VIEWS

VIEWS

LET'S SAY THERE IS A REALLY COMPLICATED QUERY YOU USE AGAIN AND AGAIN

YOU COULD SAVE THE QUERY AS A VIEW

VIEWS

YOU COULD THEN QUERY THAT VIEW
EXACTLY AS IF IT WERE A TABLE

RATHER THAN HAVE IT AS A COMPLICATED
SUB-QUERY IN YOUR OTHER QUERIES

HERE IS A PRETTY COMPLEX QUERY

```
SELECT PRODUCTNAME,REVENUE FROM(
SELECT
P.PRODUCTNAME,YEAR(ORDERDATE) AS ORDERDATE,SUM(REVENUE) AS REVENUE
FROM
(SELECT * FROM SALES_DATA_NEW  WHERE PRODUCTID IN (SELECT PRODUCTID FROM
TOP_SELLERS)) S
INNER JOIN
PRODUCTS P
ON
S.PRODUCTID = P.PRODUCTID
WHERE
(YEAR(S.ORDERDATE) IN (SELECT MAX(YEAR(ORDERDATE)) FROM SALES_DATA_NEW ))
GROUP BY
P.PRODUCTNAME, YEAR(ORDERDATE))
ORDER BY REVENUE DESC LIMIT 10;
```

THIS ENTIRE SUBQUERY CAN BE SAVED AS A VIEW

HERE IS A PRETTY COMPLEX QUERY

```
SELECT PRODUCTNAME,REVENUE FROM
Top_Selling_Products
ORDER BY REVENUE DESC LIMIT 10;
```

WE HAVE DRAMATICALLY REDUCED THE QUERY COMPLEXITY

YOU CAN CREATE A VIEW JUST AS YOU WOULD CREATE A TABLE FROM A SUBQUERY

```sql
CREATE VIEW Top_Selling_Products AS
SELECT
P.PRODUCTNAME,YEAR(ORDERDATE),SUM(REVENUE)
FROM
(SELECT * FROM SALES_DATA_NEW  WHERE PRODUCTID IN (SELECT PRODUCTID FROM
TOP_SELLERS)) S
INNER JOIN
PRODUCTS P
ON
S.PRODUCTID = P.PRODUCTID
WHERE
(YEAR(S.ORDERDATE) IN (SELECT MAX(YEAR(ORDERDATE)) FROM SALES_DATA_NEW ))
GROUP BY
P.PRODUCTNAME, YEAR(ORDERDATE);
```

