Fundamentals of Oracle

Oracle Data Types

Data types	Description
CHAR	This data type stores the string value, and the size of the string is fixed at the time of declaring the variable.
VARCHAR2	This data type stores the string, but the length of the string is not fixed.
DATE	This data type stores the values in date format, as date, month, and year. The standard Oracle date format for input and output is 'DD-MON-YY'
NUMBER	This data type stores fixed or floating point numbers up to 38 digits of precision. This data type is used to work with fields which will contain only number data.
LOB	This data type is mainly used to store and manipulate large blocks of unstructured data's like images, multimedia files, etc.

Creating Table

Syntax :

CREATE TABLE TABLE_NAME (columnName1 datatype(size)
.....,columnNamen datatype(size));

Description:

•A table name should be unique. it should not match with existing tables.
•table name and column name must start with alphabets , must not match with reserved keywords,and should be combination of A-Z,a-z,0-9 and '_'.
•Each column definition requires
- name
- data type
- size

•Table name and column name are not case sensitive.
•Each column definition is separated from other by a ','.
•The entire sql statement is terminated with ';'.

Example:

•CREATE TABLE STUDENT(ID NUMBER(2,0),NAME VARCHAR2(25));

Describing Table

• Logical Structure of table can be displayed using describe command.
Syntax:
• DESC tablename;
•It displays information about columns defined into the table. •displays column name , datatype ,size , and not null and key constraints of each column of table
Example:
•DESC student;

Inserting data

• To insert data into tables "INSERT INTO" sql statement is used.
• This statement creates a new row in a table , and stores the values into respective columns.

Syntax:

• INSERT INTO tablename (column1,column2 , ...) VALUES (exp1,exp2.....,expn);

Description:

• here, exp can be a constant , some variable , or mathematical expression.
• Character and Date constant can be enclosed within single quotes.
• There is one to one mapping between column specified and expression passed.
• if there are exactly the same no of values as there are columns in table , and their order is same as in the table definition , there is no need to indicate column names.
• if there are fewer values than the total number of columns in table , or their orders is not same as that of column order in the table definition , it is mandatory to indicate column names.

Example:

• INSERT INTO student(id,name) VALUES(1,'ABC');

Viewing Data

- **The SELECT statement retrieves data from a table and display them on screen.**
- **It is the most widely used and required statement among all others in SQL**
- Display all the records and all columns
- **Syntax:**
- SELECT * FROM tablename;
  - Here , * is used as the meta character and indicates all the columns of a given table.

Display Selected Columns all Rows

Syntax
• SELECT COL1, COL2, …,COLN FROM tablename;
Description
• This statement retrieves only selected columns as specified with SELECT clauses.
Example
• SELECT ID,NAME FROM STUDENT;

Eliminating Duplication

- **Sometime a table may contains duplicate rows.**
- **Duplicate row contains same value for each column in a table.**

- DISTINCT keyword removes duplicate rows from the result.
- It can be used only with SELECT statements.

Syntax

- SELECT DISTINCT column1 FROM tablename;

Example:

- SELECT DISTINCT CITY FROM student;

Sorting Data

Data can be retrieved in ascending or descending order based on specific column using Order by clauses.

Syntax:

- SELECT * FROM tablename ORDER BY column1 [order] , column2 [order];

Description :

- it retrieves data in sorted order.
- rows are sorted based on values of columns specified with ORDER BY clauses.
- By default , order is considered as ascending.
- To sort data in descending order it is necessary to specify DESC as an order.

Note:

- When multiple columns are provided and some rows contain same data for column then sorting is performed based on data of next column specified.

Example:

- SELECT * FROM STUDENT ORDER BY ID;
- SELECT * FROM STUDENT ORDER BY NAME DESC;

Displaying Selected rows and All columns

Syntax :
•SELECT * FROM tablename WHERE condition;
Description
•This statements retrieves only specific rows that satisfy the condition given with where clauses. •The condition can be provided in form of : column_name operator value •Multiple conditions can be combined with logical operator such as AND , OR
Example :
•SELECT * FROM student where ID > 2;

Displaying Selected Rows and selected columns

Syntax:
•SELECT column1, column2 ,.....,columnn FROM tablename WHERE condition;
Description:
•This statement retrieves only selected columns as specified with SELECT clauses. •Also , it retrieves only specific rows that satisfy the given condition with WHERE clauses.
Example:
•SELECT NAME FROM STUDENT WHERE ID=2;

Deleting Record From a Table

•Delete command can be used to remove either all rows of a table, or a set of rows from a table.

- Deleting All Rows
  - Syntax : DELETE FROM tablename;
  - Description :
    - This delete command deletes all the rows from the table , and display message regarding how many rows have been deleted.
  - Example :
    - DELETE FROM STUDENT;

- Deleting Specific Rows:
  - Syntax : DELETE FROM tablename WHERE condition;
  - Description :
    - This delete command deletes rows from the table that satisfy the condition provided by WHERE clauses.
    - It also display message regarding how many rows have been deleted.
  - Example:
    - DELETE FROM STUDENT WHERE ID > 2;

Updating Content of a Table

- **The update command can be used to change or modify data values in a table.**
- **It can be used to update either all rows or set of rows from a table**

- Updating All Rows
  - Syntax :
    - UPDATE tablename SET column1=exp1 , column2=exp2, ...,columnN=expN ;
  - Description:
    - This UPDATE command updates all the rows from the table , and displays message regarding how many rows have been updated.
    - Here , SET clauses specifies which column data to modify.
    - An exp can be a constant value , a variable or some expression and it specifies the new value for related column.
  - Example:
    - UPDATE student SET college_name = "GPG";

- Updating Specific Rows
  - Syntax :
    - UPDATE tablename SET column1=exp1 , column2=exp2, ...,columnN=expN WHERE condition;
  - Description:
    - This UPDATE command updates rows from the table that satisfy the condition provided by WHERE clause and display message regarding how many rows have been updated.
  - Example:
    - UPDATE student SET name="XYZ" WHERE ID=2;

Rename the Table

- **This command is used to rename an existing table.**

- Syntax :
  - RENAME TABLE old_table_name TO new_table_name;
- Description:
  - This RENAME command renames a table specified by old_table_name to new_table_name;
- Example:
  - RENAME TABLE student TO gpg_student;

Destroying the Table

- **This DROP command is used to destroy an existing table entirely.**

- Syntax :
  - DROP TABLE table_name;
- Description:
  - The DROP command drops the specified table.
  - This means , all records along with structure of the table will be destroyed.
  - Care must be taken while using this command , as all the records held within the table are lost and cannot be recovered.
- Example:
  - DROP TABLE student;

Truncating a Table

• The TRUNCATE command empties given table completely.
• it removes all the records from the given table.
• Logically this operation is equivalent to Delete command without WHERE clauses. But practically both are different.
• Truncate operation drops and recreates the table. This is much faster than deleting all records one by one.
• The deleted records cannot be recovered in truncate operation. while in delete operation , deleted records can be recovered using ROLLBACK statements.

•Syntax :
- TRUNCATE TABLE table-name;

•Example :
- TRUNCATE TABLE student;

Introduction TO SQL

Notebook:	Database Management		
Created:	25-04-2020 12:34	**Updated:**	25-04-2020 13:00
Author:	pagulearn.5334@gmail.com		

Introduction

- SQL, a Structured Query Language , is a language that provides an interface to relational database system.
- SQL was developed in IBM in 1970.
- Oracle at that time known as Relational Software , released the first commercial available implementation of SQL.
- it is pronounced as Sequel.
- SQL has been accepted as a standard language for RDBMS.

Features

- It is a non procedural language.
- It requires only to specify what operations to be done to retrieve data without specifying how to perform that operations.
- It is a english like language
- it can be used by wide range of users , even those who does not have programming experience.

The Commands Provided by SQL can be Categories into Following Category.

- **DDL : (Data Definition Language)**
    - It is a set of SQL commands used to create , modify and delete database objects such as tables ,views.
    - It is normally used by DBA and Database Designer.
        - **CREATE** : to create objects in database
        - **ALTER** : to alter the schema or logical structure of database
        - **DROP**: to delete objects from database
        - **TRUNCATE** : to remove all records from the table.

- **DML (Data Manipulation Language)**

- It is a set of SQL commands used to insert , modify and delete data in database
- it is normally used by general users who are accessing database via pre developed application
  - **INSERT** : to insert data ino table
  - **UPDATE** : to modify existing data in a table
  - **DELETE** : to delete records from table
  - **LOCK** : to lock tables to provide concurrency control among multiple users,

- **DQL (Data Query Language)**

- it is a SQL command that allows data retrieval from the database
- **SELECT** command
  - it a heart of SQL and allows data retrieval in different ways.

- **DCL (Data Control Language)**

- it is set of SQL commands used to control access to data and database
- **COMMIT** : to save work permanently
- **ROLLBACK:** undo work and restore database to previous state
- **SAVE POINT**: to identify point in a transaction to which work can be undone
- **GRANT :** to give access privileges to users on the database
- **REVOKE** : to withdraw access privileges given to users on the database.

Practical 7:Write down SQL queries using DDL command (CREATE) to create databases and tables , display their logical structure using Describe command.

Notebook:	Database Management Practicals		
Created:	2/19/2020 7:03 PM	**Updated:**	2/27/2020 1:38 PM
Author:	pagulearn.5334@gmail.com		

Creating Database

Syntax :

- **CREATE DATABASE DATABASE_NAME ;**

Example:

- CREATE DATABASE **BANK;**

Creating Tables

Syntax :

- CREATE TABLE TABLE_NAME (columnName1 datatype(size) , ...,columnNamen datatype(size));

Description:

- A table name should be unique. it should not match with existing tables.
- table name and column name must start with alphabets , must not match with reserved keywords,and should be combination of A-Z,a-z,0-9 and '_'.
- Each column definition requires
  - name
  - data type
  - size
- Table name and column name are not case sensitive.
- Each column definition is separated from other by a ','.
- The entire sql statement is terminated with ';'.

