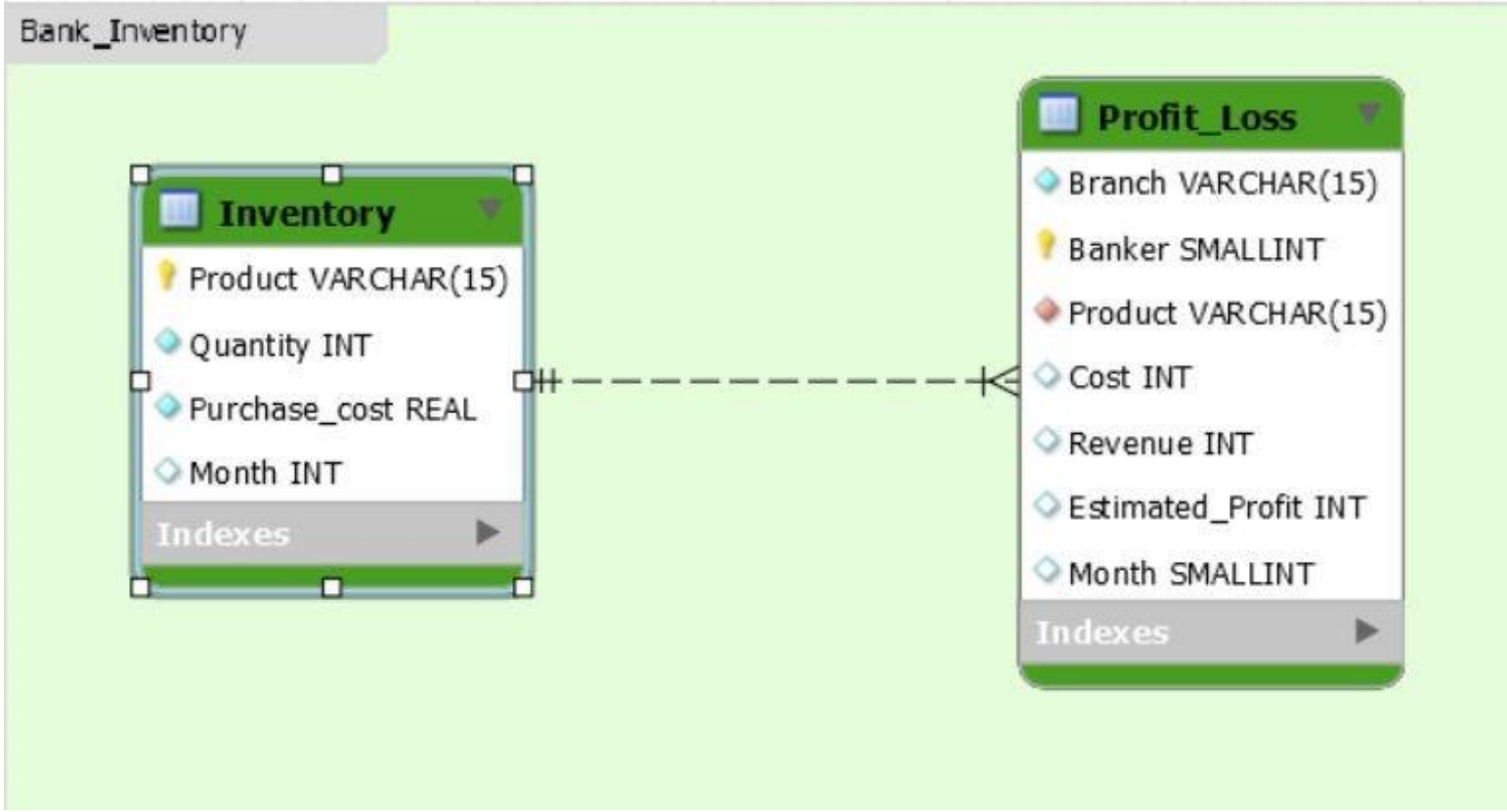
Subqueries & Query Expressions

Agenda

- Introduction to Subquery
- Properties and benefits
- Subquery using
  - where clause
  - Outer reference
  - Existence & Quantified Test
  - From Clause
- Nested Subqueries