VIEWS

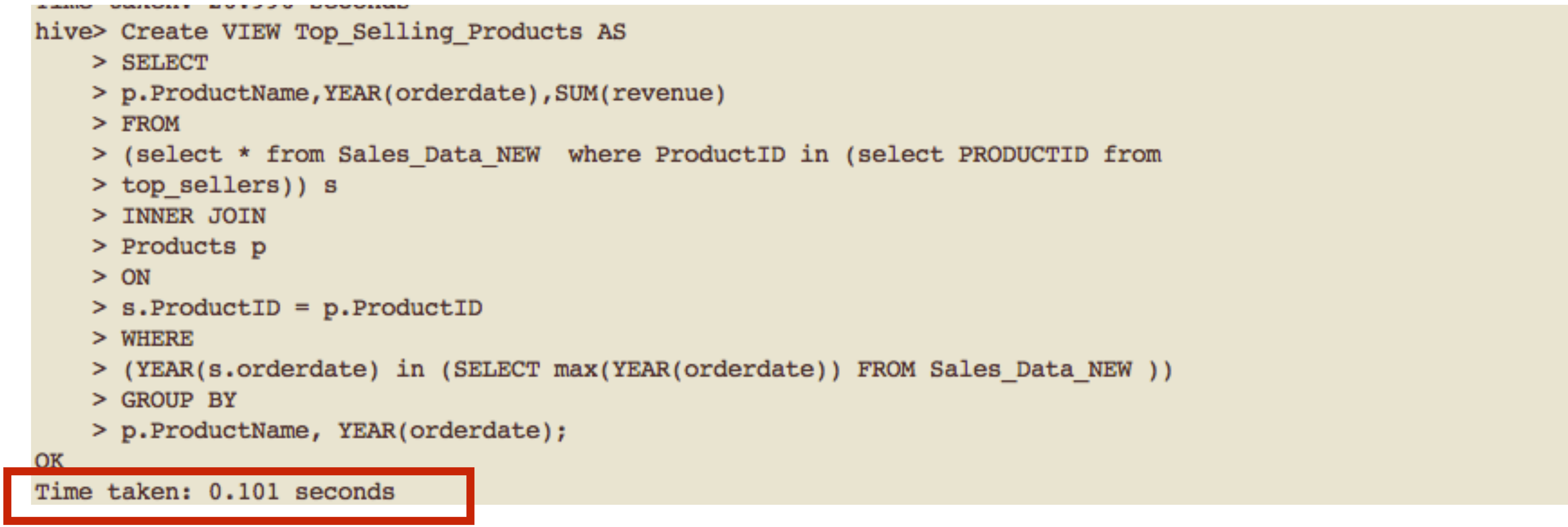
VIEWS ARE DIFFERENT FROM TEMPORARY TABLES

VIEWS

WHEN A VIEW IS CREATED, IT JUST SAVES THE QUERY IN HIVE'S METASTORE

THE QUERY IS NOT EXECUTED AT THE TIME OF VIEW CREATION

THE QUERY IS NOT EXECUTED AT THE TIME OF VIEW CREATION



```
hive> Create VIEW Top_Selling_Products AS
    > SELECT
    > p.ProductName,YEAR(orderdate),SUM(revenue)
    > FROM
    > (select * from Sales_Data_NEW  where ProductID in (select PRODUCTID from
    > top_sellers)) s
    > INNER JOIN
    > Products p
    > ON
    > s.ProductID = p.ProductID
    > WHERE
    > (YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM Sales_Data_NEW ))
    > GROUP BY
    > p.ProductName, YEAR(orderdate);
OK
Time taken: 0.101 seconds
```

VIEWS

YOU CAN TREAT A VIEW LIKE A TABLE AND QUERY IT

```
select ProductName
from Top_Selling_Products;
```

VIEWS

```
select ProductName
from Top_Selling_Products;
```

IN THE BACKGROUND, HIVE WILL REPLACE THE VIEW NAME WITH THE ACTUAL QUERY

```
SELECT
P.PRODUCTNAME,YEAR(ORDERDATE) AS
ORDERDATE,SUM(REVENUE) AS REVENUE
FROM
(SELECT * FROM SALES_DATA_NEW  WHERE PRODUCTID IN (SELECT PRODUCTID FROM
TOP_SELLERS)) S
INNER JOIN
PRODUCTS P
ON
S.PRODUCTID = P.PRODUCTID
WHERE
(YEAR(S.ORDERDATE) IN (SELECT MAX(YEAR(ORDERDATE)) FROM
SALES_DATA_NEW ))
GROUP BY
P.PRODUCTNAME, YEAR(ORDERDATE);
```