Example:

- CREATE TABLE STUDENT(ID NUMERIC(2,0),NAME VARCHAR(25));

Describing a Table

- Logical Structure of table can be displayed using **describe** command.

Syntax:

- DESC tablename;

- It displays information about columns defined into the table.
- displays column name , datatype ,size , and not null and key constraints of each column of table

Example:

- DESC student;

Practical 8: Write down SQL queries using DML commands - INSERT to insert records into the tables and display thoes records using DQL commnad- SELECT

Notebook:	Database Management Practicals		
Created:	2/27/2020 1:41 PM	**Updated:**	3/22/2021 1:24 PM
Author:	pagulearn.5334@gmail.com		

Inserting data into the table

- To insert data into tables "INSERT INTO" sql statement is uaed.
- This statement creates a new row in a table , and stores the values into respective columns.

Syntax:

- INSERT INTO tablename (column1,column2 , ...) VALUES (exp1,exp2.....,expn);

Description:

- here, **exp** can be a constant , some variable , or mathematical expression.
- **Character** and **Date** constant can be enclosed within single quotes.
- There is one to one mapping between column specified and expression passed.
- if there are exactly the same no of values as there are columns in table , and their order is same as in the table definition , there is no need to indicate column names.
- if there are fewer values than the total number of columns in table , or their orders is not same as that of column order in the table definition , it is mandatory to indicate column names.

Note:

- Order of columns need not match with column order in table definition.

Example:

- INSERT INTO student(id,name) VALUES(1,'ABC');

Inserting NULL Values

- We can insert null values into a table using one of the following ways.
  - insert **null** value as an expression into related column , where null is a keyword representing null values.
    - INSERT INTO student VALUES(1,null);
  - Omit the related column name and data value for column into which null value is to be inserted.
    - INSERT INTO student(id) values(3);

Viewing Data of a Table

- The SELECT statement retrieves data from a table and display them on screen.
- It is the most widely used and required statement among all others in SQL

- Display all the records and all columns

- Syntax:

- SELECT * FROM tablename;
  - Here , * is used as the meta character and indicates all the columns of a given table.

Practical 10: Write down SQL queries which demonstrate the use of DELETE, UPDATE ,RENAME TRUNCATE and DROP Command.

Notebook:	Database Management Practicals		
Created:	3/5/2020 8:55 PM	**Updated:**	11/17/2022 12:05 PM
Author:	pagulearn.5334@gmail.com		

Insert following values in the table **Employee**.

emp_n	emp_name	emp_sal	emp_comm	dept _no
101	Smith	800		20
102	Snehal	1600	300	25
103	Adama	1100	0	20
104	Aman	3000		15
105	Anita	5000	50,000	10
106	Sneha	2450	24,500	10
107	Anamika	2975		30

Insert following values in the table **job**.

job_id	job_name	min_sal	max_sal
IT_PROG	Programmer	4000	10000
MK_MGR	Marketing manager	9000	15000
FI_MGR	Finance manager	8200	12000
FI_ACC	Account	4200	9000
LEC	Lecturer	6000	17000
COMP_OP	Computer Operator	1500	3000

Practical 10: Write down SQL queries which demonstrate the use of DELETE, UPDATE ,RENAME TRUNCATE and DROP Command.

- Delete details of all the employees whose emp_no id 103.
- Update values of emp_name whose emp_no is 105.
- Update max salary of Programmer to 20000.
- Increase salary of an employee to 10% of all the employees whose department no is 20.

- Rename table job to bank_job.

- Delete all the records from job table.

- Drop the Job table.

- Increase amount of all depositor with 1000rs for all the employee who has tdeposit amount after '1-SEP-1995'.

Deleting Record From a Table

- Delete command can be used to remove either all rows of a table, or a set of rows from a table.

- **Deleting All Rows**
  - **Syntax :** DELETE FROM tablename;
  - **Description :**
    - This delete command deletes all the rows from the table , and display message regarding how many rows have been deleted.
  - **Example :**
    - DELETE FROM STUDENT;

- **Deleting Specific Rows:**
  - **Syntax :** DELETE FROM tablename WHERE condition;
  - **Description :**
    - This delete command deletes rows from the table that satisfy the condition provided by WHERE clauses.
    - It also display message regarding how many rows have been deleted.
  - **Example:**
    - DELETE FROM STUDENT WHERE ID > 2;

Updating Content of a Table

- The update command can be used to change or modify data values in a table.
- It can be used to update either all rows or set of rows from a table

- **Updating All Rows**
  - **Syntax :**

        - UPDATE tablename SET column1=exp1 , column2=exp2, ...,columnN=expN ;
    - **Description:**
        - This UPDATE command updates all the rows from the table , and displays message regarding how many rows have been updated.
        - Here , SET clauses specifies which column data to modify.
        - An **exp** can be a constant value , a variable or some expression and it specifies the new value for related column.
    - **Example:**
        - UPDATE student SET college_name = "GPG";

- **Updating Specific Rows**
    - **Syntax :**
        - **UPDATE** tablename **SET** column1=exp1 , column2=exp2, ...,columnN=expN **WHERE** condition;
    - **Description:**
        - This UPDATE command updates rows from the table that satisfy the condition provided by WHERE clause and display message regarding how many rows have been updated.
    - **Example:**
        - UPDATE student SET name="XYZ" WHERE ID=2;

Rename the Table

- This command is used to rename an existing table.

- **Syntax :**
    - RENAME TABLE old_table_name TO new_table_name;
- **Description:**
    - This RENAME command renames a table specified by old_table_name to new_table_name;
- **Example:**
    - RENAME TABLE student TO gpg_student;

Destroying a Table:

- This DROP command is used to destroy an existing table entirely.

- **Syntax :**
    - DROP TABLE table_name;

- **Description:**
  - The DROP command drops the specified table.
  - This means , all records along with structure of the table will be destroyed.
  - Care must be taken while using this command , as all the records held within the table are lost and cannot be recovered.
- **Example:**
  - DROP TABLE student;

Truncating a Table

- The TRUNCATE command empties given table completely.
- it removes all the records from the given table.
- Logically this operation is equivalent to Delete command without WHERE clauses.But practically both are different.
- Truncate operation drops and recreates the table.This is much faster than deleting all records one by one.
- The deleted records cannot be recovered in truncate operation. while in delete operation , deleted records can be recovered using **ROLLBACK** statements.

- **Syntax :**
  - TRUNCATE TABLE table-name;
- **Example :**
  - TRUNCATE TABLE student;

Practical 15 : Write down SQL queries that make use of ALTER , TRUNCATE and DROP Command.

Notebook:	Database Management Practicals		
Created:	4/21/2020 9:17 AM	**Updated:**	6/19/2022 1:04 PM
Author:	pagulearn.5334@gmail.com		
URL:	https://www.thoughtco.com/change-column-name-in-mysql-2693874		

ALTER COMMAND

- The ALTER TABLE Command can be used to change the schema (Logical Structure) of Tables.

- **ADDING NEW COLUMNS:**

  - **Syntax :**

    - **ALTER TABLE tableName ADD (newColumnName datatype1(size) , newColumnName datatype2(size), ...);**

  - **Description**

    - This command adds a new column or columns in an existing table.
    - Initially , this column will not contain any data. if required data can be added for this column using UPDATE command.

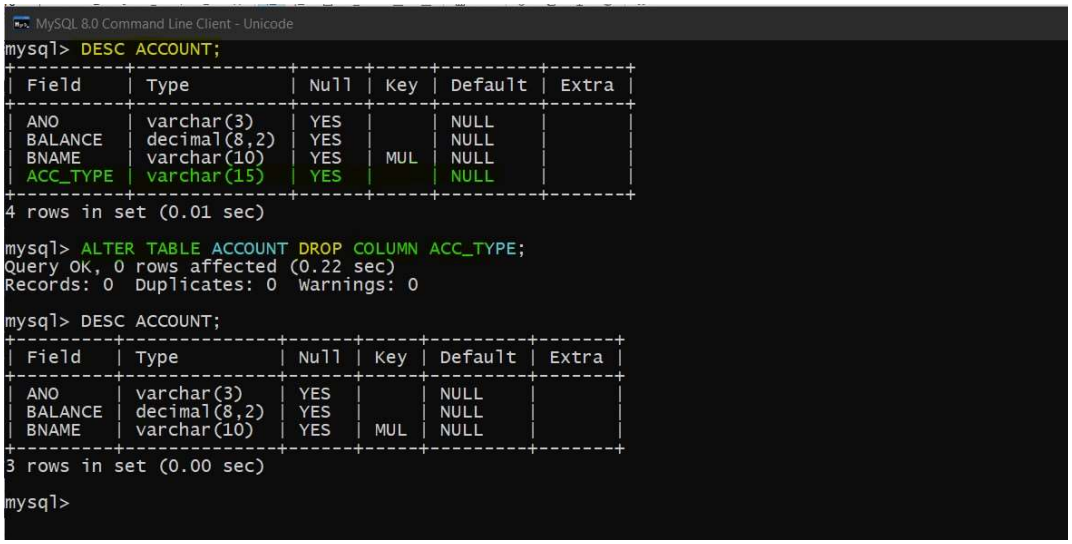
  - **Example :** Add new column ACC_TYPE specifying Account Type in Account table.

```
MySQL 8.0 Command Line Client - Unicode
mysql> DESC ACCOUNT;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| ANO     | varchar(3)  | YES  |     | NULL    |       |
| BALANCE | decimal(8,2)| YES  |     | NULL    |       |
| BNAME   | varchar(10) | YES  | MUL | NULL    |       |
+---------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> ALTER TABLE ACCOUNT ADD (ACC_TYPE VARCHAR(15));
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC ACCOUNT;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| ANO      | varchar(3)  | YES  |     | NULL    |       |
| BALANCE  | decimal(8,2)| YES  |     | NULL    |       |
| BNAME    | varchar(10) | YES  | MUL | NULL    |       |
| ACC_TYPE | varchar(15) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql>
```

