Multiple Table Queries

Agenda

- Introduction to Joins with ER Diagram
- Types of Joins
  - Simple Joins
  - Natural Joins
  - Equi Joins
  - Non Equi Join
  - Self Join
  - Left & Right Join
  - Inner & Outer Join

Introduction

- JOIN clause allows to retrieve the data from two or more tables having related data
- These relations are established or identified with the help of key attributes
- In other words, Facts and Dimensional tables are joined using Key columns to produce an output of records with more meaningful relationships between each data fields

Fact table consists of real time data, usually static in nature, and do not change frequently.
E.g Customer data is single and do not change all the time

Dimension tables consists of data that is changed more frequently than the fact tables.
E.g : Transaction data which is frequently updated

Introduction

- Consider the following two tables Customer and Account:

Customer	Account
cust_id	cust_id
name	savings
address	deposits
contact_num	credit_card

- JOIN clause will help to establish relationship between the two tables CUSTOMER and ACCOUNT and present all the details of accounts held by one single customer

did you know?

With JOINs, queries avoids a lot of repetition of the data and able to present it in a more systematic way

Leaves			
leave_ id	emp_id	days	reason
1	2	1	Comp. Leave
2	4	2	Sick leave
3	5	1	Sick leave

+

Employee		
emp_id	emp_name	Department
1	Steve J.	KPO
2	Jacob Kia	Data Hub
3	Daisy Boggs	KPO
4	Joe Cruz	R&D
5	Nina Hannah	R&D

=

Joined Table			
emp_id	emp_name	days	reason
2	Jacob Kia	1	Comp. Leave
4	Joe Cruz	2	Sick leave
5	Nina Hannah	1	Sick leave

Introduction

- In a bank database, Strategic mgt. analyzes the customers using their debit-card transactions occurred through various channels using Joins to understand customer expectation

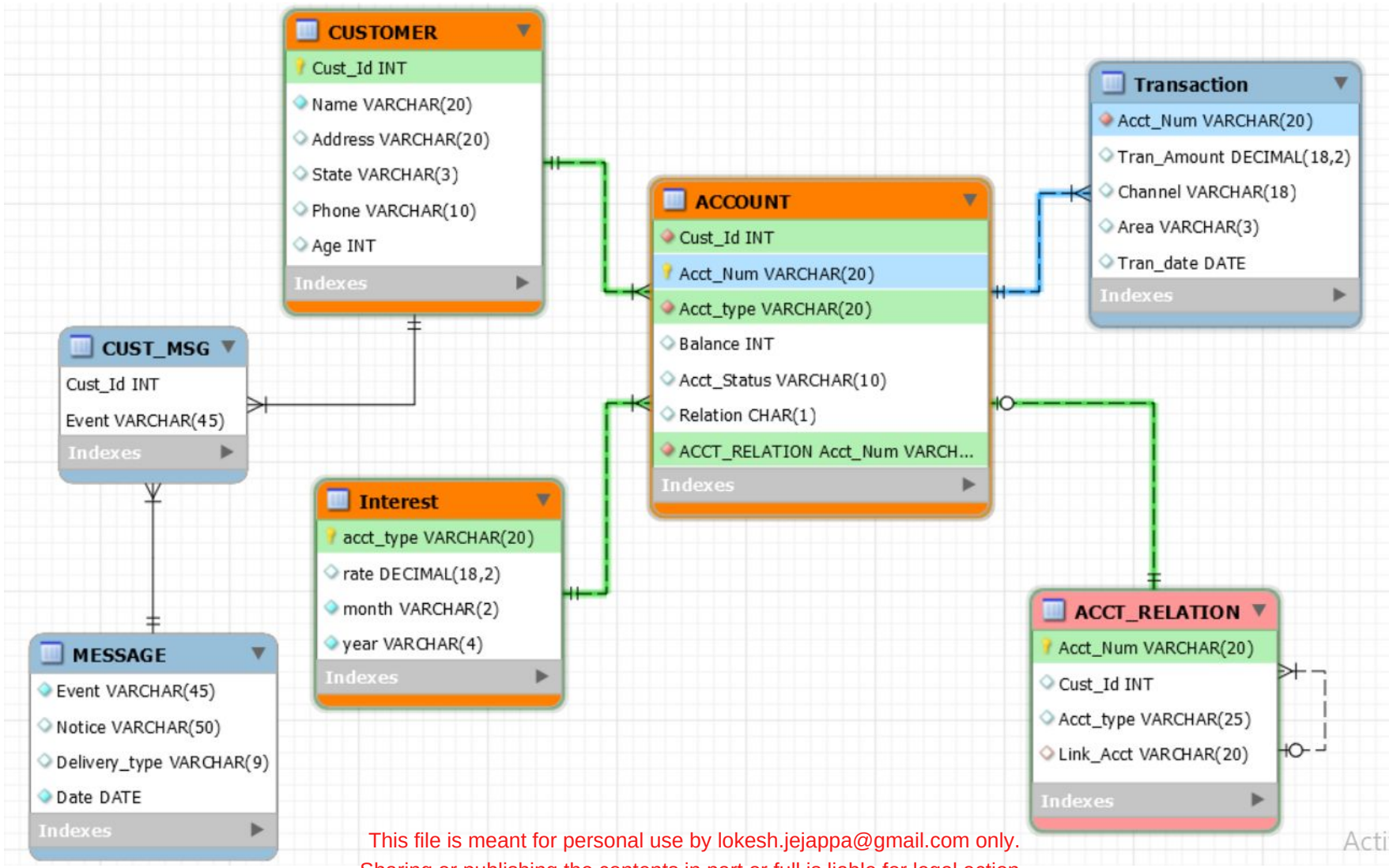
- Few relationships between customers , accounts and transactions table are below:
  - How many accounts are maintained by each customer?
  - Different modes of customer payments?
  - Customer using Smart cash (digital) payment via various channels including UPI, IVR, point_of_sale?

