.3.0 or later**)

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext.implicits._
sqlContext.sql("set spark.sql.shuffle.partitions=10");

case class Orders(orderId: Int, orderDate: String, orderCustomerId: Int,
orderStatus: String)

val orders = sc.textFile("/user/cloudera/sqoop_import/orders").
  map(rec => rec.split(",")).
  map(o => Orders(o(0).toInt, o(1), o(2).toInt, o(3))).toDF()

orders.registerTempTable("orders")
val ordersData = sqlContext.sql("select * from orders")
ordersData.collect().take(10).foreach(println)
```

```
case class OrderItems
  (orderItemId: Int,
   orderItemOrderId: Int,
   orderItemProductId: Int,
   orderItemQuantity: Int,
   orderItemSubtotal: Float,
   orderItemProductPrice: Float
  )

val orderItems = sc.textFile("/user/cloudera/sqoop_import/order_items").
  map(rec => rec.split(",")).
  map(oi => OrderItems(oi(0).toInt, oi(1).toInt, oi(2).toInt, oi(3).toInt,
oi(4).toFloat, oi(5).toFloat)).
  toDF()

orderItems.registerTempTable("orderItems")
val orderItemsData = sqlContext.sql("select * from orderItems")
orderItemsData.collect().take(10).foreach(println)

val ordersJoinOrderItems = sqlContext.sql("select o.orderDate,
sum(oi.orderItemSubtotal), count(o.orderId) from orders o join orderItems oi on
o.orderId = oi.orderItemOrderId group by o.orderDate")
val ordersJoinOrderItems = sqlContext.sql("select o.orderDate,
sum(oi.orderItemSubtotal), count(distinct o.orderId),
sum(oi.orderItemSubtotal)/count(distinct o.orderId) from orders o join orderItems
oi on o.orderId = oi.orderItemOrderId group by o.orderDate order by o.orderDate")
ordersJoinOrderItems.collect().take(10).foreach(println)
```
==

Sorting using queries

--

Global sorting and ranking

==

```
select * from products order by product_price desc;
select * from products order by product_price desc limit 10;
```

#By key sorting
#Using order by is not efficient, it serializes

```
select * from products order by product_category_id, product_price desc;
```

#Using distribute by sort by (to distribute sorting and scale it up)

```
select * from products distribute by product_category_id sort by product_price
desc;
```

#By key ranking (in Hive we can use windowing/analytic functions)

```
select * from (select p.*,
dense_rank() over (partition by product_category_id order by product_price desc)
dr
from products p
distribute by product_category_id) q
where dr <= 2 order by product_category_id, dr;
```