- **Example :** ADD 2 new column POST , MANAGER of type VARCHAR in EMPLOYEE Table.

```
MySQL 8.0 Command Line Client - Unicode
mysql> DESC EMPLOYEE;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| eid       | varchar(4)  | YES  |     | NULL    |       |
| ename     | varchar(15) | YES  |     | NULL    |       |
| birthdate | date        | YES  |     | NULL    |       |
| salary    | decimal(9,2)| YES  |     | NULL    |       |
| city      | varchar(10) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)

mysql> ALTER TABLE EMPLOYEE ADD (POST VARCHAR(15) , MANAGER VARCHAR(3));
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC EMPLOYEE;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| eid       | varchar(4)  | YES  |     | NULL    |       |
| ename     | varchar(15) | YES  |     | NULL    |       |
| birthdate | date        | YES  |     | NULL    |       |
| salary    | decimal(9,2)| YES  |     | NULL    |       |
| city      | varchar(10) | YES  |     | NULL    |       |
| POST      | varchar(15) | YES  |     | NULL    |       |
| MANAGER   | varchar(3)  | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

- **DROPPING A COLUMN**
  - **Syntax:**
    - **ALTER TABLE tableName DROP COLUMN columnName;**
  - **Description**
    - This Column deletes an existing column from the table along with data held by that column.
  - **Example :** Drop Column ACC_TYPE from the ACCOUNT Table.



- **Modifying Existing Columns**

  - An Existing Column can be modified in two ways
    - Change the Datatype of a column
      - Requires datatype to be compatible. [CHAR to VARCHAR]
      - If datatype is not compatible , such as VARCHAR to NUMBER ,column needs to be empty.
    - Change the SIZE of a Column
      - Size can be increased without any problem. But to decrease the size of column , a column needs to be empty.

  - **Syntax :**

    - **ALTER TABLE tableName MODIFY (columnName newDatatype (newSize));**

  - **Description:**

    - This command sets newdatatype and size of the specified column.

  - **Example:** Alter column ANO to hold maximum 5 characters with datatype VARCHAR in table ACCOUNT.

```
MySQL 8.0 Command Line Client - Unicode
mysql> DESC ACCOUNT;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| ANO     | varchar(3)  | YES  |     | NULL    |       |
| BALANCE | decimal(8,2)| YES  |     | NULL    |       |
| BNAME   | varchar(10) | YES  | MUL | NULL    |       |
+---------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> ALTER TABLE ACCOUNT MODIFY ANO VARCHAR(5);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC ACCOUNT;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| ANO     | varchar(5)  | YES  |     | NULL    |       |
| BALANCE | decimal(8,2)| YES  |     | NULL    |       |
| BNAME   | varchar(10) | YES  | MUL | NULL    |       |
+---------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

- **Example**: Modify Column City 's Data type from VARCHAR to NUMERIC .

```
Select MySQL 8.0 Command Line Client - Unicode
mysql> SELECT * FROM ACCOUNT;
+------+---------+-------+
| ANO  | BALANCE | BNAME |
+------+---------+-------+
| A01  | 5000.00 | VVN   |
| A02  | 6000.00 | KSAD  |
| A03  | 7000.00 | ANAND |
| A04  | 8000.00 | KSAD  |
| A05  | 6000.00 | VVN   |
+------+---------+-------+
5 rows in set (0.00 sec)

mysql> ALTER TABLE ACCOUNT ADD (CITY VARCHAR(8));
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> UPDATE ACCOUNT SET CITY='SURAT';
Query OK, 5 rows affected (0.01 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM ACCOUNT;
+------+---------+-------+-------+
| ANO  | BALANCE | BNAME | CITY  |
+------+---------+-------+-------+
| A01  | 5000.00 | VVN   | SURAT |
| A02  | 6000.00 | KSAD  | SURAT |
| A03  | 7000.00 | ANAND | SURAT |
| A04  | 8000.00 | KSAD  | SURAT |
| A05  | 6000.00 | VVN   | SURAT |
+------+---------+-------+-------+
```

```
MySQL 8.0 Command Line Client - Unicode
mysql> ALTER TABLE ACCOUNT MODIFY CITY NUMERIC(5);
ERROR 1366 (HY000): Incorrect DECIMAL value: '0' for column '' at row -1
mysql>
```

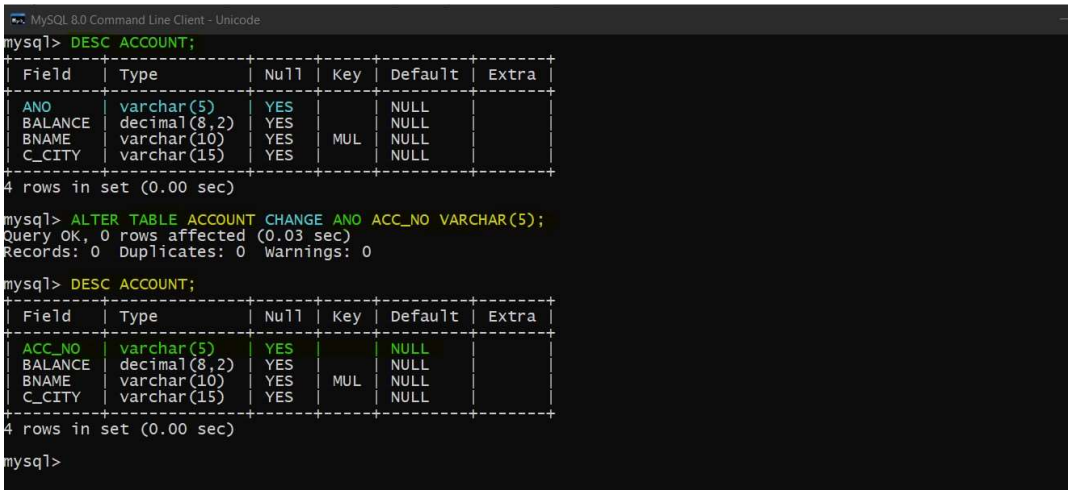
- In This example I have added new Column CITY with VARCHAR datatype .
- Add Values for CITY column in each record using UPDATE command
- Then try to modify data type for VARCHAR to NUMERIC as they are not compatible.
- If have not added any values for CITY column for each record and then try to change datatype from VARCHAR to NUMERIC then it is possible though they are not compatible.

- **Changing Name of a already Existing Column.**

- **Syntax:**

  - `ALTER TABLE tableName CHANGE oldname newName datatype(size) ;`

- **Example :** Change Name of Column ANO to ACC_NO in ACCOUNT Table



Destroying a Table:

- This DROP command is used to destroy an existing table entirely.

- **Syntax :**
  - DROP TABLE table_name;
- **Description:**
  - The DROP command drops the specified table.
  - This means , all records along with structure of the table will be destroyed.
  - Care must be taken while using this command , as all the records held within the table are lost and cannot be recovered.
- **Example:** Drop The Employee Table

```
MySQL 8.0 Command Line Client - Unicode
mysql> DESC EMPLOYEE;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| eid       | varchar(4)   | YES  |     | NULL    |       |
| ename     | varchar(15)  | YES  |     | NULL    |       |
| birthdate | date         | YES  |     | NULL    |       |
| salary    | decimal(9,2) | YES  |     | NULL    |       |
| city      | varchar(10)  | YES  |     | NULL    |       |
| POST      | varchar(15)  | YES  |     | NULL    |       |
| MANAGER   | varchar(3)   | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
7 rows in set (0.00 sec)

mysql> DROP TABLE EMPLOYEE;
Query OK, 0 rows affected (0.04 sec)

mysql> DESC EMPLOYEE;
ERROR 1146 (42S02): Table 'demo.employee' doesn't exist
mysql>
```

Truncating a Table

- The TRUNCATE command empties given table completely.
- it removes all the records from the given table.
- Logically this operation is equivalent to Delete command without WHERE clauses.But practically both are different.
- Truncate operation drops and recreates the table.This is much faster than deleting all records one by one.
- The deleted records cannot be recovered in truncate operation. while in delete operation , deleted records can be recovered using **ROLLBACK** statements.

