SUBQUERIES

FLASHBACK: SO FAR WE HAVE SEEN QUERIES AS STANDALONE COMMANDS THAT FETCH DATA FROM A DATABASE

BUT IN REALITY, QUERIES ARE PRETTY PLUG-AND-PLAY

BUT IN REALITY, QUERIES ARE PRETTY PLUG-AND-PLAY

A QUERY IS A COMMAND THAT RETURNS A TABLE (ROWS AND COLUMNS)

BUT IN REALITY, QUERIES ARE PRETTY PLUG-AND-PLAY

WE COULD CALCULATE THE UNION OF 2 QUERIES

BUT IN REALITY, QUERIES ARE PRETTY

PLUG-AND-PLAY

WE COULD CALCULATE THE
UNION OF 2 QUERIES

WE COULD USE ONE QUERY INSIDE
ANOTHER (VIA SUBQUERIES)

BUT IN REALITY, QUERIES ARE PRETTY PLUG-AND-PLAY

WE COULD CALCULATE THE UNION OF 2 QUERIES

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

WE COULD USE A SUBQUERY TO POPULATE A TABLE VIA INSERT

BUT IN REALITY, QUERIES ARE PRETTY PLUG-AND-PLAY

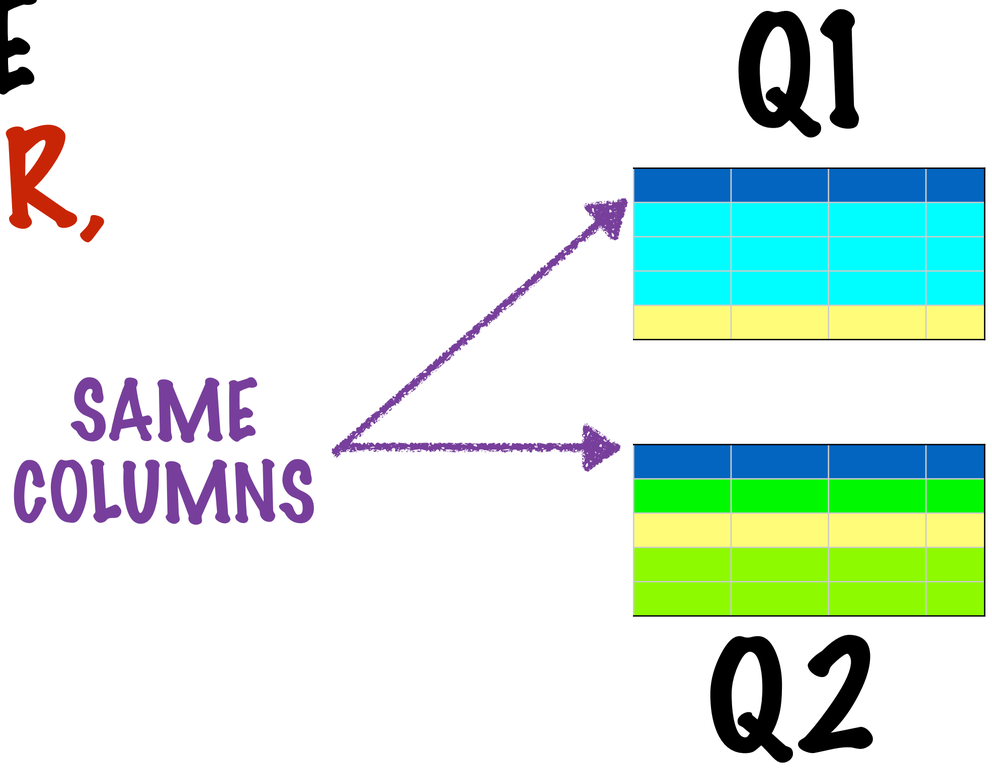
WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

WE COULD CALCULATE THE UNION OF 2 QUERIES

WE COULD USE A SUBQUERY TO POPULATE A TABLE VIA INSERT

WE COULD CALCULATE THE UNION OF 2 QUERIES

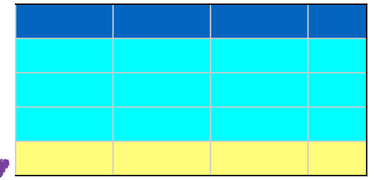
PROVIDED THEY HAVE THE SAME COLUMNS (NUMBER, ORDER AND TYPE)



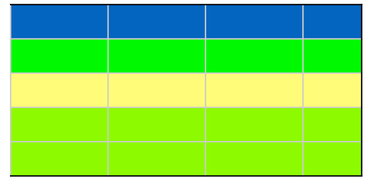
WE COULD CALCULATE THE UNION OF 2 QUERIES

PROVIDED THEY HAVE THE SAME COLUMNS (NUMBER, ORDER AND TYPE)

Q1



SOME COMMON ROWS



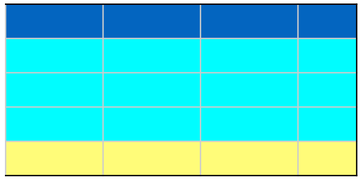
Q2

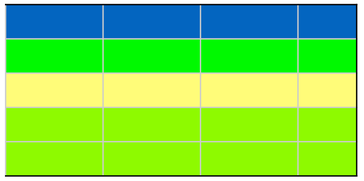
WE COULD CALCULATE THE UNION OF 2 QUERIES

PROVIDED THEY HAVE THE SAME COLUMNS (NUMBER, ORDER AND TYPE)