VIEWS

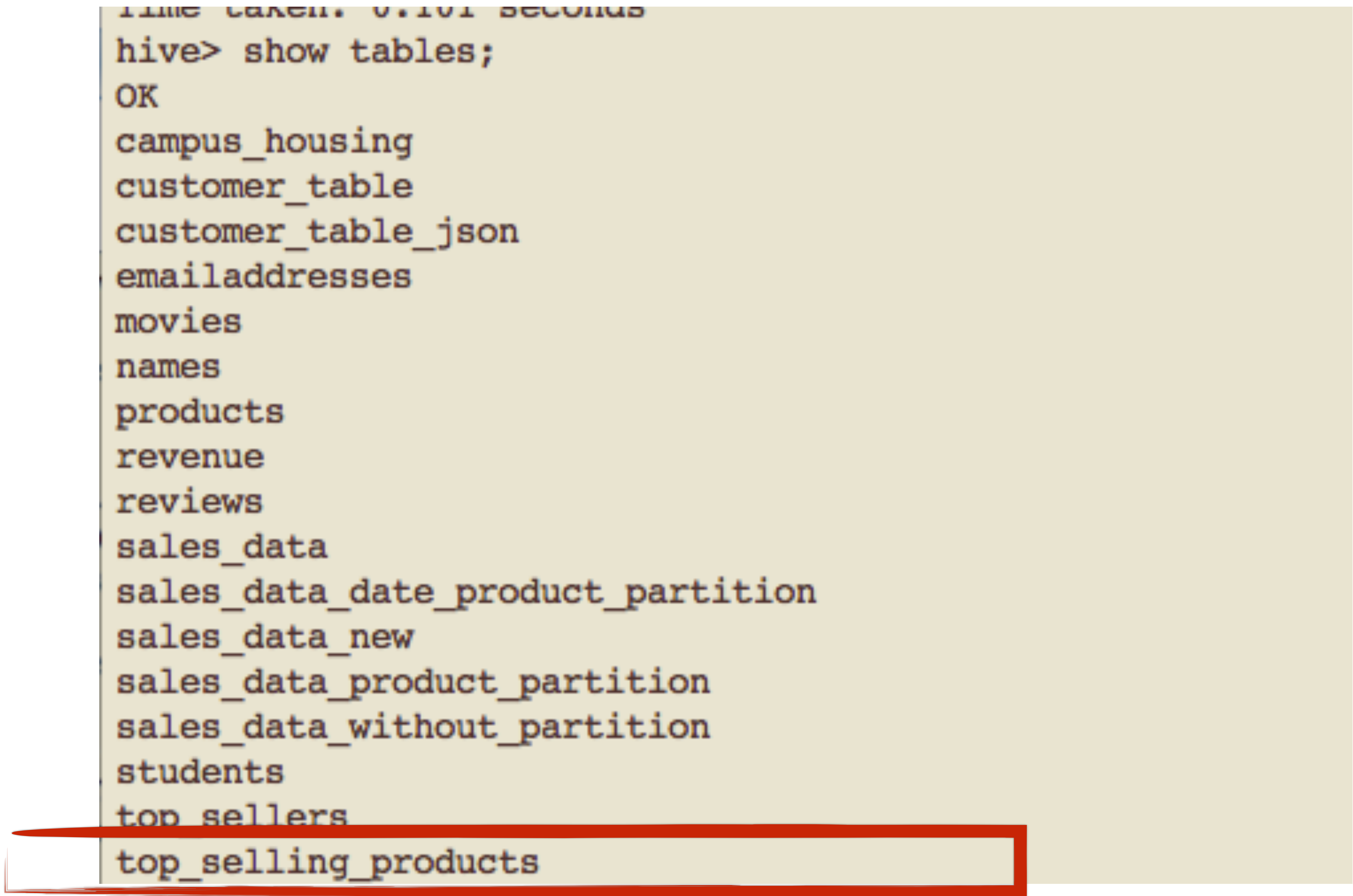
```
select ProductName
from Top_Selling_Products;
```

THE VIEW'S QUERY IS TREATED AS A SUBQUERY

```
SELECT
P.PRODUCTNAME,YEAR(ORDERDATE) AS
ORDERDATE,SUM(REVENUE) AS REVENUE
FROM
(SELECT * FROM SALES_DATA_NEW  WHERE PRODUCTID IN (SELECT PRODUCTID FROM
TOP_SELLERS)) S
INNER JOIN
PRODUCTS P
ON
S.PRODUCTID = P.PRODUCTID
WHERE
(YEAR(S.ORDERDATE) IN (SELECT MAX(YEAR(ORDERDATE)) FROM
SALES_DATA_NEW ))
GROUP BY
P.PRODUCTNAME, YEAR(ORDERDATE);
```