E-R Diagram


Bank_Inventory
Inventory
Product VARCHAR(15)
Quantity INT
Purchase_cost REAL
Month INT
Indexes
Profit_Loss
Branch VARCHAR(15)
Banker SMALLINT
Product VARCHAR(15)
Cost INT
Revenue INT
Estimated_Profit INT
Month SMALLINT
Indexes

E-R Diagram

Table	Columns	Description
INVENTORY	Product	Bank has products which are purchased from vendor and selling them to customers. **Products :** Plastic visa debit/credit cards .
	Quantity	Quantity purchased from Vendor
	Purchase _cost	Purchase cost of product from vendor
	Month	Month during which inventory maintained

E-R Diagram

Table	Columns	Description
PROFIT_LOSS	Branch	Bank has branches across different locations
	Banker	Salesperson who is said to be banker
	Product	Product details maintained with *banker* at different branches
	Cost	Cost includes purchase cost + service charges + maintenance charges of the product at branch
	Revenue	Revenue generated from selling the product
	Estimated_Profit	Estimated Profit is expected but it may not be the actual calculated profit.

E-R Diagram

- All these tables can be found in the ‘module 7 tables.sql’ and would be required for this session

Subquery

- A *select* query when it is nested in anothermain query, then it is called as a subquery
- Like joins, subqueries need common key columns for joining with main queries
- Subqueriesare otherwise called as virtual table enclosed with an independent business logic
- Subqueriesexecute independently (Depends on Types) and share its results with the main query, so that the complexity reduces while writing the queries

Subquery - Benefits

- Subquery separates the complex business logic from the main query
- It is easy to debug an individual subquery instead of large and complex main query where more tables and columns are used
- Subqueries improve the performance when they are used in a better way
- Subqueries can be written anywhere in the SELECT clause, FROM clause and WHERE clause of another SQL query, however, the constraints of clauses are applied while using subqueries

Subquery

- Subqueries are of different types and can be used in different ways according to business logic

Type	Properties
Non Correlated Subquery	**Single Row Subquery** -Returns a single value and feeds to main query **Multiple Row Subquery** -Returns more number of rows
Correlated Subquery	the subquery is dependent on outer query for retrieving each record

Subqueries in Where Clause

Subquery in Where Clause

Subqueries can be written in WHERE clause of another query by using operators

- Multi-row operators perform comparison operations on multiple rows returned by subquery includes EXISTS, IN , ANY and ALL

- Single-row operators perform comparison operations on single row returned by subquery.They can also use single-row comparison operators as <, >, =

Subquery Search Condition

Subquery Comparison Test

- Sub Queries' results are dynamic rather than constant
- That means, developer will not manually pass any input in order to fetch the records, the dynamic input is retrieved from subquery
- These subqueries when used in WHERE clause, uses its dynamic input in the condition and then fetches the records

Subquery Comparison Test

- In the next example:
  - The main query tries to return transaction details during *recent* Holiday
  - Developer usually do not need to know which is the recent holiday
- The subquery is independently executed andreturns a single (scalar) value : “2020-04-24”
- This date is dynamically returned by subquery, and it can be any other date as per database table entries

Subquery Comparison Test

```
SELECT
    Acct_Num,
    Tran_Amount,
    Tran_Date
FROM
    TRANSACTION
WHERE Tran_Date = ( SELECT MAX(event_dt )
                    FROM MESSAGE
                    WHERE Event = 'Holiday'
                  )
```

Acct_Num	Tran_Amount	Tran_Date
4000-1956-2001	-3000.00	2020-04-24
4000-1956-5102	-6500.00	2020-04-24
5000-1700-9911	2000.00	2020-04-24