A QUERY IS A COMMAND THAT RETURNS A TABLE (ROWS AND COLUMNS)

Q1



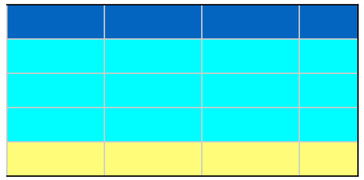


Q2

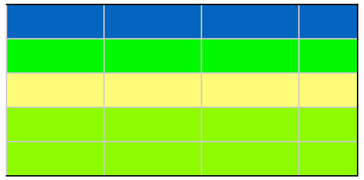
WE COULD CALCULATE THE UNION OF 2 QUERIES

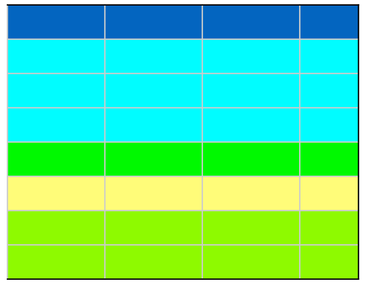
PROVIDED THEY HAVE THE SAME COLUMNS (NUMBER, ORDER AND TYPE)

A QUERY IS A COMMAND THAT RETURNS A TABLE (ROWS AND COLUMNS)

Q1 

UNION

Q2 

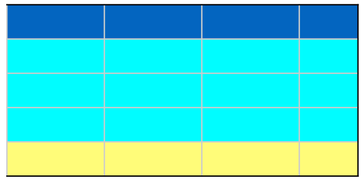
Q3 

ONLY 1 COPY OF THE COMMON ROWS (I.E. NO DUPLICATES)

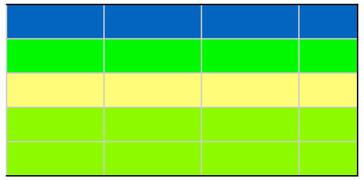
WE COULD CALCULATE THE UNION OF 2 QUERIES

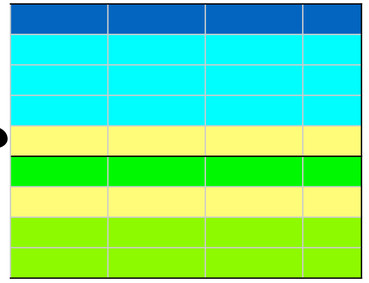
PROVIDED THEY HAVE THE SAME COLUMNS (NUMBER, ORDER AND TYPE)

A QUERY IS A COMMAND THAT RETURNS A TABLE (ROWS AND COLUMNS)

Q1 

UNION ALL

Q2 

Q4 

UNION ALL MAINTAINS DUPLICATES

WE COULD CALCULATE
THE UNION OF 2 QUERIES

ALSO THE INDIVIDUAL
QUERIES IN A UNION
CANNOT USE ORDER BY

WE COULD CALCULATE
THE UNION OF 2 QUERIES

HIVE DOES NOT SUPPORT

INTERSECT

MINUS

WHICH ARE SUPPORTED IN SQL

UNION

PET OWNERS

AptNumber	Name
123	John
345	Tim
349	Nikhil
567	Bilal

SAME COLUMNS (NUMER, ORDER AND TYPE - NAMES COULD DIFFER)

APT OWNERS

FlatNumber	Name
234	Mary
567	Bilal
897	Alan
903	Ellen

UNION

PET OWNERS

AptNumber	Name
123	John
345	Tim
349	Nikhil
567	Bilal

1 COMMON ROW

APT OWNERS

FlatNumber	Name
234	Mary
567	Bilal
897	Alan
903	Ellen

UNION

```
SELECT APTNUMBER, NAME
FROM PetOwners
```

PET OWNERS

AptNumber	Name
123	John
345	Tim
349	Nikhil
567	Bilal

UNION

```
SELECT FLATNUMBER AS
APTNUMBER, NAME FROM
AptOwners;
```

APT OWNERS

FlatNumber	Name
234	Mary
567	Bilal
897	Alan
903	Ellen

UNION

```
SELECT APTNUMBER, NAME
FROM PetOwners
```

UNION

```
SELECT FLATNUMBER AS
APTNUMBER, NAME FROM
AptOwners;
```

ONLY 1 COPY OF THE COMMON ROWS (I.E. NO DUPLICATES)

AptNumber	Name
123	John
345	Tim
349	Nikhil
234	Mary
567	Bilal
897	Alan
903	Ellen

UNION

```
SELECT APTNUMBER, NAME
FROM PetOwners
```

PET OWNERS

AptNumber	Name
123	John
345	Tim
349	Nikhil
567	Bilal

UNION ALL

```
SELECT FLATNUMBER AS
APTNUMBER, NAME FROM
AptOwners;
```

APT OWNERS

FlatNumber	Name
234	Mary
567	Bilal
897	Alan
903	Ellen

UNION

```
SELECT APTNUMBER, NAME
FROM PetOwners
```

UNION ALL

```
SELECT FLATNUMBER AS
APTNUMBER, NAME FROM
AptOwners;
```

UNION ALL MAINTAINS DUPLICATES

AptNumber	Name
123	John
345	Tim
349	Nikhil
567	Bilal
234	Mary
567	Bilal
897	Alan
903	Ellen