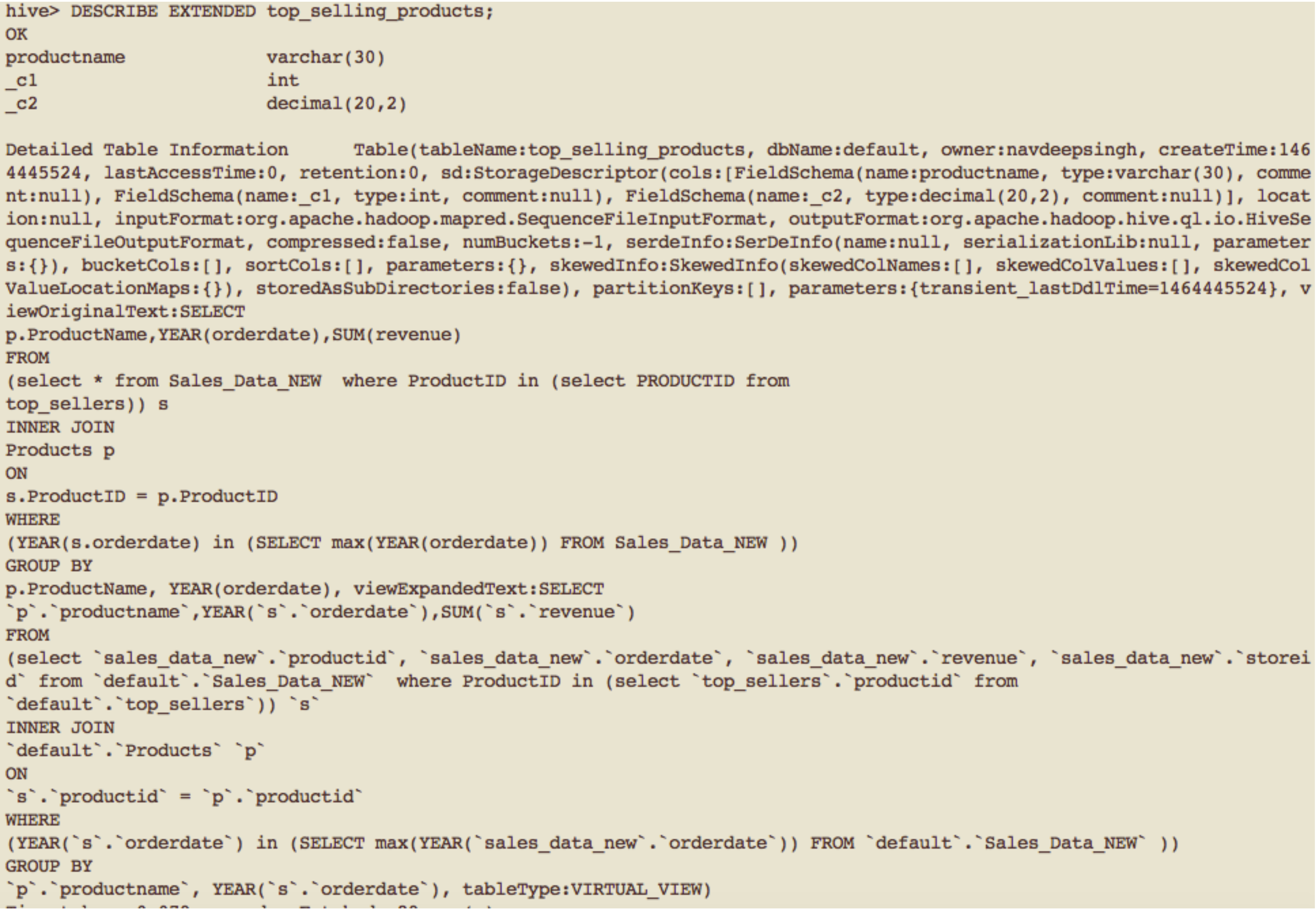
VIEWS

VIEWS ARE INCLUDED IN THE OUTPUT OF "SHOW TABLES" COMMAND



```
Time taken: 0.101 seconds
hive> show tables;
OK
campus_housing
customer_table
customer_table_json
emailaddresses
movies
names
products
revenue
reviews
sales_data
sales_data_date_product_partition
sales_data_new
sales_data_product_partition
sales_data_without_partition
students
top_sellers
top_selling_products
```