- In this example, WHER clause expects E (single) value an sca subquery result that is : 2020-04-24

Subqueries in Where Clause - Syntax using IN clause

- IN operator is well used when the main query searches all of the multiple rows returned by subquery

- Here in the example, the sub-query of Transaction is independently executed first and then return the results to main ACCOUNT query

- Later, the main ACCOUNT query searches all of the rows returned from subquery to check if any transactions are done via ATM machines

Subquery Set Membership Test

- In the next example:
  - The query tries to fetch all bank accounts and their balance amount when they had withdrawn the amount via ATMs machine only
  - Initially, the subquery is executed independently and returns the result set of “multiple” Account Numbers (***same datatype***) with ATM transactions
  - The “IN” operator acts like a multi-row operation refer to subquery result set to obtain balance amount

Subqueries in Where Clause - Syntax using IN clause

```
SELECT A.Acct_Num,
       A.Balance,
       A.Acct_type,
       A.Acct_status
FROM   ACCOUNT A
Where  A.Acct_Num IN
          ( SELECT T.Acct_Num
            FROM
            TRANSACTION T
            WHERE T.Channel = 'ATM withdrawal' )
```

*Account and Transaction Tables are the ones used from previous session

Acct_Num	Balance	Acct_type	Acct_status
4000-1956-3456	200000	SAVINGS	ACTIVE
4000-1956-5102	300000	SAVINGS	ACTIVE

Subquery Set Membership Test

```
SELECT A.Acct_Num,
       A.Balance,
       A.Acct_type,
       A.Acct_status
FROM ACCOUNT A
WHERE A.Acct_Num IN
(
 SELECT Acct_Num FROM Transaction
 WHERE Channel = 'ATM withdrawal'
)
```

Acct_Num	Balance	Acct_type	Acct_status
4000-1956-3456	200000	SAVINGS	ACTIVE
4000-1956-5102	300000	SAVINGS	ACTIVE

- Here, multiple values are provided to the main query with the help of IN operator
- Based on the values returned by subquery, the outer Q\query will

Subquery Quantified Test using ANY

- In the next example:
  - The main Query tries to retrieve all of the given account transactions if at least one transaction is happened beyond any of the holidays
  - “>” or “<” range operator along with “ANY” walks through of all of the account transactions
- ANY quantifies that if atleast one transaction is beyond holiday then the main Query returns all the transaction records of the account

Subquery Quantified Test

```
SELECT
    Acct_Num,
    Tran_Date
FROM TRANSACTION
WHERE Acct_Num=     '4000-1956-2001’ AND   Tran_Date > ANY
(
SELECT Event_dt
FROM Message
WHERE event = 'Holiday'
)
```

Subquery Quantified Test

Result of subquery

```
#subquery
SELECTEvent_dt
FROM Message
WHERE event = 'Holiday'
```

Event_dt
2020-02-19
2020-03-16
2020-04-24

Result of main query

```
#main query
SELECT Acct_Num,Tran_Date
FROM TRANSACTION
WHERE Acct_Num   =   '4000-1956-2001'
```

Acct_Num	Tran_Date
4000-1956-2001	2020-02-14
4000-1956-2001	2020-01-19
4000-1956-2001	2020-03-23
4000-1956-2001	2020-04-24
4000-1956-2001	2020-04-26
4000-1956-2001	2020-03-15

Subquery Quantified Test

```
SELECT
    Acct_Num,
    Tran_Date
FROM TRANSACTION
WHERE Acct_Num=     '4000-1956-2001' AND   Tran_Date > ANY
(
SELECT Event_dt
FROM Message
WHERE event = 'Holiday'
)
```

Acct_Num	Tran_Date
4000-1956-2001	2020-03-23
4000-1956-2001	2020-04-24
4000-1956-2001	2020-04-26
4000-1956-2001	2020-03-15