UNION

```
(SELECT APTNUMBER, NAME FROM
  PetOwners ORDER BY NAME)
```

WONTWORK!!

```
(SELECT FLATNUMBER AS
APTNUMBER, NAME FROM
AptOwners ORDER BY NAME);
```

PET OWNERS

AptNumber	Name
123	John
345	Tim
349	Nikhil
567	Bilal

APT OWNERS

FlatNumber	Name
234	Mary
567	Bilal
897	Alan
903	Ellen

UNION

THIS IS FINE!

```
SELECT APTNUMBER,
NAME FROM
PetOwners
UNION
SELECT FLATNUMBER AS
APTNUMBER, NAME FROM
AptOwners
Order by APTNUMBER;
```

PET OWNERS

AptNumber	Name
123	John
345	Tim
349	Nikhil
567	Bilal

APT OWNERS

FlatNumber	Name
234	Mary
567	Bilal
897	Alan
903	Ellen

BUT IN REALITY, QUERIES ARE PRETTY PLUG-AND-PLAY

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

WE COULD CALCULATE THE UNION OF 2 QUERIES

WE COULD USE A SUBQUERY TO POPULATE A TABLE VIA INSERT

BUT IN REALITY, QUERIES ARE PRETTY PLUG-AND-PLAY

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

WE COULD CALCULATE THE UNION OF 2 QUERIES

WE COULD USE A SUBQUERY TO POPULATE A TABLE VIA INSERT

WE COULD USE ONE
QUERY INSIDE ANOTHER
(VIA SUBQUERIES)

HIVE OFFERS LIMITED
SUPPORT FOR SUBQUERIES

WE COULD USE ONE
QUERY INSIDE ANOTHER
(VIA SUBQUERIES)

HIVE SUPPORTS SUBQUERIES

ONLY IN THE FROM CLAUSE

&

IN THE WHERE CLAUSE

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

HIVE SUPPORTS SUBQUERIES

IN THE WHERE CLAUSE

USING

IN/EXISTS

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

HIVE SUPPORTS SUBQUERIES

IN THE WHERE CLAUSE

THROUGH

IN/EXISTS

AND NOT THROUGH

"=" EQUALITY

IN/EXISTS

HIVE SUPPORTS SUBQUERIES

IN THE WHERE CLAUSE

THROUGH

IN

EXISTS

IN

'WHERE IN' RETURNS ROWS THAT MATCHES VALUES IN A SUBQUERY

LET US SAY WE HAVE TWO TABLES

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

IN

'WHERE IN' RETURNS ROWS THAT MATCHES VALUES IN A SUBQUERY

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

```
SELECT NAMES.SYMBOL
FROM NAMES
WHERE  NAMES.SYMBOL IN
(SELECT SYMBOL FROM REVENUE);
```

IN

'WHERE IN' RETURNS ROWS THAT MATCHES VALUES IN A SUBQUERY

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

```
SELECT NAMES.SYMBOL
FROM NAMES
WHERE  NAMES.SYMBOL IN
 (SELECT SYMBOL FROM REVENUE);
```

IN

'WHERE IN' RETURNS ROWS THAT MATCHES VALUES IN A SUBQUERY

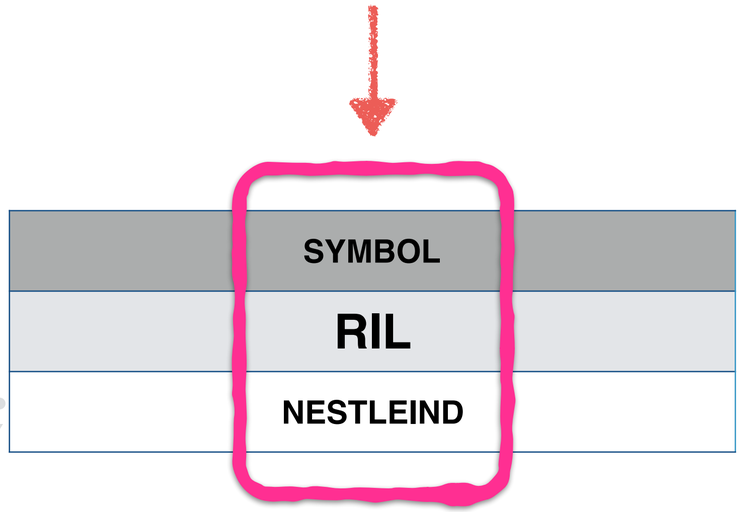
TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

```
SELECT NAMES.SYMBOL
FROM NAMES
WHERE  NAMES.SYMBOL IN
 (SELECT SYMBOL FROM REVENUE);
```



SYMBOL
RIL
NESTLEIND

NOT IN

'WHERE IN' RETURNS ROWS THAT DOES NOT MATCH VALUES IN A SUBQUERY

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

```
SELECT NAMES.SYMBOL
FROM NAMES
WHERE  NAMES.SYMBOL NOT IN
(SELECT SYMBOL FROM REVENUE);
```

SYMBOL
TATA

NOT IN & IN

IN/NOT IN SUBQUERIES ONLY SELECT A SINGLE COLUMN

IN

IN/NOT IN SUBQUERIES ONLY SELECT A SINGLE COLUMN

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

```
SELECT NAMES.SYMBOL
FROM NAMES
WHERE  NAMES.SYMBOL IN
 (SELECT SYMBOL FROM REVENUE);
```

SYMBOL
RIL
NESTLEIND

NAMES

IN/EXISTS

IN/EXISTS SUBQUERY ARE SUPPORTED IN THE WHERE CLAUSE

IN

EXISTS

EXISTS

THE SQL EXISTS CONDITION IS USED IN COMBINATION WITH A SUBQUERY

```
SELECT *
FROM TABLE_NAME
WHERE   EXISTS
SUBQUERY;
```