ONE CAN GET EXTRA INFORMATION ABOUT THE VIEW BY USING "DESCRIBE EXTENDED"



```
hive> DESCRIBE EXTENDED top_selling_products;
OK
productname             varchar(30)
_c1                     int
_c2                     decimal(20,2)

Detailed Table Information      Table(tableName:top_selling_products, dbName:default, owner:navdeepsingh, createTime:146
4445524, lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:[FieldSchema(name:productname, type:varchar(30), comme
nt:null), FieldSchema(name:_c1, type:int, comment:null), FieldSchema(name:_c2, type:decimal(20,2), comment:null)], locat
ion:null, inputFormat:org.apache.hadoop.mapred.SequenceFileInputFormat, outputFormat:org.apache.hadoop.hive.ql.io.HiveSe
quenceFileOutputFormat, compressed:false, numBuckets:-1, serdeInfo:SerDeInfo(name:null, serializationLib:null, parameter
s:{}), bucketCols:[], sortCols:[], parameters:{}, skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewedCol
ValueLocationMaps:{}), storedAsSubDirectories:false), partitionKeys:[], parameters:{transient_lastDdlTime=1464445524}, v
iewOriginalText:SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
(select * from Sales_Data_NEW  where ProductID in (select PRODUCTID from
top_sellers)) s
INNER JOIN
Products p
ON
s.ProductID = p.ProductID
WHERE
(YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM Sales_Data_NEW ))
GROUP BY
p.ProductName, YEAR(orderdate), viewExpandedText:SELECT
`p`.`productname`,YEAR(`s`.`orderdate`),SUM(`s`.`revenue`)
FROM
(select `sales_data_new`.`productid`, `sales_data_new`.`orderdate`, `sales_data_new`.`revenue`, `sales_data_new`.`storei
d` from `default`.`Sales_Data_NEW`  where ProductID in (select `top_sellers`.`productid` from
`default`.`top_sellers`)) `s`
INNER JOIN
`default`.`Products` `p`
ON
`s`.`productid` = `p`.`productid`
WHERE
(YEAR(`s`.`orderdate`) in (SELECT max(YEAR(`sales_data_new`.`orderdate`)) FROM `default`.`Sales_Data_NEW` ))
GROUP BY
`p`.`productname`, YEAR(`s`.`orderdate`), tableType:VIRTUAL_VIEW)
```

```
Detailed Table Information      Table(tableName:top_selling_products, dbName:default, owner:navdeepsingh, cr
4445524, lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:[FieldSchema(name:productname, type:varcha
nt:null), FieldSchema(name:_c1, type:int, comment:null), FieldSchema(name:_c2, type:decimal(20,2), comment:n
ion:null, inputFormat:org.apache.hadoop.mapred.SequenceFileInputFormat, outputFormat:org.apache.hadoop.hive.
quenceFileOutputFormat, compressed:false, numBuckets:-1, serdeInfo:SerDeInfo(name:null, serializationLib:nul
s:{}), bucketCols:[], sortCols:[], parameters:{}, skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[
ValueLocationMaps:{}), storedAsSubDirectories:false), partitionKeys:[], parameters:{transient_lastDdlTime=14
iewOriginalText:SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
(select * from Sales_Data_NEW  where ProductID in (select PRODUCTID from
top_sellers)) s
INNER JOIN
Products p
ON
s.ProductID = p.ProductID
WHERE
(YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM Sales_Data_NEW ))
GROUP BY
p.ProductName, YEAR(orderdate), viewExpandedText:SELECT
`p`.`productname`,YEAR(`s`.`orderdate`),SUM(`s`.`revenue`)
FROM
(select `sales_data_new`.`productid`, `sales_data_new`.`orderdate`, `sales_data_new`.`revenue`, `sales_data_
d` from `default`.`Sales_Data_NEW`  where ProductID in (select `top_sellers`.`productid` from
`default`.`top_sellers`)) `s`
INNER JOIN
`default`.`Products` `p`
ON
`s`.`productid` = `p`.`productid`
```

