Structured Query Language contd..

To Display Empno, Ename and Dname

Emp

Empno	Ename	Deptno
7369	SMITH	10
7499	ALLEN	30
7521	WARD	30
7566	JONES	20
7654	MARTIN	30
7698	BLAKE	30

Dept

Deptno	Dname	Location
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	HR	DALLAS
60	SYSTEMS	NEW YORK

```
SELECT   empno, ename,  Emp . deptno,  dname

         FROM  Emp , Dept   WHERE   Emp.deptno = Dept.deptno
```

Obtaining Data from Multiple Tables

- Use a **JOIN** to query data from more than one table.
- Write the join condition in the WHERE clause.

```
SELECT    < Multi-Table Column List >
FROM      Table_Name1, Table_Name2
WHERE     <Join Condition>
```

- When the ***same column name*** appears in more than one table
  - **Prefix the column name with the table name**

Obtaining Data from Multiple Tables

Traditional Syntax

```
SELECT   < Multi-Table Column List >
FROM     Table_Name1, Table_Name2
WHERE    <Join Condition>
```

Alternate Syntax

```
SELECT   < Multi-Table Column List >
FROM     Table_Name1  INNER JOIN  Table_Name2
ON       <Join Condition>
```

Obtaining Data from Multiple Tables

```
SELECT empno, ename, Emp.deptno, dname
    FROM Emp, Dept
        WHERE Emp.deptno  Dept.deptno ;
```

```
SELECT empno, ename, emp.deptno, dname
    FROM Emp  INNER JOIN  Dept
        ON  Emp.deptno = Dept.deptno ;
```

Obtaining Data from Multiple Tables

EMPLOYEES

Empno	Ename	Deptno
7369	SMITH	20
7499	ALLEN	30
7521	WARD	30
7566	JONES	20
7654	MARTIN	30
7698	BLAKE	30

DEPARTMENTS

Deptno	Dname	Location
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	HR	DALLAS
60	SYSTEMS	NEW YORK

Foreign key

Relative key

Reference key

Obtaining Data from Multiple Tables

- Join Variants
  - **Equi-Join**
    - When Join condition is formed with equality comparision
  - **Non-equijoin**
    - When Join condition is not formed with equality comparision
  - **Self join**
    - Joining table to itself
  - **Outer join**
    - To overcome cross table anomalies

Joins Using Table Aliases

```
SELECT  e.employee_id, e.last_name, e.department_id,
        d.department_id, d.location_id
FROM    employees e  departments d
```

Joining More than Two Tables

EMP

LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90
Hunold	60
Ernst	60
Lorentz	60
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Zlotkey	80
Abel	80
Taylor	80

DEPT

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

LOCATIONS

LOCATION_ID	CITY
1400	Southlake
1500	South San Francisco
1700	Seattle
1800	Toronto
2500	Oxford

To join *n* tables together, you need a minimum of n-1 join conditions. For example, to join three tables, a minimum of two joins is required.

```
SELECT  last_name, dept_id,
        city
FROM    Emp e,  Dept d, Locations l
WHERE   e.dept_id = d.dept_id
AND     d.loc_id = l.loc_id
```

Non-Equijoins

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

Display the last name, salary and Corresponding Grade

Non-Equijoins

```
SELECT   e.last_name,  e.salary,  j.grade

FROM   EMPLOYEES  e,   JOB_GRADES  j

WHERE   e.salary  BETWEEN   j.lowest_sal   AND   j.highest_sal
```

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

To find out – WHO Works For WHOM

EMPNO	ENAME	MGR	SAL	DEPTNO
7369	SMITH	7902	800	20
7499	ALLEN	7698	1600	30
7521	WARD	7698	1250	30
7566	JONES	7839	2975	20
7654	MARTIN	7698	1250	30
7698	BLAKE	7839	2850	30
7782	CLARK	7839	2450	10
7788	SCOTT	7566	3000	20

- Here fact is that
  - **WHO** and **WHOM** both are present within the **same table**.
  - So the **comparative factors in JOIN condition** are
    - **Emp.mgr and Emp.empno**
  - To avoid the ambiguity, we need to create alias twice for table Emp

Self Join

- Ultimately,
  - The comparisons are taking place within the same table
  - i.e. **every row of Emp will be compared with every row of EMP itself**
  - This can be achieved by creating the **two aliases of same table.**