IF THE SUBQUERY RETURNS AT LEAST ONE ROW, THE CONDITION IS SAID TO BE MET

EXISTS

THE SQL EXISTS CONDITION IS USED IN COMBINATION WITH A SUBQUERY

IF THE SUBQUERY RETURNS AT LEAST ONE ROW, THE CONDITION IS SAID TO BE MET

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

EXISTS

IF THE SUBQUERY RETURNS AT LEAST ONE ROW, THE CONDITION IS SAID TO BE MET

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

```
SELECT NAMES.SYMBOL
FROM NAMES
WHERE  EXISTS
(SELECT TOTAL_REVENUE FROM REVENUE
NAMES.name = REVENUE.name);
```

EXISTS

IF THE SUBQUERY RETURNS AT LEAST ONE ROW, THE CONDITION IS SAID TO BE MET

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

```
SELECT NAMES.SYMBOL
FROM NAMES
WHERE  EXISTS
(SELECT TOTAL_REVENUE FROM REVENUE
NAMES.name = REVENUE.name);
```

EXISTS

IF THE SUBQUERY RETURNS AT LEAST ONE ROW, THE CONDITION IS SAID TO BE MET

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

```
SELECT NAMES.SYMBOL
FROM NAMES
WHERE  EXISTS
(SELECT TOTAL_REVENUE FROM REVENUE
NAMES.name = REVENUE.name);
```

EXISTS

THERE IS ONE MORE CONDITION

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

THE SUBQUERY NEEDS TO BE CORRELATED

(SELECT TOTAL_REVENUE FROM REVENUE

```
NAMES.name = REVENUE.name);
```

THE SUBQUERY NEEDS TO BE CORRELATED

OUTER QUERY

```
SELECT *
FROM TABLE_NAME
WHERE   EXISTS
SUBQUERY;
```

A SUBQUERY IS CORRELATED WHEN IT USES A VALUE FROM THE OUTER QUERY

EXISTS

THE REFERENCED COLUMNS .name SHOULD BE IN SMALL CAPS

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

THE SUBQUERY NEEDS TO BE CORRELATED

```
FROM NAMES
WHERE  EXISTS
(SELECT TOTAL_REVENUE FROM REVENUE
NAMES.name = REVENUE.name);
```

THE SUBQUERY NEEDS TO BE CORRELATED

THE REFERENCED COLUMNS .name SHOULD BE IN SMALL CAPS

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

IF YOU DON'T USE SMALL CAPS, YOU WILL SEE THIS ERROR

```
FAILED: SemanticException [Error 10250]: Line 1:84 Invalid SubQuery expression 'SYMBOL': For Exists/Not Exists operator
SubQuery must be Correlated.
```

EXISTS

IF THE SUBQUERY RETURNS AT LEAST ONE ROW, THE CONDITION IS SAID TO BE MET

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

```
SELECT NAMES.SYMBOL
FROM NAMES
WHERE  EXISTS
(SELECT TOTAL_REVENUE FROM REVENUE
NAMES.name = REVENUE.name);
```

SYMBOL
RIL
NESTLEIND

NOT EXISTS

IF THE SUBQUERY DOES NOT RETURN AT LEAST ONE ROW, THE CONDITION IS SAID TO BE MET

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

REVENUE

SYMBOL	NAME	SECTOR
RIL	Reliance	ENERGY
NESTLEIND	Nestle	FMCG
TATA	TATA	AUTOMOBILE

NAMES

```
SELECT NAMES.SYMBOL
FROM NAMES
WHERE NOT EXISTS
(SELECT TOTAL_REVENUE FROM REVENUE
NAMES.name = REVENUE.name);
```

SYMBOL
TATA

WE COULD USE ONE
QUERY INSIDE ANOTHER
(VIA SUBQUERIES)

LET'S SEE AN EXAMPLE

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

'STORES'

StoreID	StoreLocation	City
1	Bellandur	Bangalore
2	Koramangala	Bangalore

'PRODUCTS'

ProductID	ProductName
1	Bananas
2	Milk
3	Nutella
4	Peanut Butter

'SALES_DATA'

StoreID	ProductID	Date	Revenue
1	1	January 18,2016	8,236.33
1	3	January 18,2016	7,455.67
1	4	January 18,2016	5,316.89
1	2	January 18,2016	2,433.76
2	1	January 18,2016	9,456.01
2	3	January 18,2016	3,644.33
2	4	January 18,2016	8,988.64

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

IF SUCH A DATABASE ACTUALLY EXISTED

WE COULD USE ONE
QUERY INSIDE ANOTHER
(VIA SUBQUERIES)

IF SUCH A DATABASE ACTUALLY EXISTED

YOU CAN BET THAT SOMEONE
WOULD PULL A REPORT FOR
ANNUAL REVENUE

WE COULD USE ONE
QUERY INSIDE ANOTHER
(VIA SUBQUERIES)

IF SUCH A DATABASE ACTUALLY EXISTED

YOU CAN BET THAT SOMEONE WOULD
PULL A REPORT FOR ANNUAL REVENUE

IN FACT, THERE WOULD BE A
REALLY BORING WEEKLY MEETING
AROUND THIS REPORT

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'SALES_DATA'