- **Syntax :**
  - TRUNCATE TABLE table-name;
- **Example :**

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT * FROM EMPLOYEE;
+------+---------+------------+----------+-----------+------+---------+
| eid  | ename   | birthdate  | salary   | city      | POST | MANAGER |
+------+---------+------------+----------+-----------+------+---------+
| E01  | Tulsi   | 1982-01-26 | 12000.00 | Ahmedabad | NULL | NULL    |
| E02  | Gopi    | 1983-08-15 | 15000.00 | Anand     | NULL | NULL    |
| E03  | Rajshree| 1984-10-31 | 20000.00 | Vadodara  | NULL | NULL    |
| E04  | Vaishali| 1985-03-23 | 25000.00 | Surat     | NULL | NULL    |
| E05  | Laxmi   | 1983-02-14 | 18000.00 | Anand     | NULL | NULL    |
| E06  | Shivani | 1984-09-05 | 20000.00 | Surat     | NULL | NULL    |
+------+---------+------------+----------+-----------+------+---------+
6 rows in set (0.00 sec)

mysql> TRUNCATE TABLE EMPLOYEE;
Query OK, 0 rows affected (0.09 sec)

mysql> DESC EMPLOYEE;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| eid       | varchar(4)   | YES  |     | NULL    |       |
| ename     | varchar(15)  | YES  |     | NULL    |       |
| birthdate | date         | YES  |     | NULL    |       |
| salary    | decimal(9,2) | YES  |     | NULL    |       |
| city      | varchar(10)  | YES  |     | NULL    |       |
| POST      | varchar(15)  | YES  |     | NULL    |       |
| MANAGER   | varchar(3)   | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
```

Write Down SQL queries for the Followings.

Note :

- Write down sql queries for below based on the table : DEPOSIT,CUSTOMER,BORROW, BRANCH

1. Add a new Column called ACCOUNT_TYPE with VARCHAR datatype and size of 15 character in ACCOUNT table.
2. Add a new Column called LOAN_TYPE with VARCHAR datatype and size of 10 character in BORROW table.
3. Change the name of column BNAME to BRANCH_NAME in BRANCH table.
4. Remove Column BNAME from Borrow Table.
5. Modify size of Amount to NUMERIC(9,2) in DEPOSIT table.
6. Change the Datatype of CNAME FROM VARCHAR TO NUMERIC and Observe the output.a
7. Destroy Branch table along with data held in it.
8. Remove all the records from Borrower table ...

Practical 13 : WriteDown Sql queries that demonstrate the use of Scalar function. [Numeriacal , Character]

Notebook:	Database Management Practicals		
Created:	4/17/2020 10:24 AM	**Updated:**	12/30/2022 10:59 AM
Author:	pagulearn.5334@gmail.com		

Scalar Functions (Single Row Functions)

- A Function that **operates on a single row(or value) are called Scalar or Single row function.**
- These Function returns one row result for every row given as an input.
    - So if 5 rows are given as an input , there will be five result as an output.
- The Scalar Functions can be further be divided into sub groups based on datatype on which they operate.
    - Number Function
    - Character Function
    - Date Function
    - Conversion Function
    - Miscellaneous Function

Numeric Function [Arithmetic Function]

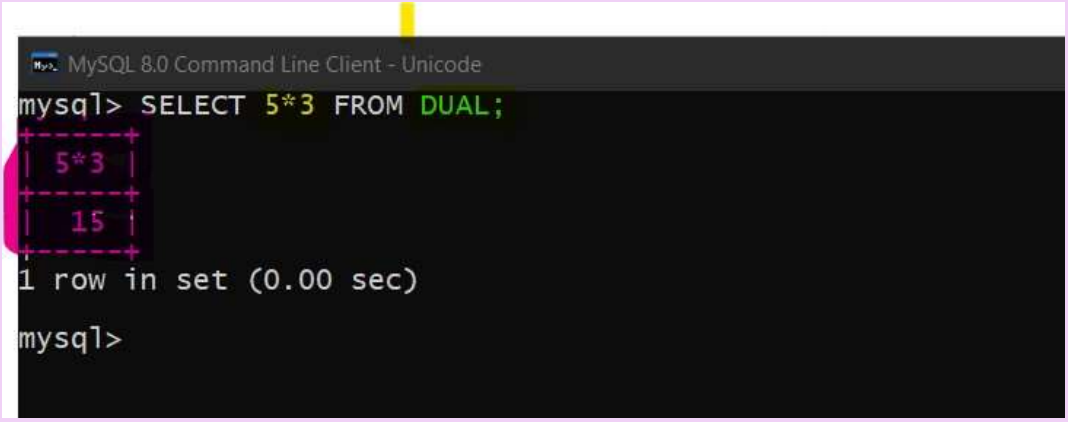
- These numeric functions can take constant or column of a table as an argument having numeric datatype.
- These Functions are scalar or single row , which returns individual result for each row passed as an input.
- In below Example We will passed constant value as an input to function for simplicity.

- **ABS**

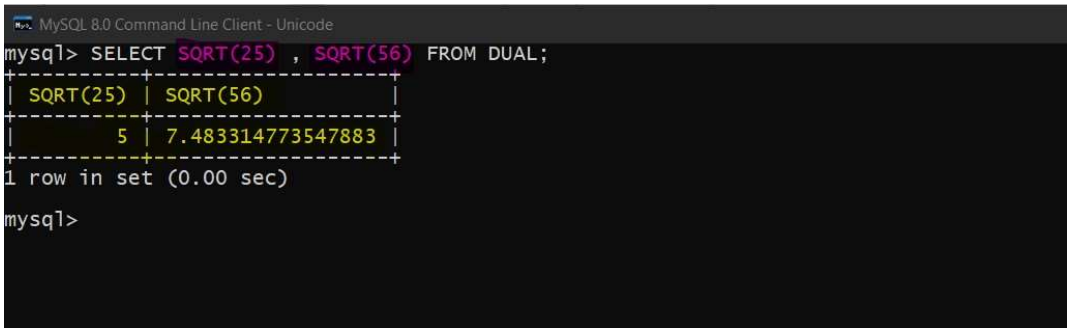
    - ABS(n)
        - Returns absolute value of 'n'
        - here , n can be constant , some expression or column name having numeric data type.

Note :

- Dual is a dummy table.
- it is provided by DBMS itself.
- If dual table is queried for records , it gives an output having only one row and one column with single value.
- In SQL , to display anything on screen SELECT is the available statement.
- But , In SELECT query we need to use Table name with FROM clauses.
- If we wanted to evaluate some expression 5*3 , we do not need user table .
- To display result of such query we provide DUAL table with SELECT.



- **SQRT**
    - SQRT(n)
        - it returns square root of 'n'
        - 'n' cannot be negative value.

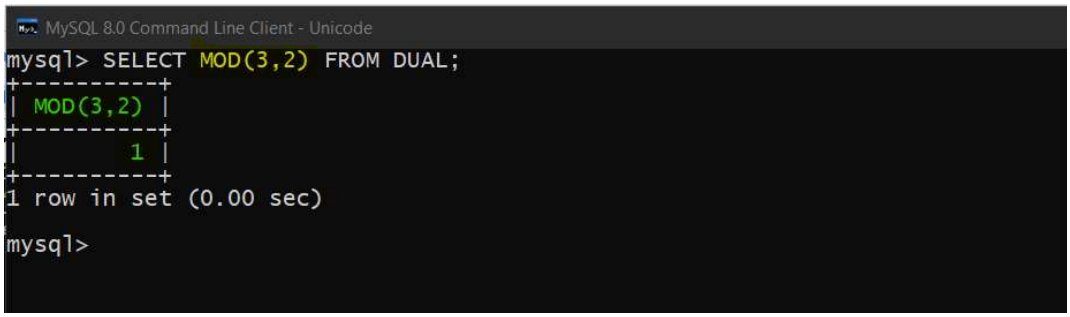


- **POWER**

  - POWER(m , n)
  - It return m raised to nth power.
  - n must be an integer value.

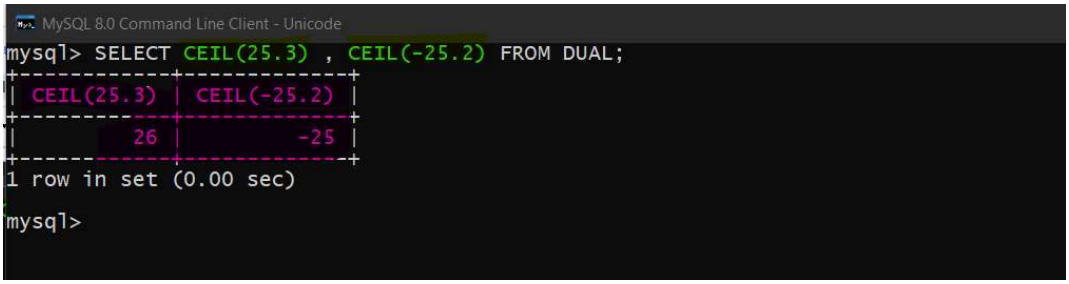
- **MOD**

  - MOD (m,n)
    - Returns remainder of a m divided by n operation



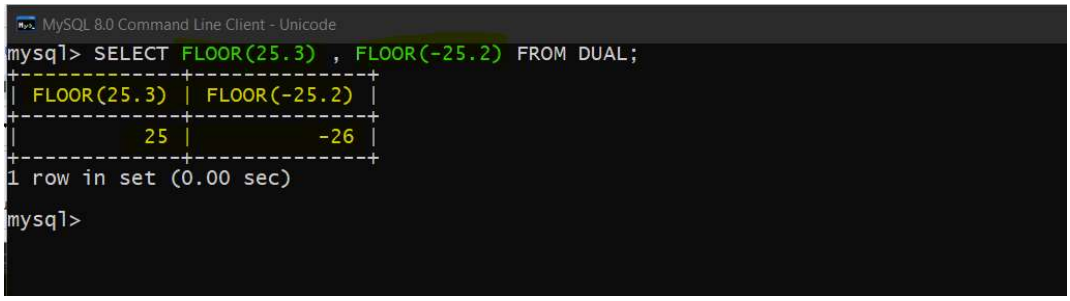
- **CEIL**

  - CEIL(n)
    - it returns the smallest integer value that is greater than or equal to n.



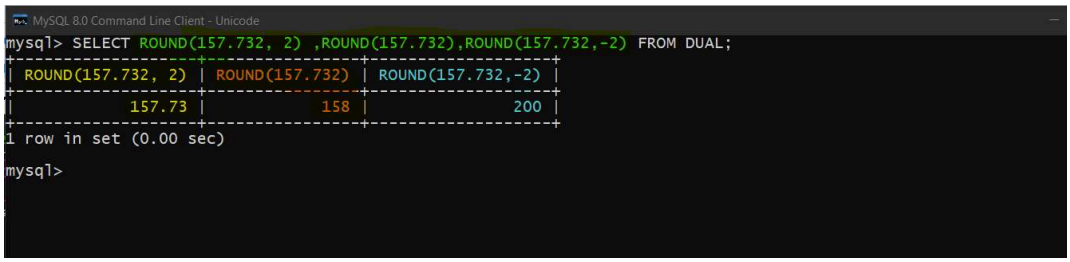
- **FLOOR**

- FLOOR(n)
  - Return the largest integer value that is less than or equal to n.



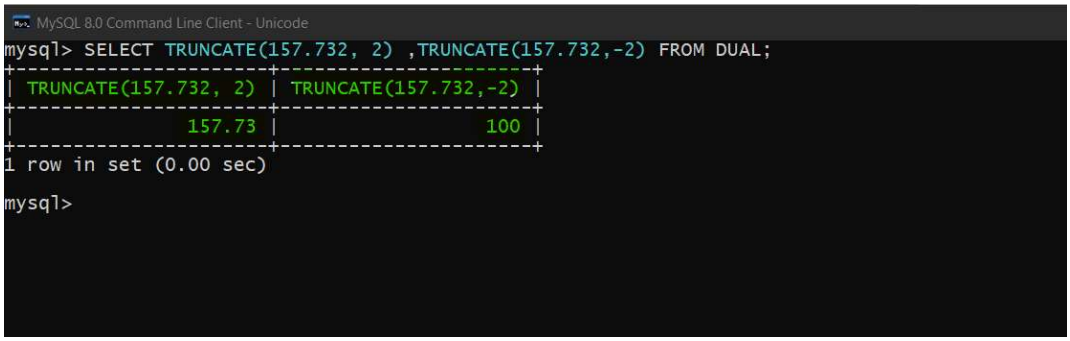
- **ROUND**

  - ROUND(m,n)
    - Returns m , rounded to n places to the right of a decimal point
    - if n is omitted , m is rounded to 0 places.
    - if n is negative , m is rounded to n places to the left of a decimal point
    - n must be an integer value



- **TRUNC**

  - TRUNC(m,n)
    - if n is positive , m is truncated to n places to the right of a decimal point.
    - if n is omitted , m is truncated to 0 places.
    - if n is negative , m is truncated to n places to the left of a decimal point.

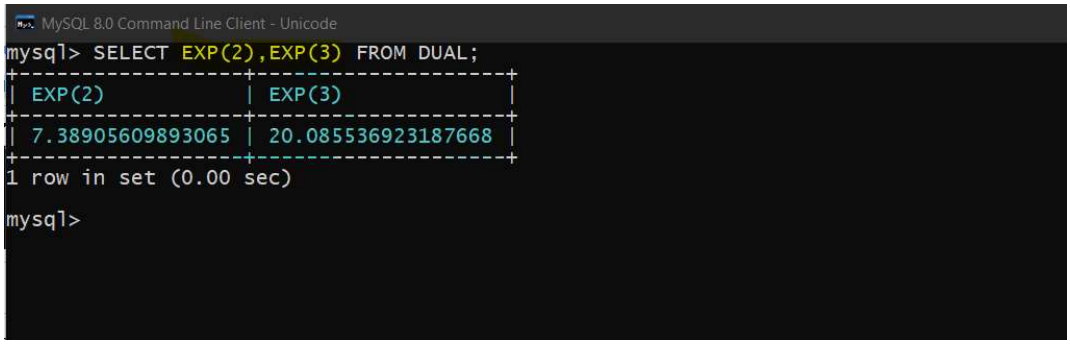


NOTE:

- In My SQL TRUNC() function does not work.
- Instead of TRUNC() function name is TRUNCATE() in MySQL.
- TRUNCATE() function does not work with single argument.

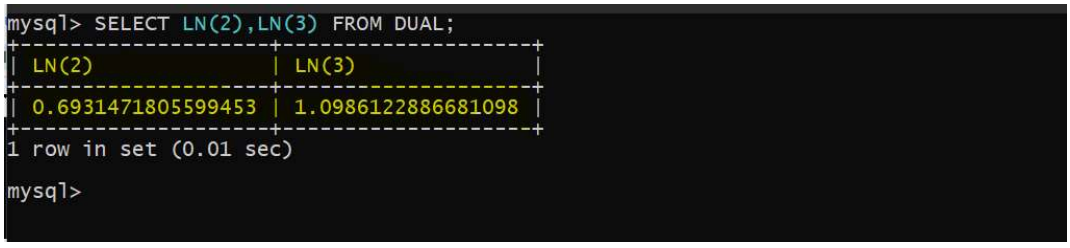
- **EXP**

    - EXP(n)
        - it returns e raised to nth power. where e=2.071828183



- **LN**

    - LN(n)
        - Returns natural . or base e , logarithm of n



- **LOG**

    - LOG(b,n)
        - Returns logarithm of the n in the base of b.

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT LOG(10,2),LOG(10,100) FROM DUAL;
+--------------------+-------------+
| LOG(10,2)          | LOG(10,100) |
+--------------------+-------------+
| 0.30102999566398114 |           2 |
+--------------------+-------------+
1 row in set (0.00 sec)

mysql>
```