This is nothing but **SELF JOIN**

To find out – WHO Works For WHOM

EMPNO	ENAME	MGR	SAL	DEPTNO
7369	SMITH	7902	800	20
7499	ALLEN	7698	1600	30
7521	WARD	7698	1250	30
7566	JONES	7839	2975	20
7654	MARTIN	7698	1250	30
7698	BLAKE	7839	2850	30
7782	CLARK	7839	2450	10
7788	SCOTT	7566	3000	20

```
SELECT   E1.ename  || ‘  Wrks for  ‘   ||  E2.ename

FROM     Emp  E1,   Emp  E2

WHERE    E1.mgr  =  E2.empno
```

Cartesian Products

- **Cartesian Products**
  - A Cartesian product is formed when:
    - A join condition is omitted
    - A join condition is invalid
    - All rows in the first table are to be joined to all rows in the second table
  - To avoid a Cartesian product, always include a valid join condition in a WHERE clause.

DBMS_Assign2_PSNO1_PSNO2

1. Write a query to match the salespeople to the customers according to the city they are living.
2. Write a query to select the names of customers and the salespersons who are providing service to them.
3. Write a query to find out all orders by customers not located in the same cities as that of their salespeople
4. Write a query that lists each order number followed by name of customer who made that order

DBMS_Assign2_PSNO1_PSNO2

5. Write a query that finds all pairs of customers having the same rating……
6. Write a query to find out all pairs of customers served by a single salesperson……..
7. Write a query that produces all pairs of salespeople who are living in same city……..

Exercise

Write a query that finds all pairs of customers having the same rating……

CNAME	CNAME	RATING
Clemens	Hoffman	100
Hoffman	Pereira	100
Clemens	Pereira	100
Giovanni	Liu	200
Cisneros	Grass	300

Exercise

Write a query to find out all pairs of customers served by a single salesperson……..

SNAME	SNUM	CNAME	CNAME
Peel	1001	Hoffman	Clemens
Serres	1002	Liu	Grass

Exercise

Write a query that produces all pairs of salespeople who are living in same city……..

SNAME	SNAME
Motika	Peel

Exercise

5. Write a query to match the salespeople to the customers according to the city they are living.

```
SELECT Customers.cname, Salespeople.sname, Salespeople.city
    FROM Salespeople, Customers
        WHERE Salespeople.city = Customers.city
```

Exercise

6. Write a query to select the names of customers and the salespersons who are providing service to them.

```
SELECT Customers.cname, Salespeople.sname
    FROM Salespeople, Customers
        WHERE Salespeople.snum = Customers.snum
```

Exercise

7. Write a query to find out all orders by customers not located in the same cities as that of their salespeople

```
SELECT Onum, Cname, Orders.Cnum, Orders.Snum
    From Salespeople, Customers, Orders
        WHERE Customers.City <> Salespeople.City AND
            Orders.Cnum = Customers.Cnum AND
            Orders.Snum = Salespeople.Snum
```

Exercise

8. Write a query that lists each order number followed by name of customer who made that order

```
SELECT onum,cname
    FROM Orders, Customers
        WHERE Customers.cnum = Orders.cnum
```

Exercise

Write a query to find out all pairs of customers served by a single salesperson……..

SNAME	SNUM	CNAME	CNAME
Peel	1001	Hoffman	Clemens
Serres	1002	Liu	Grass

```
SELECT Salespeople.snum,sname,c1.cname,c2.cname
 FROM Salespeople,Customers C1,Customers C2
WHERE c1.snum = c2.snum AND c1.snum = salespeople.snum
    AND c1.cnum<c2.cnum
```

- Consider the following two tables

Parts

Part_ID	Supp_ID
P1	S1
P2	S2
P3	
P4	

Supplier

Supp_ID	Supp_Name
S1	CUMMINS
S2	THERMAX
S3	TOSHIBA

- Notice above that there are two parts (P3 and P4) that don't have a supplier recorded yet.
- As well, there is a supplier (S3) who doesn't yet supply any part.

Cross Table Anomalies

- We want to generate a report of all the parts and their corresponding suppliers.

```
SELECT p.part_id, s.supp_name
    FROM Parts p, Supplier s
        WHERE p.supp_id = s.supp_id;
```

Part_id	Supp_name
p1	CUMMINS
p2	THERMAX

The join resulted in just the rows that have corresponding rows in both tables.

Therefore, the parts that don't have a supplier, or the suppliers that don't supply any part are excluded from the result set.