StoreID	ProductID	Date	Revenue

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_Data s
INNER JOIN
Products p
ON
s.ProductID = p.ProductID
WHERE
(p.ProductName = 'Peanut Butter' or p.ProductName =
'Nutella') AND (YEAR(OrderDate) = 2016)
GROUP BY
p.ProductName, YEAR(orderdate);
```

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'SALES_DATA'

StoreID	ProductID	Date	Revenue

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_Data s
INNER JOIN
Products p
ON
s.ProductID = p.ProductID
WHERE
(p.ProductName = 'Peanut Butter' or p.ProductName =
'Nutella') AND (YEAR(OrderDate) = 2016)
GROUP BY
p.ProductName, YEAR(orderdate);
```

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'SALES_DATA'

StoreID	ProductID	Date	Revenue

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_Data s
INNER JOIN
Products p
ON
s.ProductID = p.ProductID
WHERE
(p.ProductName = 'Peanut Butter' or p.ProductName =
'Nutella') AND (YEAR(OrderDate) = 2016)
GROUP BY
p.ProductName, YEAR(orderdate);
```

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_Data s
INNER JOIN
Products p
ON
s.ProductID = p.ProductID
WHERE
(p.ProductName = 'Peanut Butter' or p.ProductName =
'Nutella') AND (YEAR(OrderDate) = 2016)
GROUP BY
p.ProductName, YEAR(orderdate);
```

'SALES_DATA'

StoreID	ProductID	Date	Revenue

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'SALES_DATA'

StoreID	ProductID	Date	Revenue

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_Data s
INNER JOIN
Products p
ON
s.ProductID = p.ProductID
WHERE
(p.ProductName = 'Peanut Butter' or p.ProductName =
'Nutella') AND (YEAR(OrderDate) = 2016)
GROUP BY
p.ProductName, YEAR(orderdate);
```

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_Data s
INNER JOIN
Products p
ON
s
W
(
'
G
p
```

'SALES_DATA'

StoreID	ProductID	Date	Revenue

ERRM..THIS QUERY WILL NEED TO BE RE-WRITTEN EACH YEAR :-|

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_Data s
INNER JOIN
Products p
ON
s.ProductID = p.ProductID
```

'SALES_DATA'

StoreID	ProductID	Date	Revenue

```
la')
```

NOT IF WE USE A SUB-QUERY!

D

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'_DATA'

Revenue

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_
INNER
Produc
ON
s.Proc
WHERE
(p.Product
AND
(YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM Sales_Data ))
GROUP BY
p.ProductName, YEAR(orderdate);
```

WE JUST PLUGGED ONE QUERY INTO ANOTHER!

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'SALES_DATA'

Revenue

```
SELECT
p.Pr
FROM
Sale
INNE
Prod
ON
s.Pr
WHER
(p.ProductName = 'Peanut Butter' or p.ProductName = 'Nutella')
AND
(YEAR(s.orderdate) =(SELECT max(YEAR(orderdate)) FROM Sales_Data ))
GROUP BY
p.ProductName, YEAR(orderdate);
```

HERE "=" WILL NOT WORK!

X

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_
INNER
Produc
ON
s.Proc
WHERE
(p.Pro
AND
(YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM Sales_Data ))
GROUP BY
p.ProductName, YEAR(orderdate);
```

HERE, THE INNER QUERY RETURNS A SINGLE VALUE..

_DATA'

Revenue

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'SALES_DATA'

Revenue

```
SELECT
p.ProductName, YEAR(orderdate), SUM(revenue)
FROM
Sales
INNER
Produ
ON
s.Pro
WHERE
(p.ProductName = 'Peanut Butter' or p.ProductName = 'Nutella')
AND
(YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM Sales_Data ))
GROUP BY
p.ProductName, YEAR(orderdate);
```

AND THE OUTER QUERY USES THAT VALUE IN THE WHERE CLAUSE

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_Data s
INNER JOIN
Products p
ON
s.ProductID = p.ProductID
WHERE
(p.ProductName = 'Peanut Butter' or p.ProductName = 'Nutella')
AND
(YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM Sales_Data
))
GROUP BY
p.ProductName, YEAR(orderdate);
```

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

'SALES_DATA'

StoreID	ProductID	Date	Revenue

YOU CAN BET THAT
D PULL A
NNUAL
E

LET'S SAY WE DON'T WANT TO HARD CODE THE PRODUCTS, INSTEAD WE HAVE A TOP SELLERS TABLE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

'SALES_DATA'

StoreID	ProductID	Date	Revenue

```
ue)
s.ProductID = p.ProductID
WHERE
(p.ProductName = 'Peanut Butter' or p.ProductName = 'Nutella')
AND
(YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM
 Sales_Data ))
GROUP BY
p.ProductName, YEAR(orderdate);
```

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

WE MIGHT WANT TO DO SOMETHING LIKE THIS

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

roductID	ProductName

'SALES_DATA'

Date	Revenue

```
WHERE
(p.ProductName IN (SELECT ProductName FROM Top_Sellers))
AND
(YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM
Sales_Data ))
GROUP BY
p.ProductName, YEAR(orderdate);
```

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ctID	ProductName

ALES_DATA'

ate	Revenue

UNFORTUNATELY, THIS IS NOT SUPPORTED IN HIVE

```
SELECT
WHERE
(p.ProductName IN (SELECT ProductName FROM Top_Sellers))
AND
(YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM
Sales_Data ))
GROUP BY
p.ProductName, YEAR(orderdate);
```

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

S_DATA'