Introduction

- The ISO standard defines following JOIN clauses that are commonly used by all.
  - Self - Join or equi - Join
  - Non - Equi Join
  - Natural Join
  - Inner Join
  - Left Outer Join
  - Right Outer Join

ER Diagram for the study

- Please consider the following ER diagram for our study:



ER Diagram

CUSTOMER	Cust_id	Customer is Uniquely considered in the bank. Hence defined as Primary key.
	Name	Customer Name
	Address	Address of a customer.
	State	State code of customer residence
	Phone	Personal phone details

ER Diagram

ACCOUNT	Cust_id	Customer can have multiple Accounts. Hence it is a referenced by CUSTOMER table. (Foreign Key)
	Acct_Num	Account Number is uniquely represented in bank. (Primary key)
	Acct_type	Type of Account: *Savings, Deposit & Credit card*
	Balance	Balance Amount that is maintained in an account
	Acct_status	Status becomes ACTIVE or INACTIVE accounts over unused.
	Relation	Defines Primary (P) or Secondary (S) . E.g: A Deposit Account is Secondary relation and is always depend on primary - Savings Account.

ER Diagram

TRANSACTION	Acct_Num	Account Number (NON - unique key) . Because an account is transacted for multiple times.
	Tran_Amount	Transaction Amount in Numbers
	Channel	Mode of transfer: E.g: ATM Cash Withdrawal, Cash Deposits, Online Shopping.
	Area	In which area (state code) , the transaction taken place.
	Tran_date	Date of Transaction

ER Diagram

ACCT_RELATION	Acct_Num	Account Number is uniquely represented in bank. (Primary key)
	Cust_id	Customer Identify of any Account
	Acct_type	Type of Account: *Savings, Deposit & Credit card*
	Link_Acct	E.g: 1) A Deposit Account is *linked* to Savings Account (or) 2) A Credit card is *linked to* Savings Account. Here, Savings Account is the Link_Acct.

ER Diagram

INTERREST	Acct_type	Type of Account: *Savings, Deposit & Credit card*
	Rate	Rate of interest that is defined for each type : Savings, Deposits and Credit card.
	month	Month (1, 2,3 ,4,…. ,12)
	Year	Defines the year . (2020, 2021 ,...)
MESSAGE	Event	Any upcoming Event occuring in the bank.
	Notice	Pre- defined Notice message needs to be delivered to a customer when a sudden event occurs.
	Delivery_Mode	Mode of delivering a message to customer. E.g : message via phone or e-mail.
	Date	Date of the event

ER Diagram

- All these tables can be found under ‘module 6 tables.sql’ file which will be required for this session

Two Table Query Example

Two Table Query

- Business needs more complex reports than simple reports to understand relationships between different facts and dimensional tables

- The next example shows how two tables are accessed via SELECT query and will drill down how it works

A two table query example

- CUSTOMER and ACCOUNT are two different tables which are allowed to access via SELECT query to see multiple combinations of data between them

```
SELECT * FROM  CUSTOMER, ACCOUNT;
```

- The output gives the multiple combinations of data between CUSTOMER and ACCOUNT

Cust_Id	Name	Address	State	Phone	Cust_Id	Acct_Num	Acct_Type	Balance	Acct_status	Re
123001	Oliver	225-5, Emeryville	CA	1897614500	123001	4000-1956-3456	SAVINGS	200000	ACTIVE	P
123002	George	194-6,New brighton	MN	189761700	123001	4000-1956-3456	SAVINGS	200000	ACTIVE	P
123003	Harry	2909-5,walnut creek	CA	1897617866	123001	4000-1956-3456	SAVINGS	200000	ACTIVE	P
123004	Jack	229-5, Concord	CA	1897627999	123001	4000-1956-3456	SAVINGS	200000	ACTIVE	P
123005	Jacob	325-7, Mission Dist	SFO	1897637000	123001	4000-1956-3456	SAVINGS	200000	ACTIVE	P
123006	Noah	275-9, saint-paul	MN	1897613200	123001	4000-1956-3456	SAVINGS	200000	ACTIVE	P
123007	Charlie	125-1 Richfield	MN	1897617666	123001	4000-1956-3456	SAVINGS	200000	ACTIVE	P

Simple Joins

Table and Column aliases

- Table & Column aliases are useful when a SELECT query is using *similarly named columns* from two different tables
- Aliases are given for *tables* and as well as for *columns*
- It represents similar column names with different meaning in SELECT clauses
- Aliases are also given for a complete SELECT query when it is referred as a derived table, which we will come across in other sessions

Without aliases

- Without aliases , the full length of table names should be mentioned in WHERE clause conditions and in SELECT clause

```
SELECTCUSTOMER.CUST_ID,ACCOUNT.CUST_ID, NAME, ACCT_NUM
FROM  CUSTOMER, ACCOUNT
WHERE CUSTOMER.CUST_ID = ACCOUNT.CUST_ID;
```

Output:

CUST_ID	CUST_ID	NAME	ACCT_NUM
123001	123001	Oliver	4000-1956-3456
123001	123001	Oliver	5000-1700-3456
123002	123002	George	4000-1956-2001
123002	123002	George	5000-1700-5001
123003	123003	Harry	4000-1956-2900
123004	123004	Jack	5000-1700-6091

With aliases

- When the two tables, CUSTOMER & ACCOUNT, are JOINED using common key column: “cust_id” , MYSQL finds difficulty to represent the cust_id from either of these two tables. Hence it is mandatory to provide table name before the column
  - E.g : *CUSTOMER*.CUST_ID , *ACCOUNT*.CUST_ID

- In order to simplify the table reference,
  - CUSTOMER table is represented with table alias as "BC" .
  - ACCOUNT table is represented with table alias as “AC” .