- Here, subquery returns holiday days from Feb,2020 to April,2020
- Main query evaluates the least Holiday : Feb,2020 using ANY so that it retrieves as many transactions as possible

do it YOURSELF

Replace ANY with IN and compare the results: Few record gets filtered and the meaning of the result gets changed

```
SELECT
    Acct_Num,
    Tran_Date
FROM TRANSACTION
WHERE Acct_Num=     '4000-1956-2001’ AND   Tran_Date IN
(
SELECT Event_dt
FROM Message
WHERE event = 'Holiday'
)
```

Acct_Num	Tran_Date
4000-1956-2001	2020-04-24

Subquery Quantified Test using ALL

- Unlike ANY, ALL is used to quantify the infinite results rather than the sample

- In such cases, each of the main query records satisfies the condition when it matches with every/all of results of subquery

Sub-Query Quantified Test

- In next example:
  - bank transactions are retrieved by checking ALL of the bank holidays
  - “>” or “<” range operator along with “ALL” walks through all of the bank holidays
- ALL will ensure the main query to check “each and every” subquery result

Quantified Test using “ALL” with “<” operator

```
SELECT Acct_Num,
       Tran_Date
FROM TRANSACTION
WHERE Acct_Num = '4000-1956-2001' AND Tran_Date < ALL
                  ( SELECT Event_dt
                    FROM   MESSAGE
                    WHERE event = 'Holiday'
                  )
```

Acct_Num	Tran_Date
4000-1956-2001	2020-02-14
4000-1956-2001	2020-01-19

- Here, ALL the results of subquery are considered by main query
- Hence it checks for the transactions before “all of those holidays”(Refer slide 38 for details of the Queries)

Quantified Test using “ALL” with “>” operator

```
SELECT Acct_Num,
       Tran_Date
FROM TRANSACTION
WHERE Acct_Num = '4000-1956-2001' AND Tran_Date > ALL
                  ( SELECT Event_dt
                    FROM   MESSAGE
                    WHERE event = 'Holiday'
                  )
```

Acct_Num	Tran_Date
4000-1956-2001	2020-04-26

- Here, ALL the results of subquery are considered by main query
- Hence it checks for the transactions after all of those holidays

Outer References

Outer References

- A Sub-query is executed independently andreference its recordsas a derived table in the FROM clause of a Main Query

- It is beneficial in many ways instead of using full table
  - *Only Selected rows and columns are used in the FROM clause*
  - Subqueries divides the complex logic from the main Query in the FROM clause

Outer References

- In the next example:
  - The main query retrieves results from ACCOUNT table,
  - The Subquery retrieves results from TRANSACTION table which is independently executed, and also filters the records with condition that transaction amount > 23000
  - In the FROM clause, the Subquery joins the filtered records of TRANSACTION table, and joins with ACCOUNT table
  - The WHERE clause ensures to join the ACCOUNT and the “Subquery TRANSACTION table” results to give the desired output

Outer References

```
SELECT A.Acct_Num,
       Sub_T.Tran_Amount
FROM   ACCOUNT A ,
       (
         SELECT Acct_Num,
                Tran_Amount
         FROM Transaction
         Where Tran_amount > 23000
       )  Sub_T
WHERE A.Acct_Num = Sub_T.Acct_Num
```

Acct_Num	Tran_Amount
5000-1700-6091	40000.00
4000-1956-9977	50000.00

Subquery Existence Test

- The subquery is examined by each record of the main query using common key columns
- The common key column should be same in main query and subquery
- The subquery returns true (1) or false (0) when its conditions are satisfied with main query input column values

Subquery Existence Test using EXISTS

- EXISTS operator is used when a record in the main query has one or more matching records in the subquery result set

- In the next example the subquery of Transaction is executed for multiple times for each account in ACCOUNT table

