Multiple Table Queries

Agenda

- Introduction to Joins with ER Diagram
- Types of Joins
    - Inner Join
    - Outer Join
    - Left & Right Join
    - Full Outer Join
    - Self 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.

Introduction

- Consider the following two tables Customer and Account:

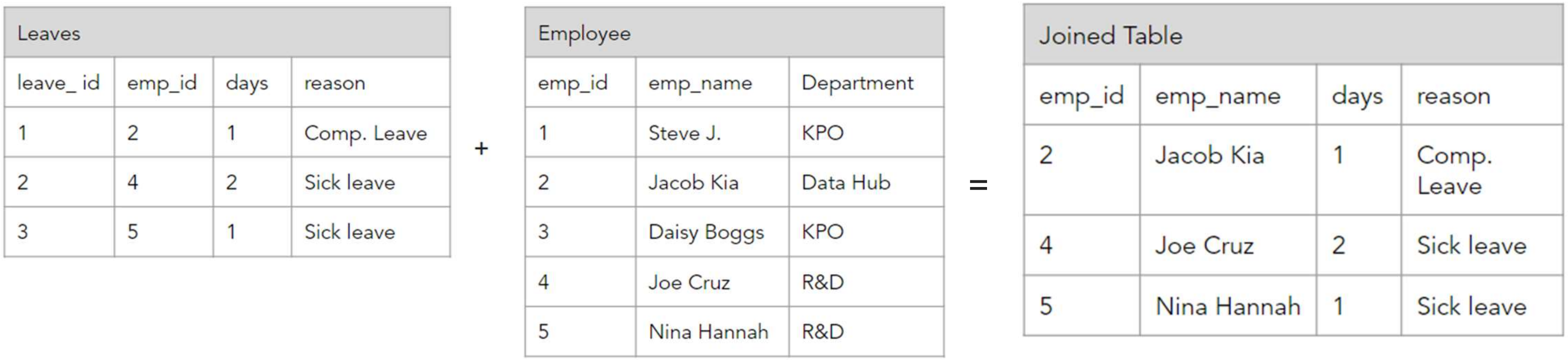
Customer
cust_id
name
address
contact_num

Account
cust_id
savings
deposits
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.

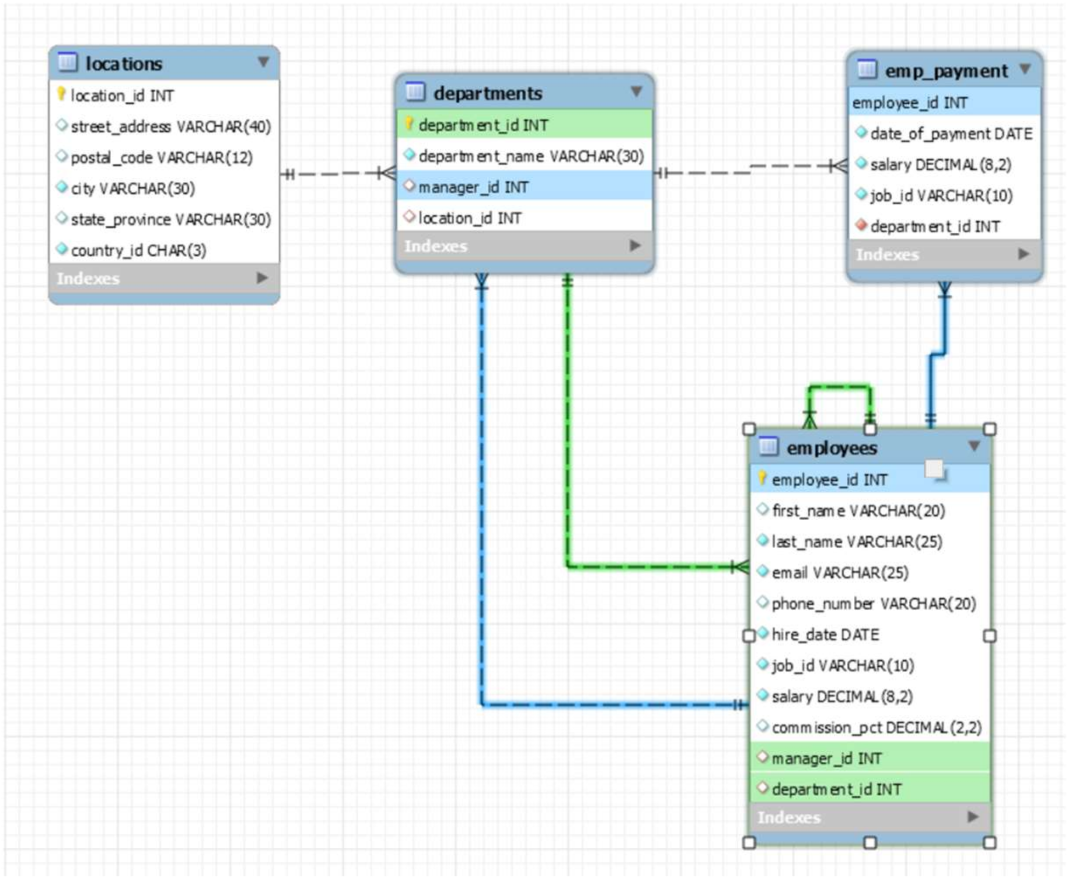
- Few relationships between customers , accounts and transactions table are below:
    - How many accounts are maintained by each customer?
    - Different modes of customer payments?
    - Customer using Smart cash (digital) payment via various channels including UPI, IVR, point_of_sale?