- **COS , SIN ,TAN**

    - COS(n) ,SIN(n) , TAN(n)
        - Returns trigonometric cosine,sine and tangent values for n.
        - n is the angle in radius.

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT COS(3.1415),SIN(3.1415),TAN(3.1415) FROM DUAL;
+--------------------+----------------------+-------------------------+
| COS(3.1415)        | SIN(3.1415)          | TAN(3.1415)             |
+--------------------+----------------------+-------------------------+
| -0.9999999957076562 | 0.00009265358966049026 | -0.00009265359005819132 |
+--------------------+----------------------+-------------------------+
1 row in set (0.01 sec)

mysql>
```

- **COSH , SINH , TANH**

    - COSH(n) , SINH(n), TANH(n)

    - Returns hyperbolic cosine , sine and tangent values for n.
    - n is in the radius.

Note : this functions does not supports in MySQL.

- **SIGN**

    - SIGN(n)
        - returns -1 , if n is negative , 0 if n is zero , 1 if n is positive

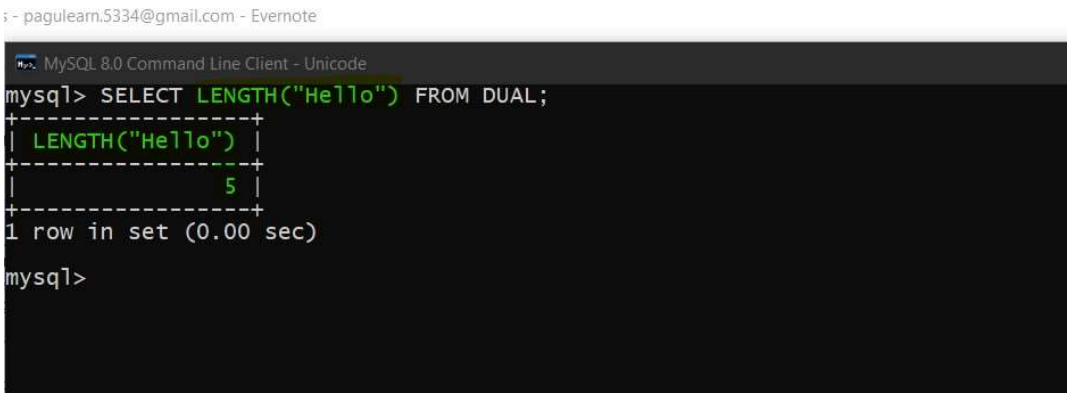
```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT SIGN(-25),SIGN(0),SIGN(25) FROM DUAL;
+-----------+---------+-----------+
| SIGN(-25) | SIGN(0) | SIGN(25) |
+-----------+---------+-----------+
|        -1 |       0 |         1 |
+-----------+---------+-----------+
1 row in set (0.00 sec)

mysql>
```

Character Function

- These functions are also scalar and single row functions.
- **Constant string** can be passed as a argument to these functions.
- Constant String need to be **enclosed within single quotes.**
- A **column name** of table having **varchar datatype** can also be **passed as argument**.

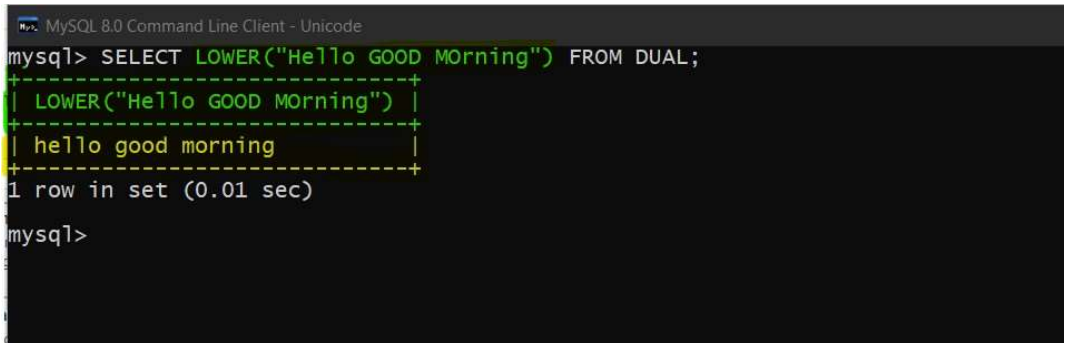
- **LENGTH**

    - LENGTH (str)
        - it will return length of a str.
        - here , str represent a string , which can be a constant string or column name having character data type.



- **LOWER**

    - LOWER(str)
        - returns str with all letters in lower case.



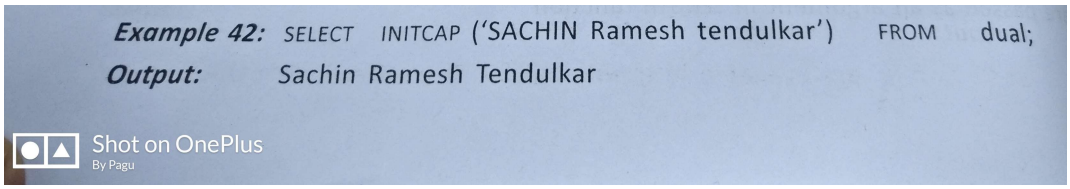
- **UPPER**

    - UPPER(str)
        - returns str with all letters in upper case

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT UPPER("hello GOOD MOrning") FROM DUAL;
+-----------------------------+
| UPPER("hello GOOD MOrning") |
+-----------------------------+
| HELLO GOOD MORNING          |
+-----------------------------+
1 row in set (0.00 sec)

mysql>
```

- **INITCAP**

    - INITCAP(str)
        - returns str with the first letter of each word in upper case.