Subquery Existence Test using EXISTS

- The query checks each account has done at least one transaction in Transaction table, otherwise, do not return those account details

```
SELECT A.Acct_Num,
          A.Balance  ,
       A.acct_type,
          A.acct_status
FROM   ACCOUNT A
WHERE Exists (SELECT 'yes'
              FROM Transaction T
              WHERE T.Acct_Num = A.Acct_Num)
```

Subquery Existence Test using EXISTS

Acct_Num	Balance	acct_type	acct_status
4000-1956-3456	200000	SAVINGS	ACTIVE
4000-1956-2001	400000	SAVINGS	ACTIVE
5000-1700-6091	7500000	FIXED DEPOSITS	ACTIVE
4000-1956-3401	655000	SAVINGS	ACTIVE
4000-1956-5102	300000	SAVINGS	ACTIVE
4000-1956-5698	455000	SAVINGS	ACTIVE
4000-1956-9977	7025000	FIXED DEPOSITS	ACTIVE
9000-1700-777...	0	CREDITCARD	INACTIVE
5000-1700-7755	NULL	SAVINGS	INACTIVE
5000-1700-9911	2000	SAVINGS	INACTIVE

- In subquery, ‘yes’ is a constant value
- Internally, the subquery returns true or false when the record is available

Subquery Quantified Test

- Quantify test means “validating many records ”

- Several times, multiple records are returned from subquery. Usually there is a one-many relationship between main query and subquery

- In such cases, the main query satisfies the condition if at least “one single record” of main Query matches with matches with “any of the multiple records” of subquery.

Subquery in FROM Clause

Subquery in FROM Clause

- Subqueries in the From Clause act like a view table derived from conditions

- Subqueries as a derived form can inline with JOINs to form a main query

Subquery in FROM Clause

- Consider the following query, the main query JOINS with a Subquery
- Initially, the subquery is executed and retrieves the occurre during transactions holidays d
- These transactions are then used by main query using JOIN clause to retrieve Account and Transaction details

```
SELECT A.Acct_Num, Tran.Tran_Amount, Tran.Tran_Date
FROM   ACCOUNT  A
JOIN(SELECT Acct_Num, Tran_Amount, Tran_Date
       FROM   Transaction, Message
       WHERE Tran_Date = Event_dt
       AND event= 'Holiday'
    ) Tran
ON Tran.Acct_Num = A.Acct_Num
```

Subquery in FROM Clause

Output:

Acct_Num	Tran_Amount	Tran_Date
4000-1956-2001	-3000.00	2020-04-24
5000-1700-6091	40000.00	2020-02-19
4000-1956-5102	-6500.00	2020-04-24
5000-1700-9911	2000.00	2020-04-24

Nested Subqueries

Nested-Subqueries

- A Subquery can be embedded or ***nested*** in another Subquery
- Any SELECT query supports multiple subqueries nesting within one another
- So each nested subquery executes independently and pass its result to next level subquery
- E.g : A SELECT query consists of Subquery - A ,
  - then Subquery - A consists of Subquery - B, and
  - then the Subquery - B can consists of Subquery - C, and so on

MYSQL executes the low level subquery first and pass its intermediate result to its next immediate subquery.

Here in Eg, Subquery-C is executed first and pass its result to Subquery-B, and so on

Nested-Subqueries - Example

- In the next example:
  - Initially, low level subquery (Evnt) returns a list of Holiday dates and pass it to its next level subquery
  - Next, the subquery (Tran_Evnt) uses Holiday dates returned by (Evnt) and returns transactions during the event to next level query
  - Finally, the main query uses records of (Tran_Evnt) subquery and joins with Account table to return Account details and Tran_Evnt subquery details.