Introduction

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

ER Diagram for the study

- Please consider the following ER diagram for our study:



Two Table Query example

Two Table Query

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

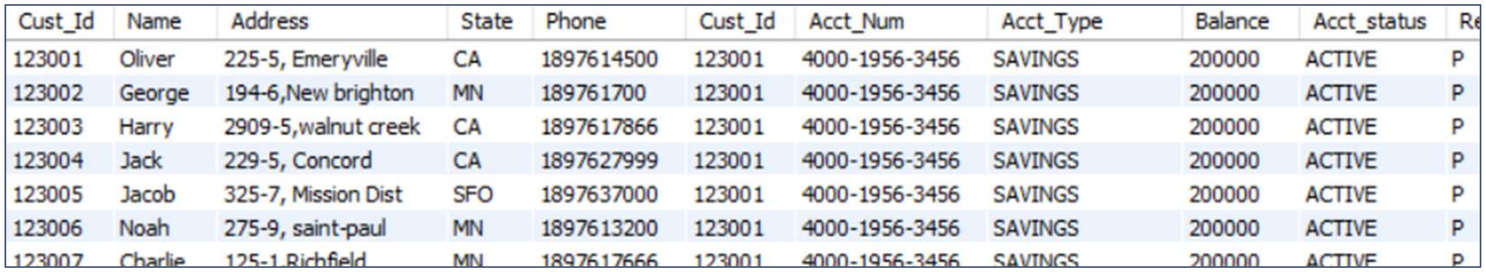
- The next example shows how two tables are accessed via SELECT query and will drill down how it works.

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

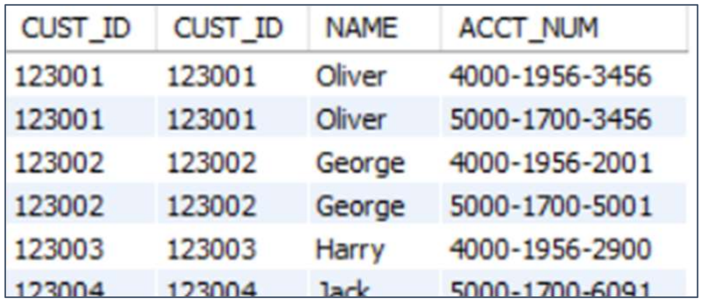
- Table & Column aliases are useful when a SELECT query is using ***similarly named*** *columns* from two different tables.
- Aliases are given for ***tables*** and as well as for ***columns.***
- It represents similar column names with different meaning in SELECT clauses.
- Aliases are also given for 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.

```
SELECT CUSTOMER.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”.

NOTE

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

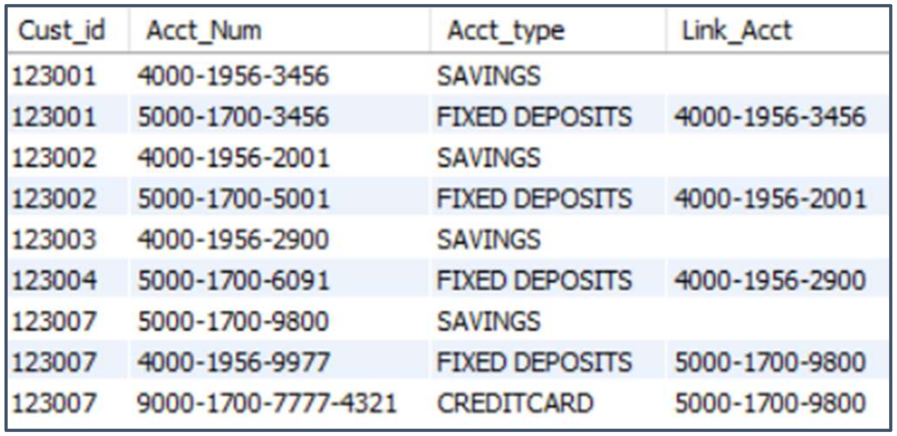
Parent - child relationship Queries

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.

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.

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 table1 JOIN  table2 JOIN  table3
On <join conditions>;
```