Note : This function does not work in MySQL

- **SUBSTR**

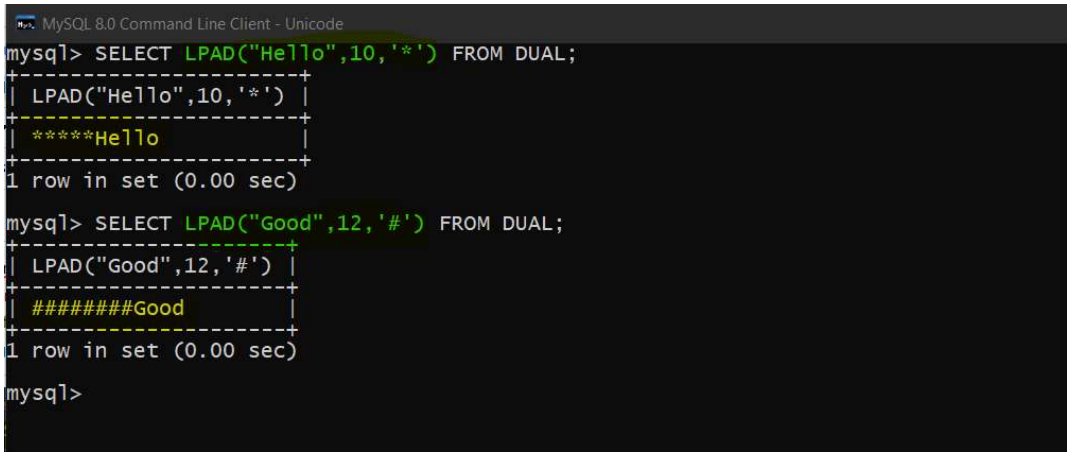
    - SUBSTR (str , pos , length)
        - return **portion** of **str** , beginning at **pos** and **going up to the length characters**.
        - the first position in the string is 1.
        - length indicates no of characters to extract.

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT SUBSTR("Morning",1,3) FROM DUAL;
+-----------------------+
| SUBSTR("Morning",1,3) |
+-----------------------+
| Mor                   |
+-----------------------+
1 row in set (0.01 sec)

mysql> SELECT SUBSTR("Morning",5,2) FROM DUAL;
+-----------------------+
| SUBSTR("Morning",5,2) |
+-----------------------+
| in                    |
+-----------------------+
1 row in set (0.00 sec)

mysql>
```

- **LPAD**

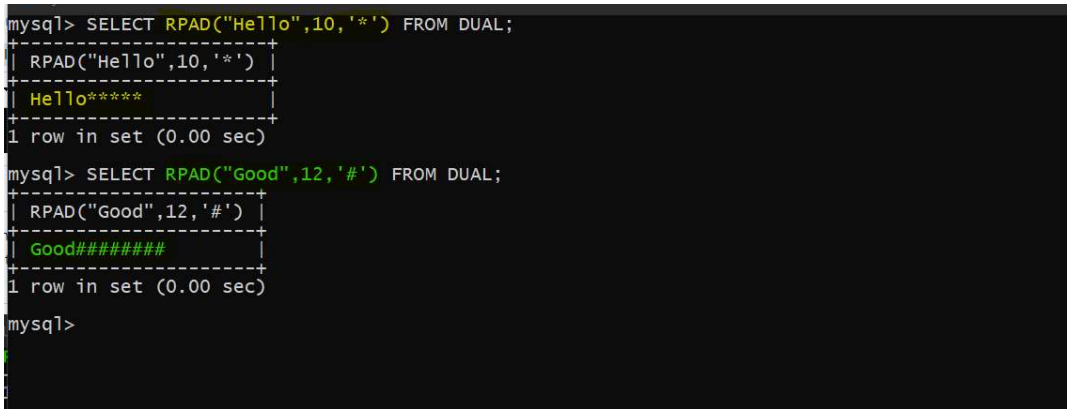
    - LPAD(str , n ,ch)

- it returns str left padded with character 'ch' based on the value of n



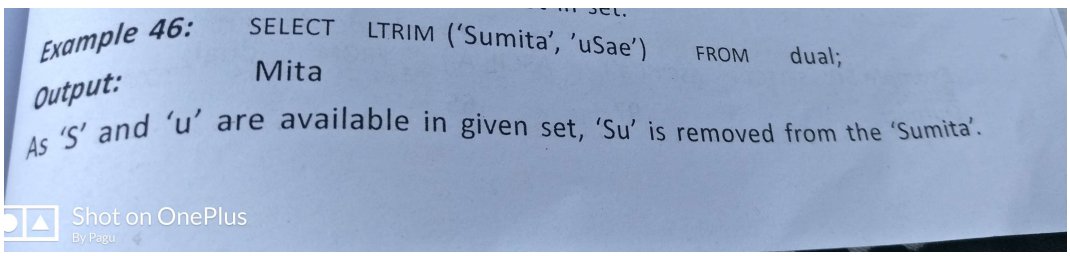
- **RPAD**

  - RPAD(str , n ,ch)
    - it returns str right padded with character 'ch' based on the value of n



- **LTRIM**

  - LTRIM(str , set)
    - Remove character from the left of str.
    - Character will be removed up to the first character not in set.



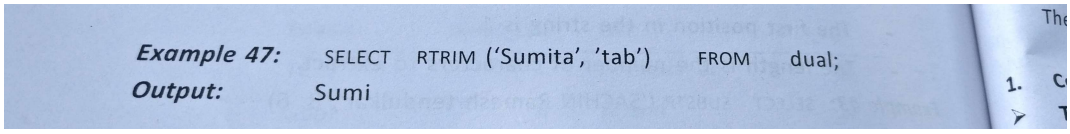
Note :

- **Syntax and functionality of LTRIM in Mysql is different**
- **In MySQL LTRIM can be used in following way**
  - **LTRIM(str)**
    - **it removes/trim space from left side of str.**

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT LTRIM('   sunita') FROM DUAL;
+--------------------+
| LTRIM('   sunita') |
+--------------------+
| sunita             |
+--------------------+
1 row in set (0.00 sec)

mysql>
```

- **RTRIM**

  - RTRIM(str,set)
    - it removes character from the right of str .
    - characters will be removed up to the first character not in set



Note :

- **Syntax and functionality of RTRIM in Mysql is different**
- **In MySQL RTRIM can be used in following way**
  - **RTRIM(str)**
    - **it removes/trim space from right side of str.**

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT RTRIM('sunita     ') FROM DUAL;
+----------------------+
| RTRIM('sunita     ') |
+----------------------+
| sunita               |
+----------------------+
1 row in set (0.01 sec)