Nested-Subqueries - Example

- Nested subquery:

```
SELECT Cust_id, A.Acct_Num, A.Balance, Tran_Evnt.Tran_Amount,
Event
FROM ACCOUNT A
JOIN (SELECT Acct_Num, Tran_Amount, Event


        FROM   Transaction
        JOIN (SELECT Event, Event_dt
              FROM Message) Evnt
        ON Tran_Date = Evnt.Event_dt) Tran_Evnt
ON    A.Acct_Num = Tran_Evnt.Acct_Num
```

Cust_id	Acct_Num	Balance	Tran_Amount	Event
123002	4000-1956-2001	400000	-3000.00	Holiday
123004	5000-1700-6091	7500000	40000.00	Holiday
123005	4000-1956-5102	300000	-6500.00	Holiday
123009	5000-1700-9911	2000	2000.00	Holiday

Nested-Subqueries - Example

- The final result consist of:
  - Message table details from subquery - (Evnt)
  - Transactions details from subquery - (Tran_Evnt)

Set Membership Test using Nested Subqueries

Set Membership Test Using Nested Subquery

- A series of subqueries nested in one another recursively are referred by common key columns using IN operator
- The Low level nested subquery is executed first and then referred by IN operator by its next level subquery
- This process is repeated until the result of nested queries reaches to main query

Set Membership Test Using Nested Subquery

```
SELECT Acct_Num, Tran_Amount, Channel, Tran_date
From Transaction
WHERE Acct_Num IN (SELECT Acct_Num
                   FROM ACCOUNT
                   WHERE Balance > 0 AND Cust_Id IN (SELECT
Cust_Id
                                                     FROM CUSTOMER
                                                    )
                    )
```

- First, the innermost nested subquery consists of customer details which is then referred by its immediate account subquery using IN operator
- Secondly, the Account table details are evaluated and the final account numbers to **main** Transaction query

Set Membership Test Using Nested Subquery

```
SELECT Acct_Num, Tran_Amount, Channel, Tran_date
From Transaction
WHERE Acct_Num IN (SELECT Acct_Num
                   FROM ACCOUNT
                   WHERE Balance > 0 AND Cust_Id IN (SELECT
Cust_Id
                                                     FROM CUSTOMER
                                                    )
                    )
```

- Finally, the transaction details are retrieved by checking in ACCOUNT and CUSTOMER table using IN operator and nested subqueries

Set Membership Test Using Nested Subquery

Output

Acct_Num	Tran_Amount	Channel	Tran_date
4000-1956-3456	-2000.00	ATM withdrawal	2020-01-13
4000-1956-2001	-4000.00	POS-Walmart	2020-02-14
4000-1956-2001	-1600.00	UPI transfer	2020-01-19
4000-1956-2001	-6000.00	Bankers cheque	2020-03-23
4000-1956-2001	-3000.00	Net banking	2020-04-24
4000-1956-2001	-2970.00	Net banking	2020-04-26
4000-1956-5102	-6500.00	ATM withdrawal	2020-04-24
4000-1956-5102	-3600.00	ATM withdrawal	2020-04-25
4000-1956-2001	23000.00	cheque deposit	2020-03-15
5000-1700-6091	40000.00	ECS transfer	2020-02-19
4000-1956-3401	8000.00	Cash Deposit	2020-01-19
4000-1956-5102	-6500.00	ATM withdrawal	2020-03-14
4000-1956-5698	-9000.00	Cash Deposit	2020-03-27
4000-1956-9977	50000.00	ECS transfer	2020-01-16

- The final result consists of only Transaction table details
- Unlike JOINS , set operator (IN) cannot bring the column results from nested subqueries

Set Comparison Test Using Nested Subquery

Set Comparison Test Using Nested Subquery

- In a recursive nested subqueries, comparison operators includes “>” , “<” , are used to compare a range of subquery results and filters the outer query results

Set Comparison Test Using Nested Sub-Query

```
SELECT*
FROM CUSTOMER
WHERE Cust_Id IN (SELECT Cust_Id
                  FROM ACCOUNT
                  WHERE Acct_Num IN
                          (SELECT Acct_Num
                           FROM Transaction
                           WHERE Tran_Date > (SELECT MAX(Event_dt )
                                              FROM Message)
                           )
                  )
```