Table and Column aliases

Example:

```
SELECT   BC.cust_id AS  Master_Cust_Id,
         BA.cust_id AS Account_Cust_Id,
         Name AS Customer_Name,
         Acct_Num AS Account_Number
FROM CUSTOMER BC, ACCOUNT  BA
WHERE BC.CUST_ID = BA.CUST_ID;
```

Table and Column aliases

- Here, the cust_id in CUSTOMER and ACCOUNT is referred with table aliases and column aliases

Master_Cust_Id	Account_Cust_Id	Customer_Name	Account_NUmber
123001	123001	Oliver	4000-1956-3456
123001	123001	Oliver	5000-1700-3456
123002	123002	George	4000-1956-2001
123002	123002	George	5000-1700-5001
123003	123003	Harry	4000-1956-2900
123004	123004	Jack	5000-1700-6091
123004	123004	Jack	4000-1956-3401

While table aliases simplifies the names of tables, column alias will simplify the SELECT query columns with more meaningful representation

Parent - Child Relationship Queries

Parent - Child Relationship Queries

- A same table is referred in a SELECT query for multiple times to establish parent - child relationships
- The table when called for first time acts like a parent and the same table acts like a child when it is called for second time and so on
- This is useful when we need to find hierarchical data and relationship between two records in the same table

Parent - Child Relationship Queries

- Consider the following ACCT_RELATION table created earlier:

Cust_id	Acct_Num	Acct_type	Link_Acct
123001	4000-1956-3456	SAVINGS	
123001	5000-1700-3456	FIXED DEPOSITS	4000-1956-3456
123002	4000-1956-2001	SAVINGS	
123002	5000-1700-5001	FIXED DEPOSITS	4000-1956-2001
123003	4000-1956-2900	SAVINGS	
123004	5000-1700-6091	FIXED DEPOSITS	4000-1956-2900
123007	5000-1700-9800	SAVINGS	
123007	4000-1956-9977	FIXED DEPOSITS	5000-1700-9800
123007	9000-1700-7777-4321	CREDITCARD	5000-1700-9800

- The column Acct_Num holds all type of accounts. Link_Acct field shows parent relationship with the account number.

Parent - Child Relationship Queries

- The below query can be used to display the primary and secondary account types for a single account number:

```
SELECT AR1.Acct_Num    as Primary_Account,
       AR1.Acct_Type     as Primary_Acct_Type,
       AR2.Acct_Num      as Secondary_Account,
       AR2.Acct_Type     as Secondary_Acct_Type
FROM   ACCT_RELATION AR1,
      ACCT_RELATION AR2
WHERE AR1.Acct_Num = AR2.Link_Acct ;
```

Parent - Child Relationship Queries

Output:

Primary_Account	Primary_Acct_Type	Secondary_Account	Secondary_Acct_Type
4000-1956-3456	SAVINGS	5000-1700-3456	FIXED DEPOSITS
4000-1956-2001	SAVINGS	5000-1700-5001	FIXED DEPOSITS
4000-1956-2900	SAVINGS	5000-1700-6091	FIXED DEPOSITS
5000-1700-9800	SAVINGS	4000-1956-9977	FIXED DEPOSITS
5000-1700-9800	SAVINGS	9000-1700-7777-4321	CREDITCARD

- The result shows that “FIXED DEPOSITS” and “Credit card” are child accounts of the parent “savings” account

Joins with Row Selection Criteria

Joins with Row Selection Criterion

- Rows returned from a SELECT Query can also act like a **temporary** table or a **derived** table where it can be joined with any other physical table.

Joins with Row Selection Criterion - Example-1

- In the below example, the independent query in the FROM clause acts like a temporary table which in turn used like a row source data by main SELECT query

Syntax:

```
SELECT BA.Acct_num, BA.Balance FROM  ( Select * FROM ACCOUNT ) BA;
```

Output:

Acct_Num	Balance
4000-1956-3456	200000
5000-1700-3456	9400000
4000-1956-2001	400000
5000-1700-5001	7500000
4000-1956-2900	750000
5000-1700-6091	7500000
4000-1956-3401	655000

Joins with Row Selection Criterion - Example-2

- In the below example, we will understand how the row source data is used in JOINS.
- Here , the result set of an independent SELECT query results are used as a temporary table in the INNER JOIN condition and then later joined with main Query

Syntax:

```
SELECT BA.Acct_Num ,
       BA.Acct_Type,
       AT.Tran_amount
FROM   ACCOUNT  BA, ( SELECT * from TRANSACTION) AT
WHERE  BA.Acct_Num  = AT.Acct_Num;
```

Joins with Row Selection Criterion - Output

Acct_Num	Acct_Type	Tran_Amount
4000-1956-3456	SAVINGS	-2000.00
4000-1956-2001	SAVINGS	-4000.00
4000-1956-2001	SAVINGS	-1600.00
4000-1956-2001	SAVINGS	-6000.00
4000-1956-2001	SAVINGS	-3000.00
4000-1956-2001	SAVINGS	-2970.00
4000-1956-5102	SAVINGS	-6500.00
4000-1956-5102	SAVINGS	-3600.00
4000-1956-2001	SAVINGS	23000.00
5000-1700-6091	FIXED DEPOSITS	40000.00
4000-1956-3401	SAVINGS	8000.00
4000-1956-5102	SAVINGS	-6500.00
4000-1956-5698	SAVINGS	-9000.00

Multiple columns of *one* table can match with *other* tables. Here the multiple columns used for joining are said be composite joining key to reference another table.