mysql>
```

- **TRANSLATE**

- TRANSLATE (str,from_set,to_set)
  - characters of str that occur in from_set are translated to the corresponding character in the to_set.

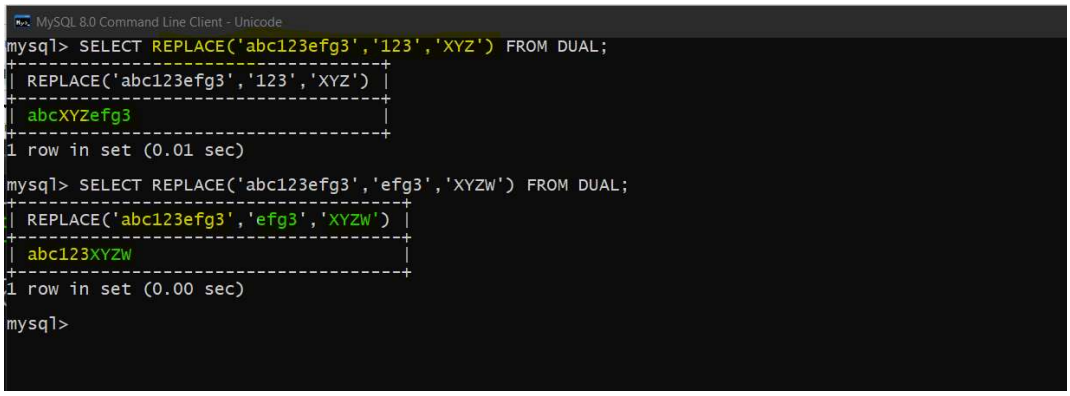
Example 48: SELECT TRANSLATE ('abc12efg3', '1234','XYZW') FROM dual;
Output: abcXYefgZ

Note :

- **This function does not work in MySQL.**

- **REPLACE**

  - REPLACE(str , from_set , to_set);
    - it is similar to translate , but it works **on whole string** rather than **individual character of set.**
    - **it replaces entire sub string instead of individual characters.**
    - **Sub string** of **str** which occurs in **from_set** is replaced with that of **to_str.**



- **ASCII**

  - ASCII(ch)
    - it returns the ascii code of ch.


MySQL 8.0 Command Line Client - Unicode
mysql> SELECT ASCII('a') , ASCII('A') FROM DUAL;
+------------+------------+
| ASCII('a') | ASCII('A') |
+------------+------------+
| 97 | 65 |
+------------+------------+
1 row in set (0.01 sec)
mysql>

SQL FUNCTIONS : (Scalar Functions)

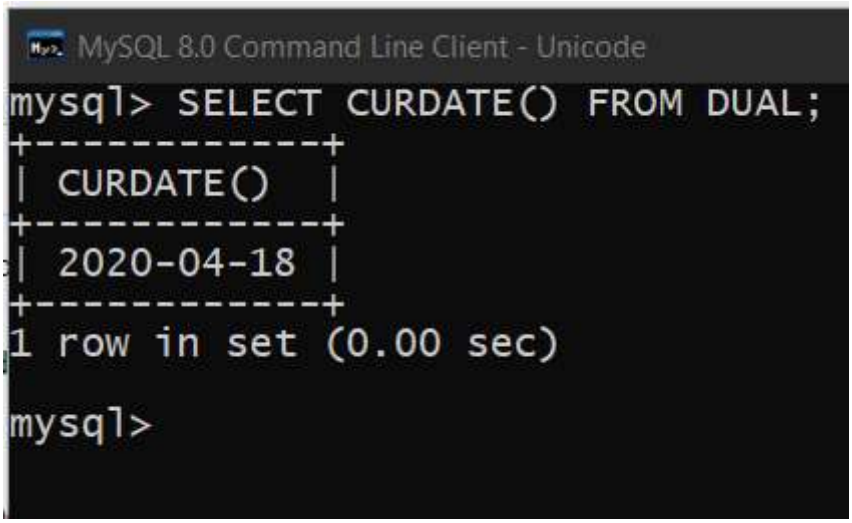
Notebook:	Database Management Practicals		
Created:	4/18/2020 9:27 AM	**Updated:**	4/18/2020 3:21 PM
Author:	pagulearn.5334@gmail.com		

NOTE :

- For the below SQL Scalar function i have not attached any MySQL Screen shots because these functions will work only in Oracle for MySQL different function are there.
- As the topic is important from theory point of view for 2,3 marks question i am just following the example given in your book which works in Oracle database.

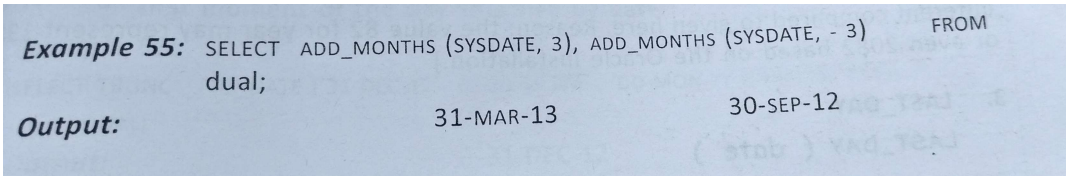
Date Functions

- Date function can take constant date values or column having date data type as an arguments.
- I MySQL , Date should be in form of 'YYYY-MM-DD' while specifying as a constant and it should be enclosed in single quotes.
- In Oracle , SYSDATE constant is used to get the current date .
- In MySQL , we will use CURDATE() function to get the current date.



- **ADD_MONTHS**

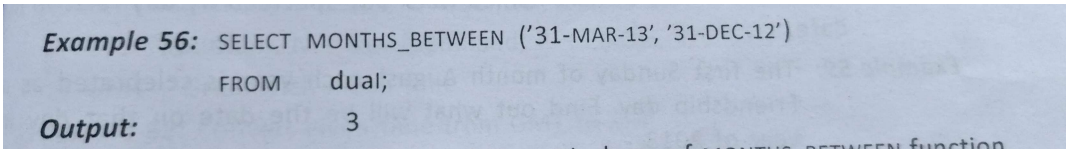
  - ADD_MONTHS(date , n)
    - Returns **new date** after **adding n months in date** specified by date.
    - if n is negative , then n month will be subtracted.



  - In this example , SYSDATE gives current date having month : December year : 2013
    - so,when n=3 it add 3 months it will display 31-MAR-13.
    - when n= -3 , it subtract and display 30-SEP-12.

- **MONTHS_BETWEEN**

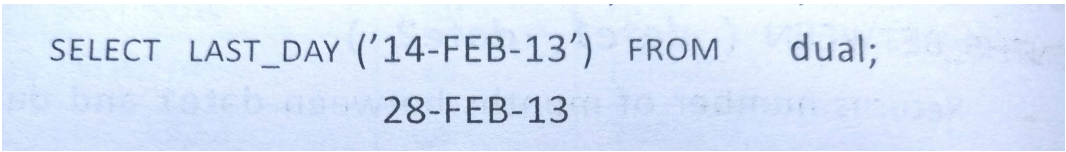
  - MONTHS_BETWEEN (date1,date2)
    - it returns **no of months between date1 and date2**
    - it subtract date2 from date1 to find out difference of months
    - Result may be positive , negative , zero in integer as well as real numbers.



  - In this example , exact difference between two dates [date1 - date2] is of 3 months.

- **LAST_DAY**

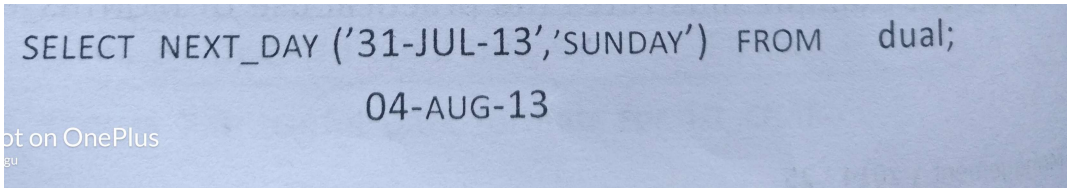
  - LAST_DAY(date)
    - it returns last date of the month specified by date.



- In this example specified date is 14th Feb 2013 , last date of Feb month is 28 so it displays 28-FEB-13

- **NEXT_DAY**

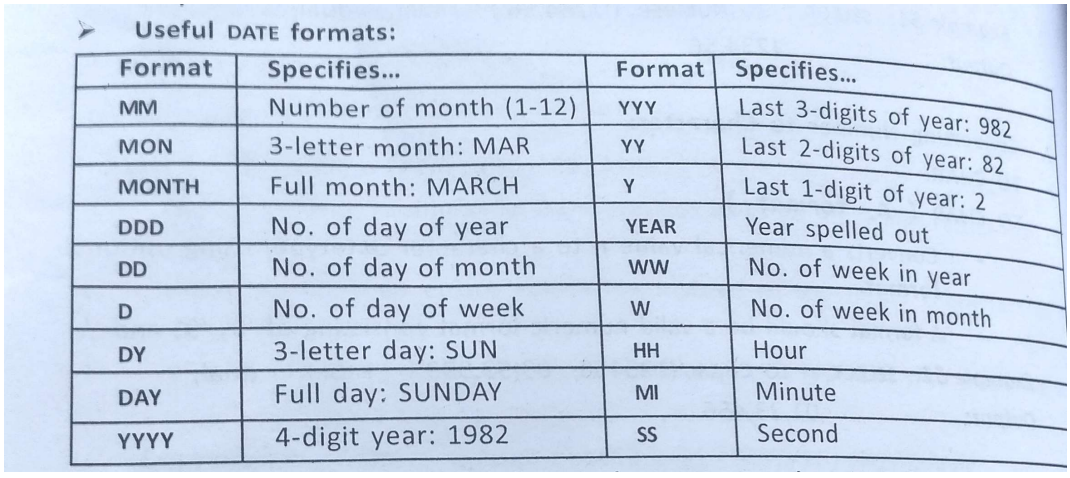
  - NEXT_DAY(date , day)
    - Returns the date of next named week day specified by day relative to date.



  - In this example , specified date '31-jul-13' and week day is 'SUNDAY'
    - So this indicates after '31-jul-13' whenever first 'SUNDAY' comes what will be the date at that day?
    - Answer this is : '4-AUG-13'

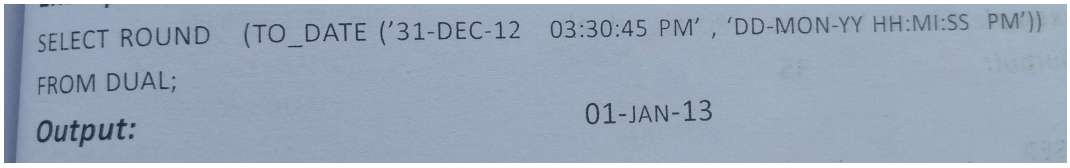
- **ROUND**

  - ROUND(date , format)
    - Returns rounded date according to format.
    - if format is omitted , **date is rounded to the next day** if the time is **12.00 PM or later** or to **today's date if before noon.**
    - A format should be valid one as shown in below table.



Useful DATE formats:

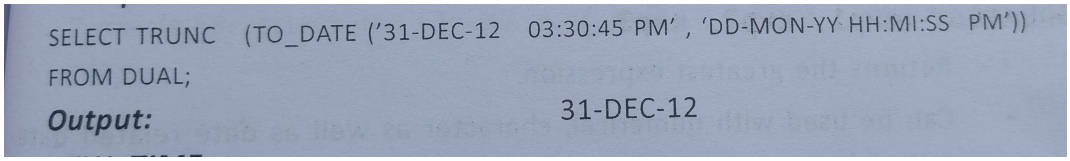
Format	Specifies...	Format	Specifies...
MM	Number of month (1-12)	YYY	Last 3-digits of year: 982
MON	3-letter month: MAR	YY	Last 2-digits of year: 82
MONTH	Full month: MARCH	Y	Last 1-digit of year: 2
DDD	No. of day of year	YEAR	Year spelled out
DD	No. of day of month	WW	No. of week in year
D	No. of day of week	W	No. of week in month
DY	3-letter day: SUN	HH	Hour
DAY	Full day: SUNDAY	MI	Minute
YYYY	4-digit year: 1982	SS	Second



- in this example the first argument of ROUND function is
  - TO_DATE('31-DEC-12 03:30:45 PM' , 'DD-MON-YY HH:MI:SS PM')
    - this function will convert the string : '31-DEC-12 03:30:45 PM' into date format.
- Second argument of Round function is omitted , so
  - we will get the result 01-JAN-13 because ROUND function round the date to next day if time 12.00 PM or later.

- **TRUNC**

  - TRUNC (date , format)
    - it returns truncated date according to format.
    - if format is omitted , **date is truncated to 12 Am (Mid night).**very first moment of the day specified by date.



  - In This example , the first argument of TRUNC function is
    - TO_DATE('31-DEC-12 03:30:45 PM' , 'DD-MON-YY HH:MI:SS PM')
      - this function will convert the string : '31-DEC-12 03:30:45 PM' into date format.
  - Second argument of TRUNC() is omitted ,
    - here , we will get the result '31-DEC-12' because TRUNC() will truncated the date to the very first moment of the day specified by the date ('31-DEC-12')

- **NEW_TIME**

  - NEW_TIME (date , zone1,zone2)
    - it returns the date after converting it from **time zone1** to **time zone2.**
    - Time zone are abbreviated in the form of XST or XDT
      - ST - Standard Time
      - DT - Daylight Time