Outer Join

- If we want all parts to be listed in the result set, irrespective of whether they are supplied by any supplier or not…
  - **We need to perform an Outer Join**

```
SELECT p.part_id, s.supp_name
    FROM Parts p, Supplier s
        WHERE p.supp_id = s.supp_id (+) ;
```

Part_id	Supp_name
p1	CUMMINS
p2	THERMAX
P3	
p4	

The outer join above lists all of the parts.

For the parts that don't have a corresponding supplier, null values are displayed for the SUPPLIER_NAME column.

Outer Join

- However, not all the suppliers are displayed. Since supplier S3 doesn't supply any parts, it gets excluded from the result set of the above outer join. If we want all the suppliers listed in the result set, irrespective of whether they supply any part or not, we need to perform an outer join like the following:

```
SELECT p.part_id, s.supp_name
    FROM Parts p, Supplier s
        WHERE p.supp_id(+) = s.supp_id ;
```

Part_id	Supp_name
p1	CUMMINS
p2	THERMAX
	TOSHIBA

The outer join above lists all of the suppliers.

For the suppliers that don't supply any part, null values are displayed for the PART_ID column.

Outer Join

- However, not all the parts are displayed. Since parts P3 and P4 are not supplied by any suppliers, they get excluded from the result set of the above outer join.
- If we want
  - all the parts (irrespective of whether they are supplied by any supplier or not)
  - all the suppliers (irrespective of whether they supply any part or not) listed in the same result set
- We have a problem.

Outer Join

- That's because the traditional outer join (using the '+' operator) is unidirectional, and you can't put (+) on both sides in the join condition.
- The following will result in an error
- SELECT p.part_id, s.supp_name

  FROM Parts p, Supplier s

  WHERE p.supp_id(+) = s.supp_id (+) ;

ERROR at line 3: ORA-01468: a predicate may reference only one outer-joined table

Full Outer JOIN – New Syntax

- Remember, if we want to retain all the parts in the result set, irrespective of whether any supplier supplies them or not, then we need to perform full outer join.
- The corresponding outer join query using the new syntax will be

```
SELECT p.part_id, s.supp_name
    FROM Parts p FULL OUTER JOIN Supplier s
        ON p.supp_id = s.supp_id ;
```

- **Suppose, we wanted to extract all of MOTIKA's orders from ORDER Table**
  - **But here we do not know the SNUM**

SELECT snum FROM Salespeople

WHERE sname = 'Motika' ;

1004

SELECT * FROM Orders

WHERE snum = 1004

Placing Queries inside one another

- SQL has ability to nest the queries within one another
- A **sub-query** is a **select-from-where** expression that is nested within another query
- Types of **sub-query**
  - **Nested Sub-Query**
    - Typically, the inner query generates values that are tested in the predicate of the outer query.
  - **Corelated Sub-Query**
    - In contrast, Inner Query is dependent on Outer Query

Placing queries inside one another – Netsed

```
SELECT * FROM Orders

    WHERE snum =

        (SELECT snum FROM Salespeople

            WHERE sname = 'Motika' ) ;
```

Here, In order to evaluate Outer Query, SQL first have to evaluate inner query (Sub-Query)

It will search through Salespeople Table where the Sname is equal to Motika and will extract Snum value of the row.

SELECT * FROM Orders WHERE snum = 1004

Placing queries inside one another

- Naturally,
  - Inner Query should select only one column
  - As well, the data type of this column should match to which it is being compared in predicate.

Subqueries that produce Multiple Rows

- Using subqueries that produce any no. of rows is **perfectly acceptable** if…
  - **We use special operator IN**
  - Operators – **BETWEEN, LIKE** should **not be** used with subqueries.
  - Here, IN defines a set of values one of which will match the other terms of predicate's equation.
  - This otherwise may not work with relational operator

Exercise – Sub-Queries

1. Write a Query to find all orders credited to the same salesperson who services Customer 2008
2. Write a Query to find out all orders that are greater than the average for Oct 4th
3. Write a Query to find all orders attributed to salespeople in London

Exercise – Sub-Queries

4. Write a query to find all the customers whose cnum is 1000 above the snum of Serres.
5. Write a query to count customers with ratings above San Jose's average rating.
6. Write a query to show each salesperson with multiple customers.