- "Acct_NUM" *and* "Acct_type" and "*balance*" Columns in ACCOUNT table are used as *composite join key* to represent other table: TRANSACTION.

```
SELECT BA.Acct_Num, BA.Acct_type, BA.Balance, TR.Tran_Amount
FROM ACCOUNT  BA
JOIN TRANSACTION  TR
ON ( BA.Acct_Num = TR.Acct_Num AND BA.Balance > abs(TR.Tran_Amount)
    AND BA.Acct_type like '%DEPOSIT%' )
```

Output:

Acct_Num	Acct_type	Balance	Tran_Amount
5000-1700-6091	FIXED DEPOSITS	7500000	40000.00
4000-1956-9977	FIXED DEPOSITS	7025000	50000.00

- In this example, Bank is trying to know the Account balance is greater than the transaction amount, and for DEPOSIT type accounts only

Natural Joins

Natural join

- A Natural JOIN maps the rows implicitly among common columns in both the tables defined in the FROM clause

- It is best to use when there is a need for joining with *all of the common* columns between the two tables and retrieve full set of columns from both the tables.

- Developer need not be aware of the columns that are used for Joining the tables.

Natural join

- NATURAL JOIN implicitly joins CUSTOMER and ACCOUNT tables using *CUST_ID*

```
SELECT * FROM CUSTOMER NATURAL JOIN ACCOUNT
```

Output :

Cust_Id	Name	Address	State	Phone	Acct_Num	Acct_Type	Balance	Acct_status	Relation
123001	Oliver	225-5, Emeryville	CA	1897614500	4000-1956-3456	SAVINGS	200000	ACTIVE	P
123001	Oliver	225-5, Emeryville	CA	1897614500	5000-1700-3456	FIXED DEPOSITS	9400000	ACTIVE	S
123002	George	194-6,New brighton	MN	189761700	4000-1956-2001	SAVINGS	400000	ACTIVE	P
123002	George	194-6,New brighton	MN	189761700	5000-1700-5001	FIXED DEPOSITS	7500000	ACTIVE	S
123003	Harry	2909-5,walnut creek	CA	1897617866	4000-1956-2900	SAVINGS	750000	INACTIVE	P
123004	Jack	229-5, Concord	CA	1897627999	5000-1700-6091	FIXED DEPOSITS	7500000	ACTIVE	S
123004	Jack	229-5, Concord	CA	1897627999	4000-1956-3401	SAVINGS	655000	ACTIVE	P
123005	Jacob	325-7, Mission Dist	SFO	1897637000	4000-1956-5102	SAVINGS	300000	ACTIVE	P
123006	Noah	275-9, saint-paul	MN	1897613200	4000-1956-5698	SAVINGS	455000	ACTIVE	P

- The output of this query gives all columns in both the tables

Natural joins usually takes all the key columns of the first table and then tries to match with other table columns

There are certain restriction on Natural joins:

- *Natural Joins needs curated data in the common columns which means all the common columns should have unique data without duplicates*
- *With Natural Joins Chances are high in retrieving too many rows without meaningful relationships because of duplicate values in Joining key columns*
- *Natural joins cannot handle NULL values in Joining key columns since it matches the column values implicitly by SQL and are not written in the SQL queries*

Queries with three or more tables

SELECT statements can be written by joining *three or more* tables to understand much more relationships between business entities

Queries with three or more tables

- A strategic team tries to understand all customer details, bank accounts, and the relationship between accounts and its transactions

```
SELECT * FROM CUSTOMER NATURAL JOIN ACCOUNT NATURAL JOIN TRANSACTION;
```

- A NATURAL JOIN is used to join all the three tables by matching the common rows
- Here in the example:
  - *Cust_Id* is common key column between CUSTOMER and ACCOUNT
  - *Acct_Num* is common key column between ACCOUNT and TRANSACTION

Queries with three or more tables

- The output of the query gives all matching rows of three tables:

Acct_Num	Cust_Id	Name	Address	State	Phone	Acct_Type	Balance	Acct_status	Relation	Tran_Amount	Channel	Area	Tran_Date
4000-1956-3456	123001	Oliver	225-5, Emeryville	CA	1897614500	SAVINGS	200000	ACTIVE	P	-2000.00	ATM withdrawal	CA	2020-01-13
4000-1956-2001	123002	George	194-6,New brighton	MN	189761700	SAVINGS	400000	ACTIVE	P	-4000.00	POS-Walmart	MN	2020-02-14
4000-1956-2001	123002	George	194-6,New brighton	MN	189761700	SAVINGS	400000	ACTIVE	P	-1600.00	UPI transfer	MN	2020-01-19
4000-1956-2001	123002	George	194-6,New brighton	MN	189761700	SAVINGS	400000	ACTIVE	P	-6000.00	Bankers cheque	CA	2020-03-23
4000-1956-2001	123002	George	194-6,New brighton	MN	189761700	SAVINGS	400000	ACTIVE	P	-3000.00	Net banking	CA	2020-04-24
4000-1956-2001	123002	George	194-6,New brighton	MN	189761700	SAVINGS	400000	ACTIVE	P	-2970.00	Net banking	CA	2020-04-26
4000-1956-5102	123005	Jacob	325-7, Mission Dist	SFO	1897637000	SAVINGS	300000	ACTIVE	P	-6500.00	ATM withdrawal	NY	2020-04-24
4000-1956-5102	123005	Jacob	325-7, Mission Dist	SFO	1897637000	SAVINGS	300000	ACTIVE	P	-3600.00	ATM withdrawal	NY	2020-04-25
4000-1956-2001	123002	George	194-6,New brighton	MN	189761700	SAVINGS	400000	ACTIVE	P	23000.00	cheque deposit	MN	2020-03-15

Other equi-joins

- Like NATURAL JOIN, the EQUI-JOIN / Inner JOIN also matches the rows using common key columns between tables

- The only difference with equi-joins and NATURAL join is, joining key columns are explicitly specified

- Equi - JOIN otherwise called INNER JOIN is simply defined with JOIN / INNER JOIN clause between two tables but essentially defines with an equal (=) operator.