- Here, the set operators (IN) and “>” are used for evaluating the nested subqueries
- The low level nested subquery returns a scalar(single) value - recent Holiday

Set Comparison Test Using Nested Sub-Query

```
SELECT*
FROM CUSTOMER
WHERE Cust_Id IN (SELECT Cust_Id
                  FROM ACCOUNT
                  WHERE Acct_Num IN
                         (SELECT Acct_Num
                          FROM Transaction
                          WHERE Tran_Date > (SELECT MAX(Event_dt )
                                             FROM Message)
                          )
                 )
```

- “>” operator expects only single value, hence we applied MAX function to return recent Holiday
- Later, the **IN** operator evaluates transaction details after the recent Holiday

Set Comparison Test Using Nested Sub-Query

Cust_Id	Name	Address	State	Phone
123002	George	194-6,New brighton	MN	189761700
123005	Jacob	325-7, Mission Dist	SFO	1897637000

- The output displays the customer details from main query, and the result is: customers who did

***Note**: IN operator cannot return the column values unlike JOINS nested subqueries*

Subqueries with the ‘WITH’ Clause

Subqueries with the ‘WITH’ Clause

- Subqueries written in the WITH Clause plays a factoring role
- Factoring means , the WITH Clause serves subquery results in the Main-Query wherever it is referenced
- Once the WITH clause is executed , the same subQuery result can be used for multiple times without execution
- WITH clause plays a major role in complex queries when there is a need for calling the subqueries for multiple times

Subqueries with the 'WITH' Clause

```
WITH    I_RATE AS
         (SELECT Acct_type, rate FROM Interest
     )
SELECT A.Acct_num, I_RATE.Acct_type,
I_RATE.rate
FROM ACCOUNT   A
JOIN I_RATE
ON A.Acct_type = I_RATE.Acct_type
```

- The WITH clause Query is referenced with I_RATE acts like a re-usable dataset and can be invoked for any number of times in the FROM clause or any other Sub-queries

I_RATE derived table to

Subqueries with the 'WITH' Clause

```
WITH    I_RATE AS
         (SELECT Acct_type, rate FROM Interest
     )
SELECT A.Acct_num, I_RATE.Acct_type,
I_RATE.rate
FROM ACCOUNT A
JOIN I_RATE
ON A.Acct_type = I_RATE.Acct_type
```

- So that Each account will be assigned with interest ratebased on its Acct_type including SAVINGS and RECURRING DEPOSITS

Subqueries with the ‘WITH’ Clause

Output:

Acct_num	Acct_type	rate
4000-1956-3456	SAVINGS	0.04
5000-1700-3456	FIXED DEPOSITS	0.07
4000-1956-2001	SAVINGS	0.04
5000-1700-5001	FIXED DEPOSITS	0.07
4000-1956-2900	SAVINGS	0.04
5000-1700-6091	FIXED DEPOSITS	0.07
4000-1956-3401	SAVINGS	0.04
4000-1956-5102	SAVINGS	0.04
4000-1956-5698	SAVINGS	0.04
5000-1700-9800	SAVINGS	0.04
4000-1956-9977	FIXED DEPOSITS	0.07
5000-1700-7755	SAVINGS	0.04
5000-1700-9911	SAVINGS	0.04

Subqueries in the Having Clause

Subqueries in the ‘HAVING’ Clause

- Having Clause condition filters the *grouped results* of the Main Query
- SubQuery in the Having Clause returns the derived values dynamically to satisfy the condition
- This concept helps to compare the grouping results with other grouped results.