THE QUERY TEXT IS PART OF THE VIEW METADATA

VIEWS

VIEWS IN HIVE ARE "READ-ONLY", IT MAY NOT BE USED AS THE TARGET OF LOAD/INSERT

VIEWS

METADATA OF VIEWS CAN BE ALTERED USING THE ALTER VIEW COMMAND

VIEWS

METADATA OF VIEWS CAN BE ALTERED USING ALTER VIEW COMMAND

```
ALTER VIEW Top_Selling_Products AS
SELECT
P.PRODUCTNAME,YEAR(ORDERDATE)AS ORDERDATE,SUM(REVENUE) AS REVENUE
FROM
(SELECT * FROM SALES_DATA_NEW  WHERE PRODUCTID IN (SELECT
PRODUCTID FROM
TOP_SELLERS)) S
INNER JOIN
PRODUCTS P
ON
S.PRODUCTID = P.PRODUCTID
GROUP BY
P.PRODUCTNAME, YEAR(ORDERDATE);
```

WE HAVE CHANGED THE QUERY

VIEWS

VIEWS CAN COME IN PRETTY HANDY FOR

1. REDUCING QUERY COMPLEXITY
2. RESTRICTING ACCESS TO DATA
3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

VIEWS

VIEWS CAN COME IN PRETTY HANDY FOR

1. REDUCING QUERY COMPLEXITY

WE'VE ALREADY SEEN HOW VIEWS HELP WITH THIS

2. RESTRICTING ACCESS TO DATA

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

VIEWS

VIEWS CAN COME IN PRETTY HANDY FOR

1. REDUCING QUERY COMPLEXITY

2. RESTRICTING ACCESS TO DATA

YOU CAN CREATE A VIEW OVER A TABLE WITH

A SUBSET OF COLUMNS

ROWS WHICH SATISFY CERTAIN CONDITIONS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

VIEWS

2. RESTRICTING ACCESS TO DATA

SAY WE HAD A TABLE WITH THE FOLLOWING COLUMNS

OrderID	CustomerID	ProductCategory	OrderDate	Revenue

A LOT OF FOLKS WANT TO QUERY THIS TABLE

ENGINEERS, MARKETERS, CATEGORY MANAGERS

VIEWS

2. RESTRICTING ACCESS TO DATA

OrderID	CustomerID	ProductCategory	OrderDate	Revenue

LET'S CONSIDER CATEGORY MANAGERS

WE DON'T WANT THEM TO HAVE ACCESS TO THE CUSTOMER ID COLUMN

WE ALSO WANT TO ALLOW THE USER TO ONLY SEE THE REVENUE FOR THEIR CATEGORY

VIEWS

2. RESTRICTING ACCESS TO DATA

OrderID	CustomerID	ProductCategory	OrderDate	Revenue

LET'S CONSIDER **CATEGORY MANAGERS**

```
CREATE VIEW
OrderData_Books_Category AS
SELECT ORDERID, PRODUCTCATEGORY,
ORDERDATE, REVENUE FROM
ORDERDATA WHERE
PRODUCTCATEGORY=“BOOKS”;
```

VIEWS

2. RESTRICTING ACCESS TO DATA

```
CREATE VIEW OrderData_Books_Category AS
SELECT ORDERID, PRODUCTCATEGORY,
ORDERDATE, REVENUE FROM
ORDERDATA WHERE
PRODUCTCATEGORY=“BOOKS”;
```

THE VIEW DOES NOT HAVE THE CUSTOMERID COLUMN

VIEWS

2. RESTRICTING ACCESS TO DATA

```
CREATE VIEW OrderData_Books_Category AS
SELECT ORDERID, PRODUCTCATEGORY,
ORDERDATE, REVENUE FROM
ORDERDATA WHERE
PRODUCTCATEGORY=“BOOKS”;
```

THE VIEW ONLY HAS THE DATA FOR PRODUCT CATEGORY “BOOKS”

VIEWS

2. RESTRICTING ACCESS TO DATA

```
CREATE VIEW OrderData_Books_Category AS
SELECT ORDERID, PRODUCTCATEGORY,
ORDERDATE, REVENUE FROM
ORDERDATA WHERE
PRODUCTCATEGORY=“BOOKS”;
```

WE’LL GIVE THE BOOKS CATEGORY MANAGER ACCESS TO THIS VIEW INSTEAD OF THE ORDERDATA TABLE

VIEWS

2. RESTRICTING ACCESS TO DATA

WHY DIDN'T WE CREATE A SEPARATE TABLE INSTEAD OF A VIEW?