Exercise – Sub-Queries

7. Write a query to show salesperson who has customer with highest order on given date

Exercise – Sub-Queries - With DMLs

8. **Write a Query to increase the commission of all salespeople with total current orders above 3000.**
9. **Write a Query to insert all salespeople with more than one customer into Sales Table.**
10. **Write a Query to reduce the commission by 10% of the salespeople who have produced smallest order.**
11. **Write a Query to increase the commission by 10% of all salespeople who have been assigned at least 2 customers.**

Exercise – Sub-Queries

12. Write a Query to find the names and numbers of all salespeople who had more than one customer.

Exercise – Sub-Queries - With DMLs

11. Write a Query to increase the commission by 10% of all salespeople who have been assigned at least 2 customers.

Using Sub-Queries for DML statements

- As a Developer …
- Many times we may need to use sub-queries within any query that generates values for DML Statements like INSERT / DELETE / UPDATE.
- How can we add all salespeople who have customers in San Jose to a separate table ???

```
INSERT INTO Sales
  SELECT * FROM Salespeople
    Where snum = ANY
      (SELECT snum FROM Customers
        WHERE city = 'San Jose');
```

1. Here, First find out the salespeople for customers in San Jose

This is INNER Task

2. And accordingly find out their details from salespeople

3. Then insert them into other table

With Delete

- Remove the records from Customers Table to whom the service is given by the salespeople from London City

```
DELETE FROM Customers
    WHERE snum  =
       ANY
         (SELECT snum
              FROM Salespeople
                   WHERE city = 'LONDON';
```

With Update / Insert

1. Increase the commission of all salespeople with total current orders above 3000
2. Insert all salespeople with more than one customer into Sales Table
3. Reduce the commission by 10% of the salespeople who have produced smallest order.

1.

```
UPDATE Salespeople
   SET comm = comm + (comm*0.2)
         WHERE 3000 <
           ( SELECT SUM(amt) FROM Orders
              WHERE Snum = Salespeople.snum);
```

2.

```
INSERT INTO Sales100
    SELECT * FROM Salespeople
        WHERE 1 <
            (SELECT COUNT(*) FROM Customers
                WHERE snum =
                                Salespeople.snum);
```

- 3.

```
UPDATE Salespeople
  SET comm = comm – comm*0.1
    WHERE Snum IN
      (SELECT cnum FROM Orders a
        WHERE amt =
          (SELECT MIN(amt)
            FROM Orders b
              WHERE a.odate = b.odate ) );
```

With Delete and Update

- Remove the records from Customers Table to whom the service is given by the salespeople from London City
- Find all the ratings for each salesperson's customers and deletes the salesperson if 100 is among them.
- Increase the commission by 10% of all salespeople who have been assigned at least 2 customers
- Reduce the commission by 10% of the salespeople who have produced smallest order.

Expressions in Subqueries

- Find all the customers whose cnum is 1000 above the snum of Serres.

```
SELECT * FROM Customers
    WHERE cnum >=
        ( SELECT  snum + 1000
            FROM Salespeople
                WHERE sname = 'Serres') ;
```

Subqueries for HAVING

- Write a query to count customers with ratings above San Jose's average rating.

```
SELECT rating, count (cnum)
 FROM Customers
    GROUP BY rating HAVING rating >
         ( SELECT  avg ( rating )
              FROM Customers
                   WHERE city = 'San Jose') ;
```

Distinct with Subqueries

- To find all orders credited to the same salesperson who services Customer 2008

```
SELECT * FROM Orders
    WHERE snum =
        ( SELECT  DISTINCT snum
            FROM Orders
                WHERE cnum = 2008 ) ;
```

Aggregate Functions with Subqueries

- Find out all orders that are greater than the average for October 4

```
SELECT * FROM Orders
    WHERE amt >
        ( SELECT  AVG(amt)
            FROM Orders
                WHERE odate = '04-OCT-90') ;
```

```
Select * From Salespeople
    where 1 < (Select count(*) from Customers
        where Salespeople.Snum = Customers.Snum)

Select b.Odate, a.Snum, a.Sname
    From Salespeople a, Orders b
        Where a.Snum = B.Snum
            AND b.Amt = ( Select Max(Amt) From
  Orders
                    Where Orders.Odate = b.Odate)
```

Subqueries that produce Multiple Rows

- Using subqueries that produce any no. of rows is **perfectly acceptable** if…
  - **We use special operator IN**
  - Operators – **BETWEEN, LIKE** should **not be** used with subqueries.
  - Here, IN defines a set of values one of which will match the other terms of predicate's equation.
  - This otherwise may not work with relational operator

Subqueries that produce multiple rows with IN

- Find all orders attributed to salespeople in London

```
SELECT * FROM Orders
    WHERE snum IN
        ( SELECT  snum
            FROM Salespeople
                WHERE city = 'London') ;
```

Relational Algebra

- Relational Algebra is comprised of many operators.
- These operators takes tables (relations) as their operands.
- These operators are read-only
  - In the sense, they just read their operands and in turn return the result
  - They are not expected to update anything.