Other equi-joins

- Equi-Joins / INNER JOINS efficiently handles NULL values and comparisons

- Assign default values by replacing NULL values in joining key columns, and ensures the records are not dropped in the output

```
SELECT BA.Acct_Num, BA.Acct_type, BA.Balance, TR.Tran_Amount
FROM ACCOUNT  BA
JOIN TRANSACTION  TR
WHERE BA.Acct_Num  = TR.Acct_Num AND ifNULL(BA.BALANCE, 0) = TR.Tran_Amount
```

Other equi-joins

Output:

Acct_Num	Acct_type	Balance	Tran_Amount
5000-1700-7755	SAVINGS	NULL	0.00
5000-1700-9911	SAVINGS	2000	2000.00

- JOIN clause with an equal (=) operator joins two tables ACCOUNT and TRANSACTION using common key column: Acct_num

- In first account number , Balance column has Null value which replaced with ‘0’ in order to compare with tran_amount

Other equi- Joins

- Equi - Join can also be implemented with USING clause which replaces WHERE and equal operator (=)

```
SELECT cust_id,Name, Acct_Num
FROM   CUSTOMER
JOIN   ACCOUNT
USING (CUST_ID)
```

- The output of this Query gives all matching records of CUSTOMER and ACCOUNT by joining with common key : Cust_Id

Other equi- Joins

Output:

cust_id	Name	Acct_Num
123001	Oliver	4000-1956-3456
123001	Oliver	5000-1700-3456
123002	George	4000-1956-2001
123002	George	5000-1700-5001
123003	Harry	4000-1956-2900
123004	Jack	5000-1700-6091
123004	Jack	4000-1956-3401
123005	Jacob	4000-1956-5102
123006	Noah	4000-1956-5698
123007	Charlie	5000-1700-9800
123007	Charlie	4000-1956-9977
123007	Charlie	9000-1700-7777-4321
123008	Robin	5000-1700-7755

USING clause identifies the common joining key columns in both tables and produces matching rows

Non Equi Joins

Non Equi- Joins

- Non equi-JOINS uses comparison operators like >, < , NOT , <> in order to filter the records in one table and map the remaining rows across the other table rows

- It is widely used for creating different dimensional reports

Non Equi- Joins

- Below is an example of Non equi JOIN with range operator “>” that filters lower interest_rates and provide higher **privileged** interest-rate to all deposit / saving holders

```
SELECT AC.Acct_Num, AC.Acct_type, IR.Rate Modified_rate, IR.Acct_type
Interest_type
FROM   ACCOUNT AC
JOIN  INTEREST IR
WHERE IR.RATE >  0.07 AND    AC.Acct_Type NOT LIKE '%CARDS%'
```

Non Equi- Joins

- Interest rates are lower for Accounts, so bank decided to provide higher interest rate called privileged interest rate greater than 7%

- NOT LIKE is used to filter CREDIT CARD accounts

Acct_Num	Acct_type	Modified_rate	Interest_type
4000-1956-3456	SAVINGS	0.08	PRIVILEGED
5000-1700-3456	FIXED DEPOSITS	0.08	PRIVILEGED
4000-1956-2001	SAVINGS	0.08	PRIVILEGED
5000-1700-5001	FIXED DEPOSITS	0.08	PRIVILEGED
4000-1956-2900	SAVINGS	0.08	PRIVILEGED
5000-1700-6091	FIXED DEPOSITS	0.08	PRIVILEGED
4000-1956-3401	SAVINGS	0.08	PRIVILEGED
4000-1956-5102	SAVINGS	0.08	PRIVILEGED
4000-1956-5698	SAVINGS	0.08	PRIVILEGED
5000-1700-9800	SAVINGS	0.08	PRIVILEGED
4000-1956-9977	FIXED DEPOSITS	0.08	PRIVILEGED
9000-1700-7777-4321	CREDITCARD	0.08	PRIVILEGED
5000-1700-7755	SAVINGS	0.08	PRIVILEGED
5000-1700-9911	SAVINGS	0.08	PRIVILEGED

SQL Consideration for Multiple Queries

Qualified Column Names

- Qualified names of the tables or columns in the Database plays a vital role in data modeling
- Data architects and Business Analysts easily design the models and make a clear lineage of the Business flow process
- Qualified names are the standards established by Data management and Governance, and banking authorities

Qualified Column Names - Example

- CUSTOMER is a table which is a qualified name in the Bank which consists of only customer details and is not conflicting with any other different terms
- Similarly *Cust_Id* , *Name* columns represent unique customer details

```
SELECT Cust_Id, Name, Address FROM CUSTOMER
```

All Column Selection

- Any RDBMS including MySQL allows to create more than 250 columns in a table which depends on size of individual column
- However it depends on Business usage
- For ex: TRANSACTION table can be defined with at least 50 columns because these days we are doing payments in different modes unlike conventional methods to go bank and withdraw money
- In such cases when there are more columns to represent, a wildcard " * " can be used to select all columns in the table

Ex: `SELECT * FROM CUSTOMER`

Self Join

SELF JOIN - Syntax

- SELF JOIN is usually applied when we see meaningful data in a same table
- Self join means joining the same table to itself for multiple times
- Below is the query that is used to produce comparative results for the current month and previous month side by side

```
SELECT T1.Acct_Num as Account,
    T1.Tran_date current_month,
    T1.Tran_Amount Latest_transaction,
    T2.Tran_date previous_month,
    T2.Tran_Amount Previous_transaction
FROM TRANSACTION T1
JOIN TRANSACTION T2
ON  T1.Acct_Num = T2.Acct_Num AND T1.Tran_date > T2.Tran_date
```

SELF JOIN - Syntax

- Both T1 and T2 tables referring to same TRANSACTION table

- With this SELF JOIN , query produced the comparative results side by side for the current and previous month details from one single TRANSACTION table