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.
- In such cases when there are more columns to represent, a wildcard " * " can be used to select all columns in the table.

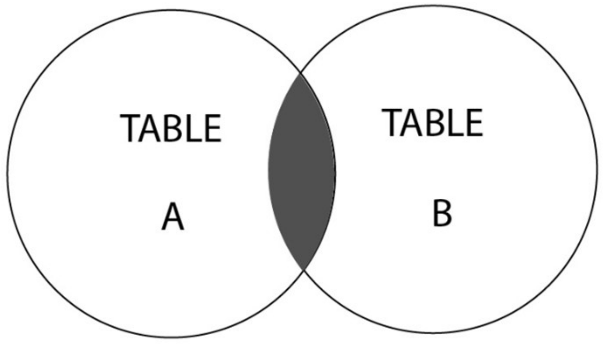
Ex:
```
SELECT * FROM CUSTOMER
```

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

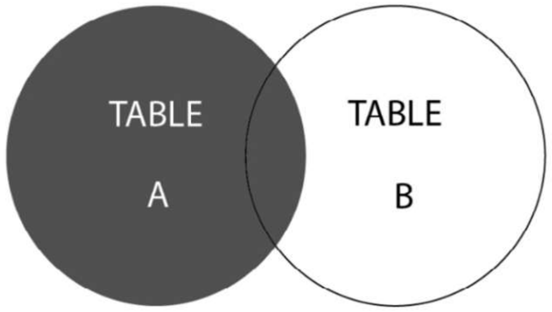
Syntax:

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

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:

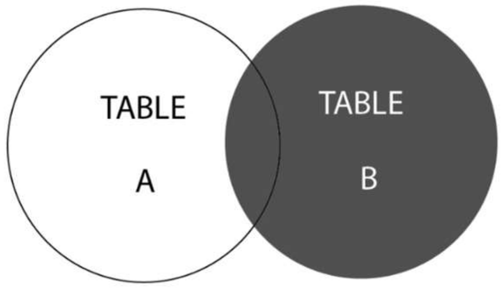
```
SELECT [Column List]
  FROM [Table 1] LEFT OUTER JOIN [Table 2]
    ON [Table 1 Column Name] = [Table 2 Column Name]
WHERE [Condition]
```

- 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 .

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.

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 *rows* from LEFT JOIN
  - Retrieve result set of ALL *rows* from RIGHT JOIN

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.

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

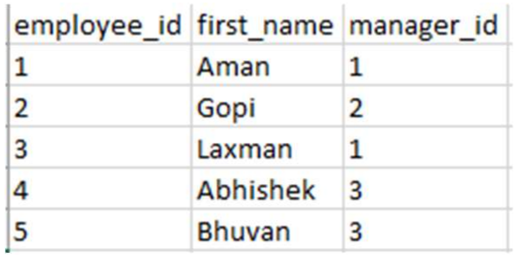
- Number of Tables joining in a multi table queries should have at least (No.of tables – 1) joining conditions.
  Ex: 4 tables when joined needs at least 3 or more joining conditions.

- 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.

Self Join

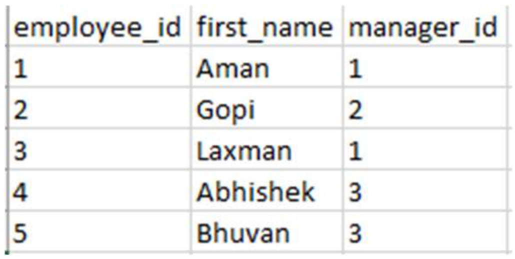
SELF JOIN - Syntax

- **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.