THE TABLE WOULD HAVE TO BE UPDATED WHENEVER THE ORIGINAL TABLE CHANGES

VIEWS

VIEWS CAN COME IN PRETTY HANDY FOR

1. REDUCING QUERY COMPLEXITY

2. RESTRICTING ACCESS TO DATA

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

LET'S SAY WE HAVE THE FOLLOWING TABLE

CommunicationID	Type	Date	Attributes

THIS TABLE CONTAINS DETAILS OF CUSTOMER COMMUNICATIONS

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

CommunicationID	Type	Date	Attributes

EMAIL, TWEET, CALL ETC

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

CommunicationID	Type	Date	Attributes

A MAP WITH DIFFERENT KEYS FOR DIFFERENT TYPES OF COMMUNICATION

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

CommunicationID	Type	Date	Attributes

FOR EMAILS

{‘EMAIL ID’:, ‘SUBJECT’:, ...}

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

CommunicationID	Type	Date	Attributes

FOR CALLS

{‘PHONE NO’:, ‘CALL DURATION’:, ...}

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

CommunicationID	Type	Date	Attributes

WE CAN CREATE A VIEW FOR EACH TYPE OF COMMUNICATION

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

CommunicationID	Type	Date	Attributes

FOR EMAILS

```
Create VIEW emailData AS
SELECT Communicationid, Date,
Attributes[“emailID”] as emailID,
Attributes[“Subject”] as Subject
from
CommData where Type=“EMAIL”;
```

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

CommunicationID	Type	Date	Attributes

FOR EMAILS

THE VIEW CONTAINS ONLY THE EMAIL COMMUNICATIONS

```
Create VIEW emailData AS
SELECT CommunicationID, Date,
Attributes["email"] as emailID,
Attributes["Subject"] as Subject
from
CommData where Type="EMAIL";
```

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

CommunicationID	Type	Date	Attributes

FOR EMAILS

```
Create VIEW emailData AS
SELECT Communicationid, Date,
Attributes[“emailID”] as emailID,
Attributes[“Subject”] as Subject
from
CommData where Type=“EMAIL”;
```

THE VIEW HAS THE EMAILID AND SUBJECT ATTRIBUTES AS COLUMNS

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

CommunicationID	Type	Date	Attributes

FOR CALLS

```
Create VIEW callData AS
SELECT Communicationid, Date,
Attributes[“PhoneNo”] as PhoneNo,
Attributes[“CallDuration”] as CallDuration
from
CommData where Type=“CALL”;
```

FOR CALLS, WE HAVE A DIFFERENT SET OF ATTRIBUTES

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

CommunicationID	Type	Date	Attributes

FOR CALLS

```
Create VIEW callData AS
SELECT Communicationid, Date,
Attributes[“PhoneNo”] as PhoneNo,
Attributes[“CallDuration”] as CallDuration
from
CommData where Type=“CALL”;
```

THE CALLS VIEW HAS DIFFERENT COLUMNS

VIEWS

3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE

CommunicationID	Type	Date	Attributes

IN EFFECT WE CREATED DIFFERENT LOGICAL TABLES WHICH USE THE SAME UNDERLYING PHYSICAL TABLE

VIEWS

VIEWS CAN COME IN PRETTY HANDY FOR

1. REDUCING QUERY COMPLEXITY
2. RESTRICTING ACCESS TO DATA
3. CONSTRUCTING DIFFERENT LOGICAL TABLES USING 1 PHYSICAL TABLE