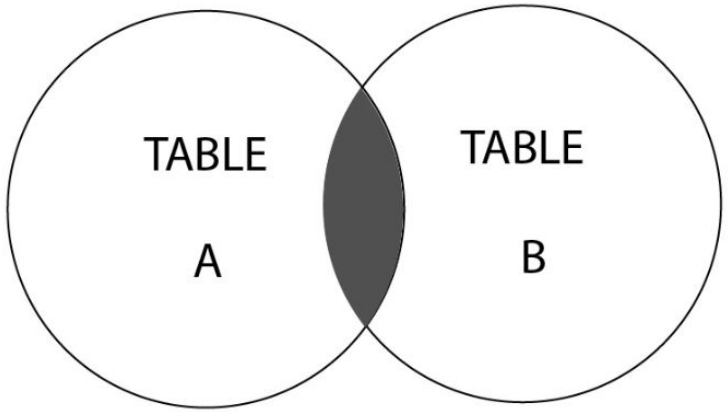
Account	current_month	Latest_transaction	previous_month	Previous_transaction
4000-1956-2001	2020-03-23	-6000.00	2020-02-14	-4000.00
4000-1956-2001	2020-04-24	-3000.00	2020-02-14	-4000.00
4000-1956-2001	2020-04-26	-2970.00	2020-02-14	-4000.00
4000-1956-2001	2020-03-15	23000.00	2020-02-14	-4000.00
4000-1956-2001	2020-02-14	-4000.00	2020-01-19	-1600.00
4000-1956-2001	2020-03-23	-6000.00	2020-01-19	-1600.00
4000-1956-2001	2020-04-24	-3000.00	2020-01-19	-1600.00
4000-1956-2001	2020-04-26	-2970.00	2020-01-19	-1600.00
4000-1956-2001	2020-03-15	23000.00	2020-01-19	-1600.00

INNER JOIN

What is an INNER JOIN?

- The INNER JOIN is the default type of join that is used to select matching rows in both tables using common key joining column

- It can be represented with the following Venn diagram:



- The INNER JOIN represents the highlighted section, which is the intersection between these two tables. *Intersection part is the matching rows.*

INNER JOIN - Syntax

Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

INNER JOIN - Example

- If you want to know the Customers and their account details by joining CUSTOMER and ACCOUNT table using customer_id – common key column

```
SELECT BC.cust_id, BC.name, BA.Acct_Num, BA.Balance
FROM CUSTOMER BC
INNER JOIN  ACCOUNT  BA
ON BC.cust_id = BA.cust_id
```

- INNER JOIN ensures there are matching records in both tables based on CUST_ID, and can retrieve all the fields from Customer and Account tables.

INNER JOIN - Example

Output:

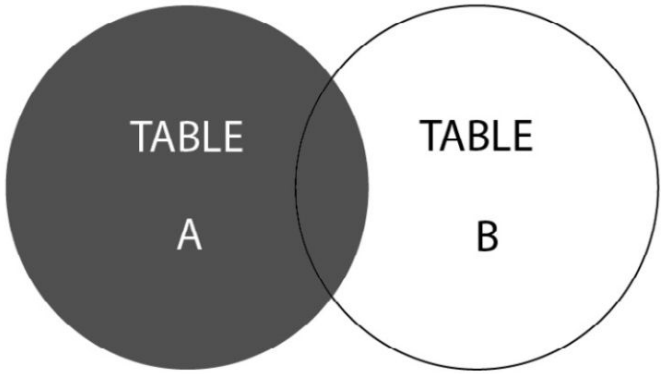
cust_id	name	Acct_Num	Balance
123001	Oliver	4000-1956-3456	200000
123001	Oliver	5000-1700-3456	9400000
123002	George	4000-1956-2001	400000
123002	George	5000-1700-5001	7500000
123003	Harry	4000-1956-2900	750000
123004	Jack	5000-1700-6091	7500000
123004	Jack	4000-1956-3401	655000
123005	Jacob	4000-1956-5102	300000
123006	Noah	4000-1956-5698	455000
123007	Charlie	5000-1700-9800	355000
123007	Charlie	4000-1956-9977	7025000
123007	Charlie	9000-1700-7777-4321	0
123008	Robin	5000-1700-7755	NULL

- The output of this query consists of records if and only if cust_id exists in both the tables

LEFT JOIN

What is a LEFT JOIN?

- The LEFT JOIN is used when you want to select
  - Matching records that are selected from both the tables and
  - All the valid records from *the left table* with Null assignment of columns in the right table



- The LEFT JOIN represents the highlighted section from TABLE A and the intersected section from TABLE B

LEFT JOIN - Syntax

Syntax:

```
SELECT [Column List]
  FROM [Table 1] LEFT OUTER JOIN [Table 2]
    ON [Table 1 Column Name] = [Table 2 Column Name]
WHERE [Condition]
```

- The OUTER word in the query is optional
- This type of join is very similar to the normal JOIN, with the only difference being that it pulls complete details of Left table

LEFT JOIN - Example

- If you want to retrieve all customer accounts in the bank though there are no transactions occurred:

```
SELECT BA.Acct_Num, BA.Balance,BT.Acct_Num as Tran_Account,  BT.Tran_Amount,
       BT.Channel
FROM ACCOUNT BA
LEFT JOIN TRANSACTION BT
ON BA.Acct_Num =BT.Acct_Num order by 1;
```

LEFT JOIN - Example

Output:

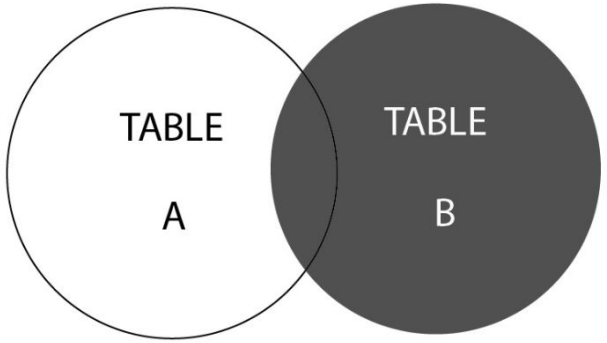
Acct_Num	Balance	Tran_Account	Tran_Amount	Channel
4000-1956-2001	400000	4000-1956-2001	-4000.00	POS-Walmart
4000-1956-2001	400000	4000-1956-2001	-1600.00	UPI transfer
4000-1956-2001	400000	4000-1956-2001	-6000.00	Bankers cheque
4000-1956-2001	400000	4000-1956-2001	-3000.00	Net banking
4000-1956-2001	400000	4000-1956-2001	-2970.00	Net banking
4000-1956-2001	400000	4000-1956-2001	23000.00	cheque deposit
4000-1956-2900	750000	NULL	NULL	NULL
4000-1956-3401	655000	4000-1956-3401	8000.00	Cash Deposit

- LEFT JOIN extracts matching records from both tables based on ACCOUNT_NUMBER
- For unmatched records, it pulls all the account records from left table i.e. ACCOUNT, and shows NULL values for fields in the right table : TRANSACTION as there are no transaction records

RIGHT JOIN

What is a RIGHT JOIN?

- The RIGHT JOIN is used when you want to select
  - Matching records from both the tables and
  - All the valid records from *the right table* with Null assignment of columns in the left table



- The RIGHT JOIN represents the highlighted section, that is, TABLE B, and the intersected section of TABLE A.

RIGHT JOIN - Syntax

Syntax:

```
SELECT [Column List]
  FROM [Table 1] RIGHT OUTER JOIN [Table 2]
    ON [Table 1 Column Name] = [Table 2 Column Name]
WHERE [Condition]
```

The OUTER word in the query is optional

RIGHT JOIN - Syntax - ERROR

- Use the following query retrieve any money transactions that do not exist in ACCOUNT table but exist in TRANSACTION table

- Usually, we will see such transactions when customer do not pay credit card bills but the credit card Account is expired or inactive

```
SELECT BA.Acct_Num,
    BA.Balance,
    BT.Acct_Num as Tran_Account,
    BT.Tran_Amount,
    BT.Channel
FROM ACCOUNT BA
RIGHT JOIN TRANSACTION BT
ON BA.Acct_Num = BT.Acct_Num
```

RIGHT JOIN - Example

Output:

Acct_Num	Balance	Tran_Account	Tran_Amount	Channel
4000-1956-3456	200000	4000-1956-3456	-2000.00	ATM withdrawal
4000-1956-2001	400000	4000-1956-2001	-4000.00	POS-Walmart
4000-1956-2001	400000	4000-1956-2001	-1600.00	UPI transfer
4000-1956-2001	400000	4000-1956-2001	-6000.00	Bankers cheque
4000-1956-2001	400000	4000-1956-2001	-3000.00	Net banking
4000-1956-2001	400000	4000-1956-2001	-2970.00	Net banking
4000-1956-2001	400000	4000-1956-2001	23000.00	cheque deposit
5000-1700-6091	7500000	5000-1700-6091	40000.00	ECS transfer
4000-1956-3401	655000	4000-1956-3401	8000.00	Cash Deposit
4000-1956-5102	300000	4000-1956-5102	-6500.00	ATM withdrawal
4000-1956-5102	300000	4000-1956-5102	-3600.00	ATM withdrawal
4000-1956-5102	300000	4000-1956-5102	-6500.00	ATM withdrawal
4000-1956-5698	455000	4000-1956-5698	-9000.00	Cash Deposit
4000-1956-9977	7025000	4000-1956-9977	50000.00	ECS transfer
9000-1700-7777-4321	0	9000-1700-777...	-5000.00	POS-Walmart
9000-1700-7777-4321	0	9000-1700-777...	-8000.00	Shopping Cart
5000-1700-7755	NULL	5000-1700-7755	0.00	NIL
5000-1700-9911	2000	5000-1700-9911	2000.00	Cash Deposit
NULL	NULL	4000-1700-444...	-5500.00	Shopping Cart
NULL	NULL	4000-1700-444...	-2500.00	Shopping Cart

- Using RIGHT Join, though the accounts that are not present in ACCOUNT table, their transactions are still exist in Transaction table
- In order to catch such pending transactions, especially CREDIT cards, we can apply RIGHT JOIN

FULL OUTER JOIN

FULL OUTER JOIN - Syntax - ERROR

- Full outer Joins helps to retrieve combination of LEFT and RIGHT join results
  - Retrieve result set of ALL *active* accounts in Bank from LEFT JOIN
  - Retrieve result set of ALL *transactions* in Bank from RIGHT JOIN

```
SELECT  BA.Acct_Num, BT.Acct_Num as Tran_Account,
BT.Tran_Amount
FROM ACCOUNT BA
LEFT JOIN TRANSACTION BT
ON  BA.Acct_Num =BT.Acct_Num
UNION
SELECT BA.Acct_Num, BT.Acct_Num as Tran_Account,
BT.Tran_Amount
FROM ACCOUNT BA
RIGHT JOIN TRANSACTION BT
ON  BA.Acct_Num = BT.Acct_Num
```

FULL JOIN - Example

Output:

Acct_Num	Tran_Account	Tran_Amount
4000-1956-3456	4000-1956-3456	-2000.00
4000-1956-2001	4000-1956-2001	-4000.00
4000-1956-2001	4000-1956-2001	-1600.00
4000-1956-2001	4000-1956-2001	-6000.00
4000-1956-2001	4000-1956-2001	-3000.00
4000-1956-2001	4000-1956-2001	-2970.00
4000-1956-5102	4000-1956-5102	-6500.00
4000-1956-5102	4000-1956-5102	-3600.00
4000-1956-2001	4000-1956-2001	23000.00
5000-1700-6091	5000-1700-6091	40000.00
4000-1956-3401	4000-1956-3401	8000.00
4000-1956-5698	4000-1956-5698	-9000.00
4000-1956-9977	4000-1956-9977	50000.00
9000-1700-777...	9000-1700-777...	-5000.00
9000-1700-777...	9000-1700-777...	-8000.00
5000-1700-7755	5000-1700-7755	0.00
5000-1700-9911	5000-1700-9911	2000.00
5000-1700-3456	NULL	NULL
5000-1700-5001	NULL	NULL
4000-1956-2900	NULL	NULL