Operators in Relational Algebra

- **SELECT** σ
  - Conditional Retrieval of columns and rows
  - Mapped in SQL with SELECT Varities
- **UNION** $\cup$
  - Returns the tuples retrieved from either of the relations without duplicates
- **INTERSECT** $\cap$
  - Returns those tuples which are common to both the relations
- **DIFFERENCE – MINUS** $-$
  - Returns all tuples retrieved from the first relation, but not from the second relation

- **CARTESIAN** X
  - Works as a Product of all rows of the relations
- **PROJECT** $\prod$
  - Eliminates the duplicate values of the column in a relation
  - Mapped in SQL as DISTINCT
- **RENAME** ρ
  - Assigns additional reference for column discriminator
  - Mapped in SQL with table aliases

SET Operators in SQL

- SET Operators are used to combine the results of two or more queries.
- They will take SELECT queries as their operands.
- They are as follows
  - UNION
  - INTERSECT
  - MINUS

The UNION Operator

- Returns the records retrieved by both of the queries
- Eliminates duplicate records by default
- To retain duplicates, we use UNION ALL

Set Operators

- UNION:

```
SELECT * FROM emp100

UNION

SELECT * FROM emp200;
```

- UNION ALL: Gets all duplicates

```
SELECT * FROM emp100

UNION ALL

SELECT * FROM emp200;
```

The INTERSECT Operator

- Returns those rows which are common to both queries
- E.g. List the employees who are listed in both Emp100 & Emp200

```
SELECT * FROM Emp100
INTERSECT
SELECT * FROM Emp200;
```

The MINUS Operator

- Returns all rows retrieved by the first query, but not by the second query
- E.g. List the Employees who are only listed in Emp100 and not in Emp200

```
SELECT * FROM Emp100
MINUS
SELECT * FROM Emp200
```

Indexes

- Used to optimize the performance of query operation
- A database index has an ordered list of values, with pointers to the row in which the value and its corresponding data reside.
- Indexes help us to avoid FULL TABLE SCAN for a search operation

Indexes

- Without indexes, any query or data modification causes the SQL engine to search the referenced tables from the top down.

```
CREATE INDEX Emp_idx
  ON Emp(Empno);
```