Subqueries in the ‘HAVING’ Clause

```
Select Branch, Product, SUM(Estimated_profit) as Profit
FROM Profit_Loss
GROUP BY Branch, Product
HAVING SUM(Estimated_profit) >( SELECT
AVG(estimated_profit)
                        FROM Profit_Loss )
```

- The main query is trying to compare the branches whose actual profit (calculated) is higher than the minimum profit across the bank

- The Having Clause filters the branches with least earned profit with help of subquery result

Subqueries in the ‘HAVING’ Clause

Output:

Branch	Product	Profit
Delhi	SuperSave	20050070
Delhi	SmartSav	30050070
Hyd	SmartSav	60050224
Banglr	SmartSav	30000154
Hyd	BusiCard	35110140

- The products that are returned with profit that is more than the average profit of ALL products at various branches
- The objective is to know which products returned high profit than the average profit of all products profit

Correlated Subqueries in the ‘HAVING’ Clause

- In our previous example, subquery is independently and dynamically executed in HAVING clause

- Here , the SubQuery in the Having Clause can also ***correlates*** with non grouping fields selected in the Main query

- So that each result of the main query is conditionally retrieved based on the each record of subquery

Correlated Subqueries in the ‘HAVING’ Clause

```
SELECT Branch, P1.Product, SUM(Estimated_profit) AS Profit
FROM PROFIT_LOSS P1
GROUP BY Branch, P1.Product
HAVING SUM(Estimated_profit) > (SELECT AVG(Estimated_profit)
                                FROM PROFIT_LOSS P2
                                WHERE P2.Product = P1.Product)
```

- Here, for each individual product, branch level profit is verified with overall organization level profit
- The main query retrieves total- profit of each product at branch
- Since the subquery is correlated with main query, the subquery is executed with an input from main query, and returns average of profit per each product

Correlated Subqueries in the ‘HAVING’ Clause

```
SELECT Branch, P1.Product, SUM(Estimated_profit) AS Profit
FROM PROFIT_LOSS P1
GROUP BY Branch, P1.Product
HAVING SUM(Estimated_profit) > (SELECT AVG(Estimated_profit)
                                FROM PROFIT_LOSS P2
                                WHERE P2.Product = P1.Product)
```

- The execution of the subquery is repeated for all the corresponding inputs from main query
- Having clause will conditionally filters the records of main query

Correlated Subqueries in the ‘HAVING’ Clause

Subquery result only:

```
SELECT p2.product, AVG(Estimated_profit) FROM
PROFIT_LOSS P2
group by p2.product;
```

product	AVG(Estimated_profit)
SuperSave	20050070.0000
SmartSav	20016741.3333
EasyCash	10000077.0000
BusiCard	17555070.0000

- Here, The subquery returns the “average of estimated profit" per each “product” across the company. (includes ALL the branches)

- The result is then compared with main Query

Correlated Subqueries in the ‘HAVING’ Clause

Example-2
Output:

Branch	Product	Profit
Delhi	SmartSav	30050070
Delhi	EasyCash	10050077
Hyd	SmartSav	60050224
Banglr	SmartSav	30000154
Hyd	BusiCard	35110140

- The main query retrieves “SUM of profit” of *each product* at branch level
- The results returned from main Query are then filtered in the HAVING clause
- In Having clause, the branch level profit is verified against organization level average profit

Summary

Summary

- Subqueries are written to separate the complexity of logic
- Subqueries performs the same way like JOINS but improves the performance of overall query execution
- Subqueries written on tables with less records gives better performance
- Subqueries can be written anywhere in the SELECT,FROM ,WHERE and HAVING clauses of outer SQL query
- Subqueries can be written in another subquery
- Easy for debugging the subquery

Summary

- Always use WITH clause when there is a necessary to invoke the subquery for multiple times in differents part of the query
- Subqueries correlates with outer query by joining the common key columns
- EXISTS Operator validates the subquery results to return true or false and filters the main query results
- IN Operator lookup on the values returned by Subquery and pass the outer query condition
- ANY/ALL Operators quantifies by walking through all of the subquery results with outer query results.

Thank You