- Here, we see the records with
  - NULL transactions in TRANSACTION table
  - Missing accounts in ACCOUNT table
- The purpose of FULL Join is to get a glance on all accounts and transactions , and take measures on

CROSS JOIN

CROSS JOIN - Syntax

- CROSS JOIN combines all rows from left and right tables

```
SELECT BC.Cust_id,
       BC.Name,
       MG.NOTICE
FROM CUSTOMER BC
CROSS JOIN MESSAGE MG
```

- The result of the query gives an output that each customer is sent with all type of notifications present in the MESSAGE table

CROSS JOIN - Syntax

Output:

Cust_id	Name	NOTICE
123007	Charlie	banks are closed today
123007	Charlie	banks are closed today
123007	Charlie	All banks are closed due to Lockdown
123007	Charlie	Limit of only 4000/- withdrawal per card is allowed
123007	Charlie	banks are closed today
123008	Robin	banks are closed today
123008	Robin	banks are closed today
123008	Robin	All banks are closed due to Lockdown
123008	Robin	Limit of only 4000/- withdrawal per card is allowed
123008	Robin	banks are closed today

- CROSS JOIN combines all rows from left and right tables
- The result of the query gives an output that each customer is sent with all type of notifications present in the Bank_customer_Messages table.

Join multiple tables

JOIN MULTIPLE Tables - Example

- Multi table JOINS are allowed when a complex report needs complete details from all the tables including customer details, their active accounts , transactions with business conditions

```
SELECT BC.cust_id,
       BC.Name,
       BC.Address,
        BA.Acct_Num ,
        BA.Balance ,
        BT.Tran_Amount ,
        BT.Channel
FROM CUSTOMER BC
INNER JOIN  ACCOUNT BA
ON BA.cust_id=BC.cust_id
LEFT JOIN TRANSACTION BT
ON BA.Acct_Num =BT.Acct_Num
```

JOIN MULTIPLE Tables - Example

Output:

Cust_id	Name	Address	Acct_Num	Balance	Tran_Amount	Channel
123001	Oliver	225-5, Emeryville	4000-1956-3456	200000	-2000.00	ATM withdrawal
123002	George	194-6,New brighton	4000-1956-2001	400000	-4000.00	POS-Walmart
123002	George	194-6,New brighton	4000-1956-2001	400000	-1600.00	UPI transfer
123002	George	194-6,New brighton	4000-1956-2001	400000	-6000.00	Bankers cheque
123002	George	194-6,New brighton	4000-1956-2001	400000	-3000.00	Net banking
123002	George	194-6,New brighton	4000-1956-2001	400000	-2970.00	Net banking
123005	Jacob	325-7, Mission Dist	4000-1956-5102	300000	-6500.00	ATM withdrawal
123005	Jacob	325-7, Mission Dist	4000-1956-5102	300000	-3600.00	ATM withdrawal
123002	George	194-6,New brighton	4000-1956-2001	400000	23000.00	cheque deposit
123004	Jack	229-5, Concord	5000-1700-6091	7500000	40000.00	ECS transfer
123004	Jack	229-5, Concord	4000-1956-3401	655000	8000.00	Cash Deposit

- Multiple tables are joined with multiple type of JOINS like INNER / LEFT outer

did you know?

In practise, Multi table JOINS are allowed when a complex report needs complete details from all the tables including customer details, their active accounts, transactions with business conditions.

There is no maximum limit on joining the tables. An expert can write queries on tables by joining 20- 30 tables easily

Rules for Multi Join Query

- Number of Tables joining in a multi table queries should have at least (No.of tables – 1) joining conditions.

  Ex: 4 tables when joined needs at least 3 or more joining conditions.

- It is preferred to have unique column values in the driving tables especially when using INNER JOIN otherwise it leads to cross distribution of rows among the tables
  - CUST_ID in the CUSTOMER table is always **unique** , however the *cust_id* is repeated in ACCOUNT table since a customer has multiple accounts
  - Similarly Acct_Num is unique in ACCOUNT table but it is repeated in TRANSACTION table since there are many transactions for each account

Join Notations using (+) operator (Non ANSI) Supported in Oracle- RDBMS

JOIN Notations using (+) Operator - Syntax (Non - ANSI)

- (+) Operator can perform LEFT and RIGHT joins by simply changing the positions in the WHERE clause

Syntax: perform LEFT JOIN using (+)

```
SELECT BA.Acct_Num ,
       BA.Balance_Amount ,
       AT.Tran_Amount ,
       AT.Channel
FROM ACCOUNT BA
     TRANSACTION AT
WHERE BA.Acct_Num = AT.Account_Number(+)
```

JOIN Notations using (+) Operator - Syntax (Non - ANSI)

Syntax: perform RIGHT JOIN using (+)

```
SELECT BA.Acct_Num ,
       BA.Balance_Amount,
       AT.Tran_Amount ,
       AT.Channel
FROM ACCOUNT BA
     TRANSACTION AT
WHERE
    (+)BA.Acct_Num
    =AT.Account_Number
AND       BA.ACCOUNT_STATUS = 'ACTIVE'
```

- (+) Operator is simply a replace RIGHT JOIN clause
- By any means (+) operator is not added, then the Query will be converted to INNER JOIN. Developers should be familiar to write queries using (+) operator

Thank You