1.Clustered Index

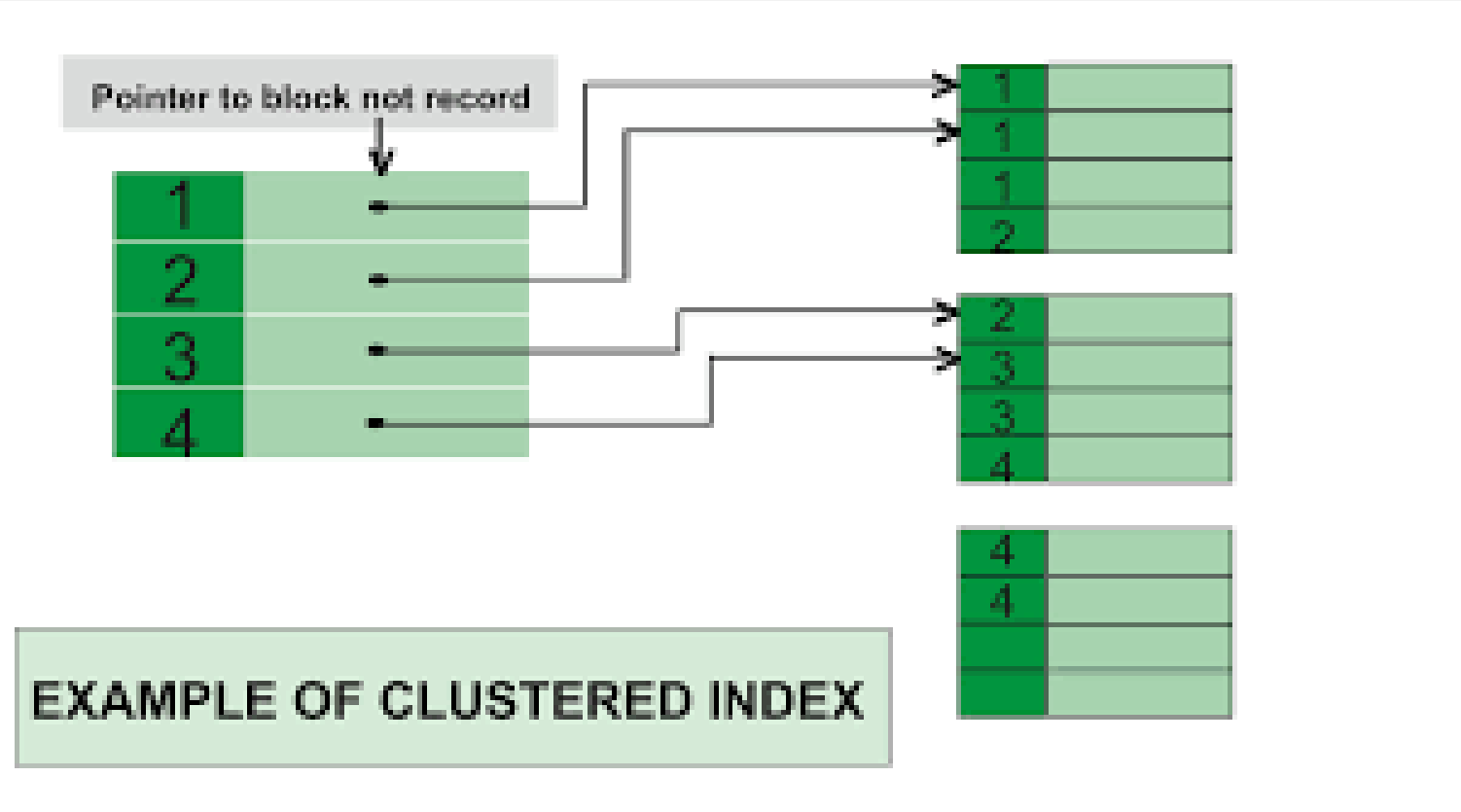
A clustered index is a table where the data for the rows are stored.

It defines the order of the table data based on the key values that can be sorted in only one direction.

In the database, each table can contains only one clustered index.

In a relational database, if the table column contains a primary key or unique key, MySQL allows you to create a clustered index named PRIMARY based on that specific column.

1.Clustered Index



2.Non Clustered Index

The indexes other than PRIMARY indexes (clustered indexes) called a non-clustered index.