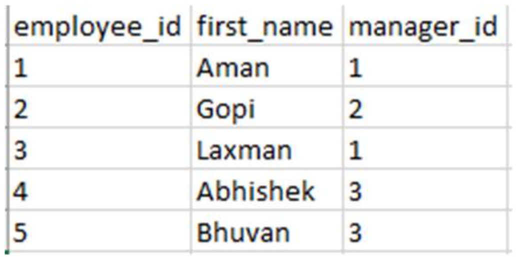


employee_id	first_name	manager_id
1	Aman	1
2	Gopi	2
3	Laxman	1
4	Abhishek	3
5	Bhuvan	3

SELF JOIN - Syntax



employee_id	first_name	manager_id
1	Aman	1
2	Gopi	2
3	Laxman	1
4	Abhishek	3
5	Bhuvan	3



employee_id	first_name	manager_id
1	Aman	1
2	Gopi	2
3	Laxman	1
4	Abhishek	3
5	Bhuvan	3

Summary

This deck explains the multiple joins like, Right, Left, Full outer etc. we can use in the queries. Their syntax and rules are briefly explained in the above slides.

Each type of join is explained with multiple examples.

Subqueries & Query Expressions

Agenda

- Introduction to Subquery
- Properties and benefits
- Subquery using
    - where clause
    - From Clause
- Nested Subqueries

Subquery

- When a select query is nested inside another main query, it is called a subquery.
- Subqueries require common key columns for joining with main queries just like joins.
- Subqueries are also known as virtual tables with independent business logic.
- Subqueries operate independently and share results with the main query, so the complexity of writing queries decreases.

Subquery

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

Type	Properties
Single Row Subquery	Feeds a single value to the main query.
Multiple Row Subquery	Returns more number of rows.
Multiple Column Subquery	Gives one or more columns results which matches with outer query columns.
Correlated Subquery	the subquery is dependent on outer query for retrieving each record.
Nested Subquery	Subqueries are placed within another subquery. It is called as nesting.

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.

Subqueries in Where Clause

Subquery in Where Clause

- Using multi-row operators such as EXISTS, IN, ANY, and ALL, subqueries can be written in the WHERE clause of another query.
- They can also use single-row comparison operators such as <, >, =

Subqueries in Where Clause - Syntax using IN clause

The IN operator is useful when the main query searches all of the rows returned by a subquery.

Subquery Search Condition

Subquery Comparison Test

- Sub Queries' results are dynamic, not static.
- Developers will not need to manually enter records because dynamic input will be retrieved from a subquery.
- In WHERE clause, these subqueries use dynamic input in a condition to fetch records.

Subquery Existence Test

- Each record of the main query examines the subquery using the common key columns.
- In both the main query and the subquery, the common key column must be the same.
- When the subquery's conditions are satisfied with the input column values in the main query, it returns true (1) or false (0).

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.

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 Quantified Test using ANY

- In the next example:
  - The main Query tries to retrieve all the rows if at least one condition in subquery is satisfied.
  - “>” or “<” range operator along with “ANY” walks through of all of the rows.

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.

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…

CRITICAL NOTE

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…

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.
- In complex queries, where there is a need to call subqueries repeatedly, the WITH clause plays a major role.

Subqueries and Joins

- Subquery and JOIN perform the execution of queries similarly and retrieves the same output but varies with below features.

Subqueries and Joins

- Unlike subqueries, which separate complex logic from the main query, joins include the whole logic.
- We will see examples on how subqueries can be used along with JOINS.
- Determinations can be performed in the subquery and returned as a single value, while JOIN queries perform those calculations in the main SELECT query.
- Filtering records in a table with subquery is more efficient than join queries.

Summary

This slide explains, often referred to as Inner Queries, Sub Queries, or Nested Queries, nested queries are a query within another SQL query embedded in the WHERE clause. As a result of a subquery, data will be returned that will be used in the main query as a condition for further restricting data to be retrieved.

Normalization

Anomaly

- Anomaly is mismatch / inconsistencies in the data

- Types of Anomaly

    Insertion Anomaly

    Updation Anomaly

    Deletion Anomaly

Functional Dependency

All the non-key Columns of a table are said to be dependent on Primary key, which uniquely identifies the rows in a table.

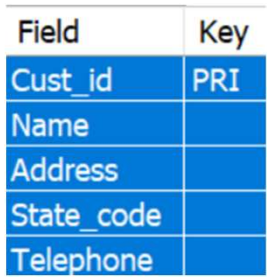
For example:

Consider a CUSTOMER table with attributes:

Cust_Id, Name, Address, State, Telephone.