			Revenue

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sale
INNE
Proc
ON
s.Pr
WHERE
(p.ProductName IN (SELECT ProductName FROM Top_Sellers))
AND
(YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM
Sales Data ))
p.ProductName, YEAR(orderdate);
```

THIS IS THE ERROR MESSAGE IN HIVE 2.0.0

```
FAILED: SemanticException [Error 10249]: Line 12:19 Unsupported SubQuery Expression 'orderdate': Only 1 SubQuery expres
ion is supported.
```

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

_DATA'

			Revenue

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sale
INNE
Proc
ON
s.Pr
WHERE
 (p.ProductName IN (SELECT ProductName FROM Top_Sellers))
AND
 (YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM
Sales_Data ))
```

CURRENTLY, ONLY 1 SUBQUERY EXPRESSION IS SUPPORTED

```
FAILED: SemanticException [Error 10249]: Line 12:19 Unsupported SubQuery Expression 'orderdate': Only 1 SubQuery express
ion is supported.
```

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

'SALES DATA'

ue

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_Data s
(YEAR(s.orderdate) in (SELECT max(YEAR(orderdate)) FROM
p.ProductName, YEAR(orderdate);
```

IN THIS QUERY WE CAN FIX THIS IN A SLIGHTLY DIFFERENT WAY, YOU'LL SEE THIS LATER IN THIS CLASS

```
FAILED: SemanticException [Error 10249]: Line 12:19 Unsupported SubQuery Expression 'orderdate': Only 1 SubQuery express
ion is supported.
```

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
Sales_Data s
INNER JOIN
Products p
ON
s.Pr
WHER
(YEA
Sale
GROU
p.Pr
```

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

'SALES_DATA'

StoreID	ProductID	Date	Revenue

TIME PASSES, AND OUR SALES_DATA TABLE BECOMES ENORMOUS - TOO BIG TO PULL LIKE THIS

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
(select * from Sales_Data where ProductID in (select
PRODUCTID from Top_Sellers)) s
INNER JOIN
Products p
ON
s.Pro
WHERE
(YEAR
Sales
GROUP
p.Pro
```

'SALES_DATA'

Revenue

WE JUST PLUGGED ONE QUERY INTO ANOTHER!

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
(select * from Sales_Data where ProductID in (select
PRODUCTID from Top_Sellers)) s
INNER JOIN
Produ
ON
s.Pro
WHERE
(YEAR
Sales
GROUP
p.ProductName, YEAR(orderdate);
```

'SALES_DATA'

Revenue

HERE, THE INNER QUERY IS A SUBSET OF A TABLE...

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
(select * from Sales_Data  where ProductID in (select
PRODUCTID from Top_Sellers)) s
INNER JOIN
Produ
ON
s.Pro
WHERE
(YEAR
Sales
GROUP
p.Pro
```

'SALES_DATA'

Revenue

THE OUTER QUERY USES THAT TABLE SUBSET (ALSO A TABLE) IN THE FROM CLAUSE

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

'SALES_DATA'

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
(select * from Sales_Data  where ProductID in (select
PRODUCTID from Top_Sellers)) s
INNER JOIN
Products p
ON
s.
WHE
(YE
Sal
GRO
p.
```

THE OUTER QUERY ASSESSES THE RESULTS OF THE SUBQUERY LIKE IT DOES WITH A TABLE

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

'SALES_DATA'

Revenue

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
(select * from Sales_Data where ProductID in (select PRODUCTID
from Top_Sellers))  s
INNER JOIN
Produc
ON
s.Prod
WHERE
(YEAR(
Sales_
GROUP
p.Prod
```

THE SUB-QUERY QUERY MUST BE GIVEN AN ALIAS. HERE 'S' IS THE SUBQUERY ALIAS

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

'SALES_DATA'

Revenue

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
(select * from Sales_Data where ProductID in (select PRODUCTID
from Top_Sellers))  s
INNER JOIN
Produ
ON
s.Pro
WHERE
(YEAR
Sales
GROUP
p.Pro
```

THIS ALSO FIXES OUR OLDER ISSUE WHERE WE COULD NOT USE MORE THAN ONE SUBQUERY IN A TABLE

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

'PRODUCTS'

ProductID	ProductName

'TOP SELLERS'

ProductID	ProductName

```
SELECT
p.ProductName,YEAR(orderdate),SUM(revenue)
FROM
(select * from Sales_Data  where ProductID in (select
PRODUCTID from Top_Sellers)) s
```

THE QUERY IS NOW ENTIRELY FREE OF HARDCODED VALUES OR VERY LARGE INTERMEDIATE TABLES - THANKS TO THE USE OF SUBQUERIES!

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

IF SUCH A DATABASE ACTUALLY EXISTED

YOU CAN BET THAT SOMEONE WOULD PULL A REPORT FOR ANNUAL REVENUE

IN FACT, THERE WOULD BE A REALLY BORING WEEKLY MEETING AROUND THIS REPORT

THE QUERY IS NOW ENTIRELY FREE OF HARDCODED VALUES OR VERY LARGE INTERMEDIATE TABLES - THANKS TO THE USE OF SUBQUERIES!