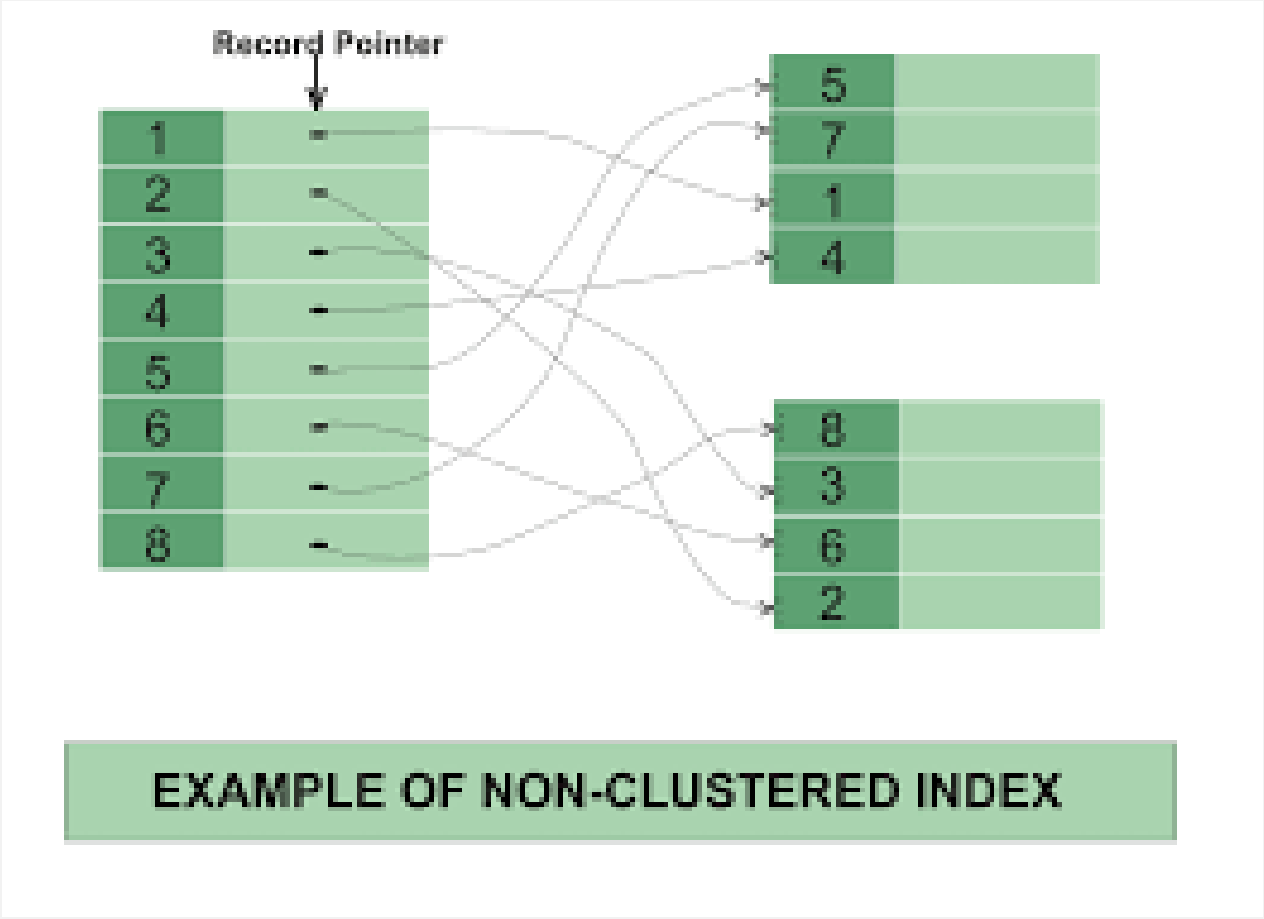
The non-clustered indexes are also known as secondary indexes.

The non-clustered index and table data are both stored in different places. It is not able to sort (ordering) the table data.

The non-clustered indexing is the same as a book where the content is written in one place, and the index is at a different place.

The non-clustered indexing improves the performance of the queries which uses keys without assigning primary key.

2.Non Clustered Index



Views

- Views are created from table data.
- Views do not contain data of their own.
  - Can be treated as **Virtual Tables**
- Views are used to present different views of the same data.
- **Views are typically used to restrict data access**

Views

- A view is essentially a query definition and does not contain any data
- A view is not a physical copy of data
- Every operation against a view executes the query contained within the view against all underlying tables.

Handling Views

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	240
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	170
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	170
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	90
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	42
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	58
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	35
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	31
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	26
					JUL-98	ST_CLERK	25
					JAN-00	SA_MAN	105
					MAY-96	SA_REP	110
					MAR-98	SA_REP	86
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	70
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	44
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	130
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	60
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	120
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	83

EMPLOYEE_ID	LAST_NAME	SALARY
149	Zlotkey	10500
174	Abel	11000
176	Taylor	8600

Typically used to restrict data access

Used to present different views of the same data

Creating Views

- CREATE VIEW view_name AS SELECT statement

```
CREATE VIEW   View1
        AS SELECT  empno, ename, sal
              FROM    emp
                    WHERE   deptno = 30;
```

View created.

- Create a view that stores each order and its salesperson and customer associated
- Create a view that stores all of the customers who have highest rating.

```
Select Odate "Order Date", Count(Distinct
    CNum) "Customers", Count(Distinct
ONum) "Order No", Avg(Amt) "Avg Amount",
    Sum(Amt) "Total Amount"
        From Orders
            Group By ODate
```

```
Select ONum,Amt,a.Snum,SName,CName
 From Orders a, Customers b, Salespeople c
      Where a.Cnum = b.CNum AND a.Snum =
c.Snum;

select * from Customers
 where rating = (select Max(rating) From
Customers)
```

Materialized views

- A materialized view materializes underlying physical data by making a physical copy of data from tables.
- So, unlike normal view as described previously,
- when a query is executed against a materialized view, the materialized view is physically accessed rather than the underlying tables.
- **This type of view frees the underlying table for other uses, effectively creating two separate physical copies.**