Here **Cust_Id** is a primary Key, which uniquely identifies the remaining columns.

Hence, the non-key columns are said to be functionally dependent on Primary Key.



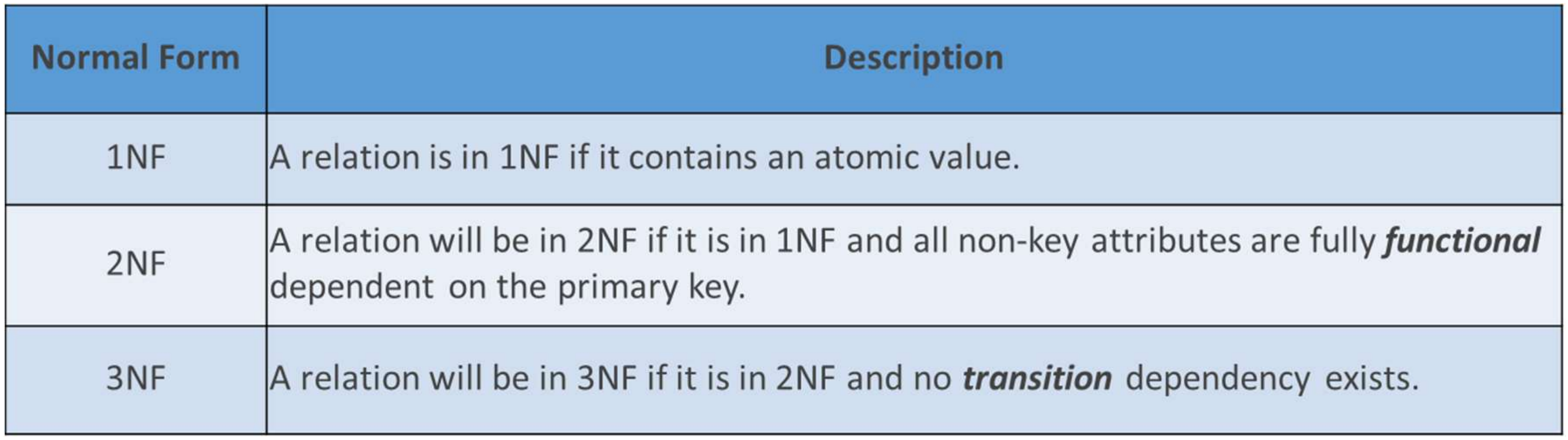
Field	Key
Cust_id	PRI
Name	
Address	
State_code	
Telephone	

Normalization

- Normalization is the process of organizing the data attributes with their relationships.
- Normalization minimizes the redundancy of data rows.
- Normalization minimizes the dependency of columns.
- Anomalies (flaws) such as inserting, updating and deleting are eliminated.
- Normalization divides the larger table into smaller ones and establishes entity relationships among the smaller ones.

Normalization

- Normalization is evolved into several stages over a period of time.



Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully ***functional*** dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no ***transition*** dependency exists.

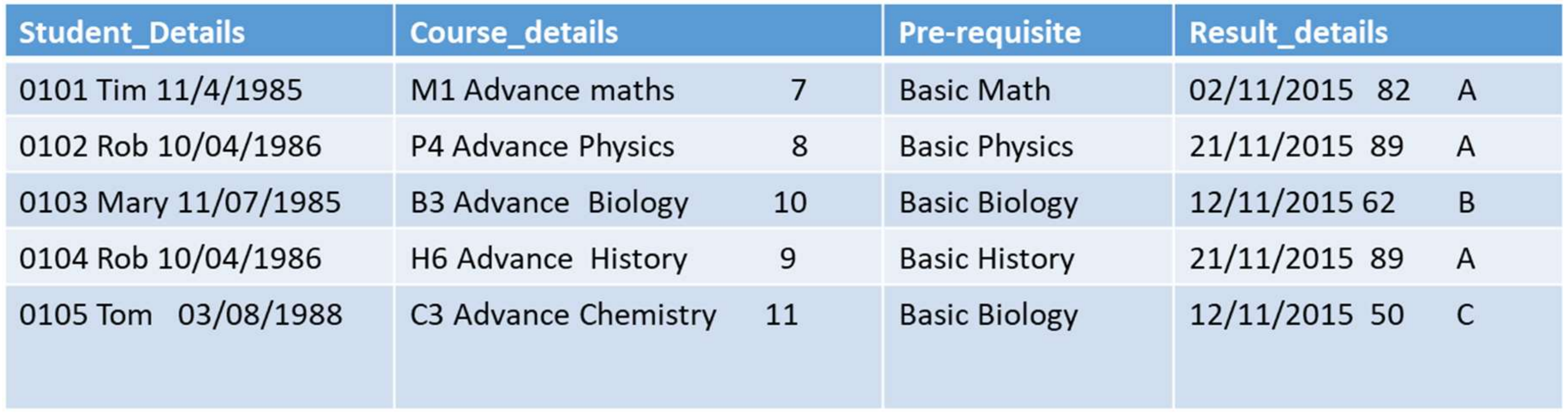
First Normal Form (1NF)