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

'STORES'

StoreID	StoreLocation	City
1	Bellandur	Bangalore
2	Koramangala	Bangalore

'PRODUCTS'

ProductID	ProductName
1	Bananas
2	Milk
3	Nutella
4	Peanut Butter

SUBQUERIES REALLY SIMPLIFY & ROBUSTIFY QUERIES :-)

'SALES_DATA'

StoreID	ProductID	Date	Revenue
1	1	January 18,2016	8,236.33
1	3	January 18,2016	7,455.67
1	4	January 18,2016	5,216.89

WE COULD USE ONE
QUERY INSIDE ANOTHER
(VIA SUBQUERIES)

SUBQUERIES REALLY SIMPLIFY &
ROBUSTIFY QUERIES :-)

HIVE SUPPORTS SUBQUERY

IN THE WHERE/FROM CLAUSE

THROUGH

IN

EXISTS

BUT IN REALITY, QUERIES ARE PRETTY PLUG-AND-PLAY

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

WE COULD CALCULATE THE UNION OF 2 QUERIES

WE COULD USE A SUBQUERY TO POPULATE A TABLE VIA INSERT

BUT IN REALITY, QUERIES ARE PRETTY PLUG-AND-PLAY

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

WE COULD CALCULATE THE UNION OF 2 QUERIES

WE COULD USE A SUBQUERY TO POPULATE A TABLE VIA INSERT

WE COULD USE A SUBQUERY TO POPULATE A TABLE VIA INSERT

WE HAVE ALREADY COME ACROSS THE INSERT & CREATE TABLE STATEMENTS

RECAP

LET'S SAY WE WANT TO CREATE TABLE WHICH STORES LIST OF PRODUCTS ALONG WITH THE STORE LOCATIONS

CREATE A TABLE FIRST

```
CREATE TABLE stores_product_data
(
StoreLocation VARCHAR(30),
Product VARCHAR(30)
);
```

StoreLocation	Product
Bellandur	Bananas
Bellandur	Nutella
Bellandur	Peanut Butter
Bellandur	Milk
Koramangala	Bananas
Koramangala	Nutella
Koramangala	Peanut Butter
Koramangala	Milk

StoreLocation	Product	Date	Revenue

RECAP

INSERT INTO THAT TABLE

StoreLocation	Product
Bellandur	Bananas
Bellandur	Nutella
Bellandur	Peanut Butter
Bellandur	Milk
Koramangala	Bananas
Koramangala	Nutella
Koramangala	Peanut Butter
Koramangala	Milk

```
insert overwrite table stores_product_data
select StoreLocation,Product
from Sales_Data;
```

StoreLocation	Product	Date	Revenue

WE USED A SUBQUERY TO POPULATE A TABLE VIA INSERT

SUBQUERIES ARE FAR MORE CONVENIENT TO POPULATE A TABLE FROM EXISTING TABLES IN THE DATABASE

BUT SUBQUERIES ARE FAR MORE CONVENIENT TO POPULATE A TABLE FROM EXISTING TABLES IN THE DATABASE

WE COULD EITHER -

- CREATE THE TABLE AS USUAL, THEN INSERT DATA USING A SUBQUERY
- CREATE THE TABLE AND POPULATE DIRECTLY USING A SUBQUERY

- # CREATE THE TABLE AS USUAL, THEN INSERT DATA USING A SUBQUERY

```
CREATE TABLE EmailAddresses
(
Email VARCHAR(30),
Category VARCHAR(10)
);

INSERT INTO EmailAddresses
SELECT distinct Email,"Student" AS Category FROM
Students;
```

• CREATE THE TABLE AS USUAL, THEN INSERT DATA USING A SUBQUERY

```
CREATE TABLE EmailAddresses
(
Email VARCHAR(30),
Category VARCHAR(10)
);
```

FIRST CREATE THE TABLE EXACTLY AS USUAL

```
INSERT INTO EmailAddresses
SELECT distinct Email,"Student" AS Category FROM
Students;
```

• CREATE THE TABLE AS USUAL, THEN INSERT DATA USING A SUBQUERY

```
CREATE TABLE EmailAddresses
(
Email VARCHAR(30),
Category VARCHAR(10)
);
```

NEXT USE A DIFFERENT FORM OF THE INSERT STATEMENT, THAT IS FOLLOWED BY A QUERY

```
INSERT INTO EmailAddresses
SELECT distinct Email,"Student" AS Category FROM
Students;
```

• CREATE THE TABLE AS USUAL, THEN INSERT DATA USING A SUBQUERY

```
CREATE TABLE EmailAddresses
(
Email VARCHAR(30),
Category VARCHAR(10)
);

INSERT INTO EmailAddresses
SELECT distinct Email,"Student" AS Category FROM
Students;
```

THIS INSERT STARTS OFF AS USUAL, WITH THE NAME OF THE TABLE TO INSERT INTO

• CREATE THE TABLE AS USUAL, THEN INSERT DATA USING A SUBQUERY

```
CREATE TABLE EmailAddresses
(
Email VARCHAR(30),
Category VARCHAR(10)
);

INSERT INTO EmailAddresses
SELECT distinct Email,"Student" AS Category FROM
Students;
```

THIS INSERT STARTS OFF AS USUAL, WITH THE NAME OF THE TABLE TO INSERT INTO

- CREATE THE TABLE AS USUAL, THEN INSERT DATA USING A SUBQUERY

```
CREATE TABLE EmailAddresses
(
Email VARCHAR(30),
Category VARCHAR(10)
);

INSERT INTO EmailAddresses
SELECT distinct Email,"Student" AS Category FROM
Students;
```

BUT WHAT FOLLOWS IS A QUERY

• CREATE THE TABLE AS USUAL, THEN INSERT DATA USING A SUBQUERY

```
CREATE TABLE EmailAddresses
(
Email VARCHAR(30),
Category VARCHAR(10)
);
```