- A table is said to be in *First Normal Form*, when it follows the following 4 rules:

- A column/attribute in a table should consists of a scalar atomic value.
- All the values stored in a column should have same business term.
- All the columns in a table should be represented with unique names.
- The order of rows and columns doesn’t matter.

First Normal Form (1NF) - Example

Raw Data:



Student_Details	Course_details	Pre-requisite	Result_details
0101 Tim 11/4/1985	M1 Advance maths 7	Basic Math	02/11/2015 82 A
0102 Rob 10/04/1986	P4 Advance Physics 8	Basic Physics	21/11/2015 89 A
0103 Mary 11/07/1985	B3 Advance Biology 10	Basic Biology	12/11/2015 62 B
0104 Rob 10/04/1986	H6 Advance History 9	Basic History	21/11/2015 89 A
0105 Tom 03/08/1988	C3 Advance Chemistry 11	Basic Biology	12/11/2015 50 C

Normalization before and after 1 - NF

Student#	Student_Name	DOB	Course#	CourseName	Pre Requisite	Duration in days	Date of Exam	Marks	Grade
0101	Tim	11/4/1985	M1	Advance Math	Basic Math	7	02/11/2015	82	A
0102	Rob	10/04/1986	P4	Advance Physics	Basic Physics	8	21/11/2015	89	A
0103	Mary	11/07/1985	B3	Advance Biology	Basic Biology	10	12/11/2015	62	B

Second Normal Form (2NF)

- Each and every non-key & independent attribute has a functional dependency on the primary key.
- 2NF is evolutional after 1NF , and handled the redundancy of data effectively.
- Below rules are followed to achieve 2NF.
- In 2NF, the data must be in 1NF.

Tables after 2-NF

- Student# ,Course# → Marks
- Student#, Course# → Grade
- Marks → Grade
- Student# → StudentName, DOB
- Course# → CourseName, Pre-Requisite,
- DurationDays, Date of exam

Second Normal Form – (2NF)

- Student# ,Course# → Marks
- Student#, Course# → Grade
- Marks → Grade
- Student# → StudentName, DOB
- Course# → CourseName, Pre-Requisite,
- DurationDays, Date of exam

Partial Dependency with the Key attribute → Split/Decompose the tables to remove partial dependencies

Second Normal Form – (2NF)

Student Table

Student#	Student_Name	Date Of Birth
0101	Tim	11/4/1985
0102	Rob	10/04/1986
0103	Mary	11/07/1985

Result Table

Student#	Course#	Marks	Grade
0101	M1	82	A
0102	P4	89	A
0103	B3	62	B

Course Table

Course#	CourseName	Prerequisite	Durationindays	Date Of Exam
M1	Advance Math	Basic Math	7	02/11/2015
P4	Advance Physics	Basic Physics	8	21/11/2015
B3	Advance Biology	Basic Biology	10	12/11/2015

Third Normal Form (3-NF):

To achieve 3-NF,

- The attributes of a table should already be in 2-NF, so full functional dependency was achieved already in 2- NF.
- If any non-prime have transitive dependency, this is resolved with 3 - NF.
- Transitive dependency is like attribute - A depends on attribute - B, then attribute - B depends on attribute - C and so on

Third Normalization – (3NF)

Result_table

Student#	Course#	Marks	Grade
0101	M1	82	A
0102	P4	89	A
0103	B3	62	B

Student# ,Course# → Marks

Student#, Course# → Grade

Marks → Grade

Student#,Course#→ Marks→ Grade:TD

→ Remove

Summary

This slide discuss the anomalies and its types in the SQL.

When a correlation is generated directly from a user view, anomalies arise. Anomalies may appear as updates, deletions, or insertions. It is a data inconsistency that arises from partial updating and redundancy.

Thank You