BUT WHAT FOLLOWS IS A QUERY

ALL THAT'S NEEDED IS FOR THE QUERY TO MATCH THE TABLE IN THE NUMBER AND TYPE OF COLUMNS

```
INSERT INTO EmailAddresses
SELECT distinct Email,"Student" AS Category FROM
Students;
```

BUT SUBQUERIES ARE FAR MORE CONVENIENT TO POPULATE A TABLE FROM EXISTING TABLES IN THE DATABASE

WE COULD EITHER -

- CREATE THE TABLE AS USUAL, THEN INSERT DATA USING A SUBQUERY
- CREATE THE TABLE AND POPULATE DIRECTLY USING A SUBQUERY

- # CREATE THE TABLE AND POPULATE DIRECTLY USING A SUBQUERY

```
CREATE TABLE EmailAddresses
(
Email VARCHAR(30),
Category VARCHAR(10)
)
AS
SELECT distinct Email,"Student" AS
Category FROM Students;
```

• CREATE THE TABLE AND POPULATE DIRECTLY USING A SUBQUERY

```
CREATE TABLE EmailAddresses
(
Email VARCHAR(30),
Category VARCHAR(10)
)
AS
SELECT distinct Email,"Student" AS Category
FROM Students;
```

TABLE DEFINITION

QUERY

- CREATE THE TABLE AND POPULATE DIRECTLY USING A SUBQUERY

```
CREATE TABLE EmailAddresses
(
Email
Categ
)
AS
SELECT distinct Email,"Student" AS
Category FROM Students;
```

UNFORTUNATELY, THIS DOESN'T WORK IN HIVE-2.0.0

- CREATE THE TABLE AND POPULATE
  DIRECTLY USING A SUBQUERY

CREAT
(
Email
Category VARCHAR(10)
)
LINKED BY 'AS' QUERY

WHEN YOU RUN THIS QUERY, YOU GET THIS RESULT

```
hive> CREATE TABLE EmailAddresses
    > (
    > Email VARCHAR(30),
    > Category VARCHAR(10)
    > )
    > AS
    > SELECT distinct Email,"Student" AS Category FROM Students;
FAILED: SemanticException [Error 10065]: CREATE TABLE AS SELECT command cannot specify the list of columns for the targe
t table
```

- CREATE THE TABLE AND POPULATE DIRECTLY USING A SUBQUERY

CREAT
(
Email
Category VARCHAR(10)
)

LINKED BY 'AS' QUERY

THIS COMMAND IS CALLED "CREATE-TABLE-AS-SELECT"(CTAS) COMMAND

```
hive> CREATE TABLE EmailAddresses
    > (
    > Email VARCHAR(30),
    > Category VARCHAR(10)
    > )
    > AS
    > SELECT distinct Email,"Student" AS Category FROM Students;
FAILED: SemanticException [Error 10065]: CREATE TABLE AS SELECT command cannot specify the list of columns for the targe
t table
```

FROM HIVE DOCUMENTATION

Create Table As Select (CTAS)

Tables can also be created and populated by the results of a query in one create-table-as-select (CTAS) statement. The table created by CTAS is atomic, meaning that the table is not seen by other users until all the query results are populated. So other users will either see the table with the complete results of the query or will not see the table at all.

There are two parts in CTAS, the SELECT part can be any SELECT statement supported by HiveQL. The CREATE part of the CTAS takes the resulting schema from the SELECT part and creates the target table with other table properties such as the SerDe and storage format.

CTAS has these restrictions:

- The target table cannot be a partitioned table.
- The target table cannot be an external table.
- The target table cannot be a list bucketing table.

Example:

```
CREATE TABLE new_key_value_store
   ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"
   STORED AS RCFile
   AS
SELECT
FROM key
SORT BY
```

LET US SEE SOMETHING WHICH ACTUALLY WORKS

- # CREATE THE TABLE AND POPULATE DIRECTLY USING A SUBQUERY

TABLE DEFINITION

```
CREATE TABLE EmailAddresses
AS
SELECT distinct
Email,"Student" AS Category
FROM Students;
```

- CREATE THE TABLE AND POPULATE DIRECTLY USING A SUBQUERY

```
CREATE TABLE EmailAddresses
AS
SELECT distinct Email, 'Student', AS
Cat
UNI
SEL                                  S
Cat
```

TABLE DEFINITION

SKIP THE COLUMN DEFINITIONS AND IT WILL WORK

BUT SUBQUERIES ARE FAR MORE CONVENIENT TO POPULATE A TABLE FROM EXISTING TABLES IN THE DATABASE

WE COULD EITHER -

- CREATE THE TABLE AS USUAL, THEN INSERT DATA USING A SUBQUERY
- CREATE THE TABLE AND POPULATE DIRECTLY USING A SUBQUERY

SUBQUERIES ARE FAR MORE CONVENIENT TO POPULATE A TABLE FROM EXISTING TABLES IN THE DATABASE

BUT IN REALITY, QUERIES ARE PRETTY PLUG-AND-PLAY

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

WE COULD CALCULATE THE UNION OF 2 QUERIES

WE COULD USE A SUBQUERY TO POPULATE A TABLE VIA INSERT

BUT IN REALITY, QUERIES ARE PRETTY PLUG-AND-PLAY

WE COULD USE ONE QUERY INSIDE ANOTHER (VIA SUBQUERIES)

WE COULD CALCULATE THE UNION OF 2 QUERIES

WE COULD USE A SUBQUERY TO POPULATE A TABLE VIA INSERT