```
SELECT NEW_TIME (TO_DATE
      ('31-DEC-12   12:00:00 AM' , 'DD-MON-YY HH:MI:SS  PM'),'GMT', 'PST')
FROM DUAL;
Output:                              30-DEC-12
```

- Here in this example , given time is converted from GMT(Greenwich mean time) to PST (Pacific Standard Time)

Conversion Function

- Conversion functions are used to **convert value from one data type to another.**

- **TO_NUMBER**

  - TO_NUMBER(str)
    - converts a value of a character data type , expressing a number to NUMBER data type.
      - It converts CHAR or VARCHAR to NUMBER
    - it returns equivalent numeric value to str.
    - str must consist of 0-9,decimal point , + or - sign.

```
Example 51:  SELECT    TO_NUMBER  ('1234.56')  FROM    dual;
Output:                1234.56
```

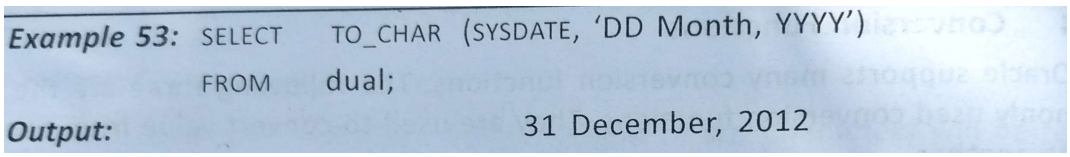
- **TO_CHAR**

    - TO_CHAR (n,format)
      - Converts a numerical value n to a character datatype , using optional argument format.
      - A Format must be a valid numeric format consisting of '0' , '9' and ','.

```
ple 52:  SELECT    TO_CHAR  (123456, '09,99,999')  FROM    dual;
lt:                01,23,456
```

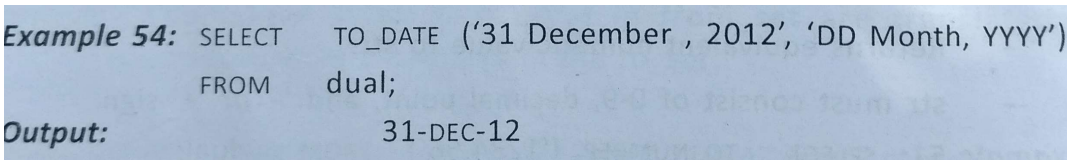
    - TO_CHAR (date, format)
      - it converts a Date datatype value date to a CHAR value using format.

- if format is omitted then default format 'DD-MON-YY' is used.
- Remember constant date values cannot be used as argument to this function. {'14-FEB-12'}



- **TO_DATE**

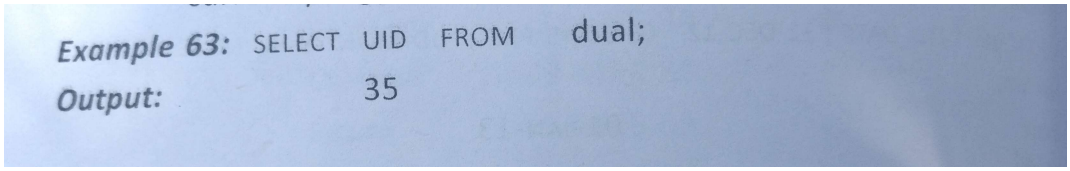
  - TO_DATE(str , format)
    - it converts a character value str into a DATE value.
    - A format specifies thee date format in which str contains the value.
    - The resultant date will be in default date format 'DD-MON-YY'.



Miscellaneous Functions

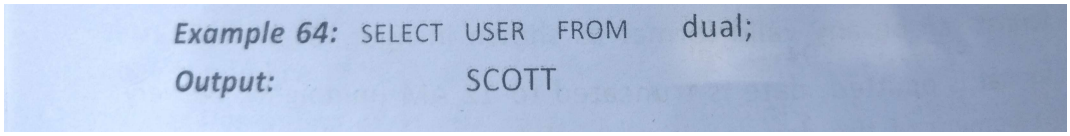
- **UID**

  - Returns an integer value corresponding to the UserID of the user currently logged in.



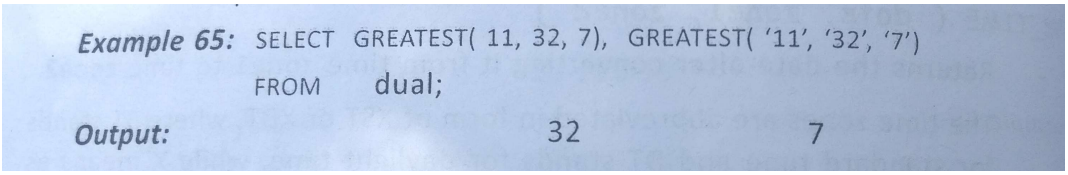
- **USER**

  - USER
    - returns the username of currently logged in user.



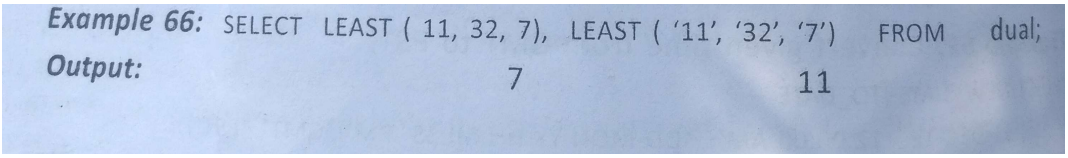
- **GREATEST**

  - GREATEST(exp1 , exp2 ,exp3 ....)
    - it returns the greatest expression
    - can be used with numerical , character and date related data.



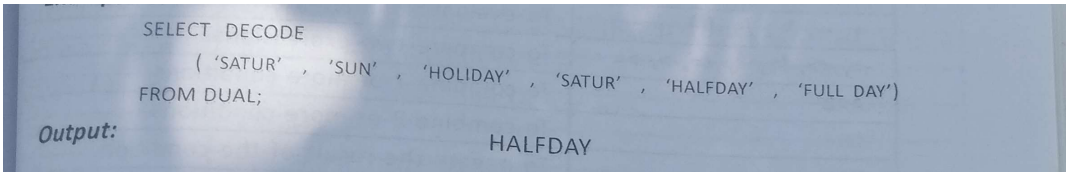
- **LEAST**

  - LEAST(exp1.exp2,exp3.....)
    - it returns the least expression



- **DECODE**

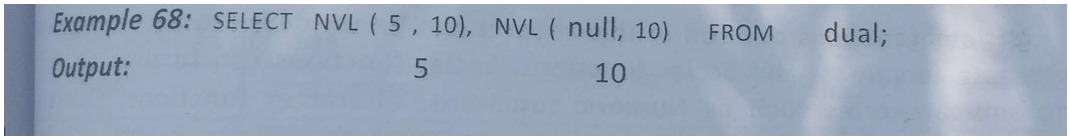
  - DECODE(value , if1 ,then1 , if2 , then2 ,.....,else)
    - it is similar to if then else construct in programming language.
    - A value is the value to be tested.
    - value is compared with if part whenever match is found , corresponding then part is returned.
    - if match not found with any of the if , then else will be returned.
    - function can be applicable to numeric, character and date type values.



- **NVL**

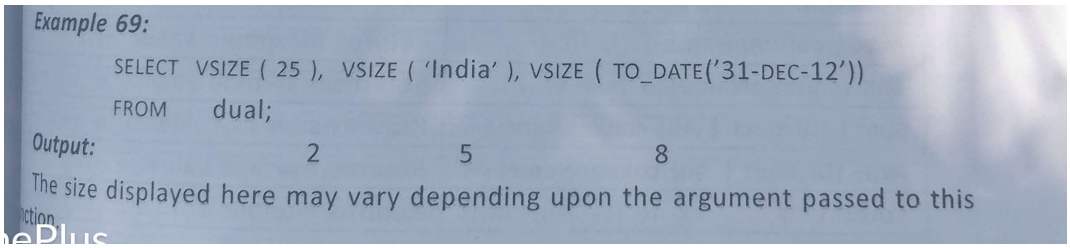
  - NVL(exp1 , exp2)
    - Returns exp2 , if exp1 is null

- Returns exp1 , if it is not null
- function can be applicable to numeric, character and date type values.



- **VSIZE**

  - VSIZE(exp)
    - Returns the storage size of exp in oracle.



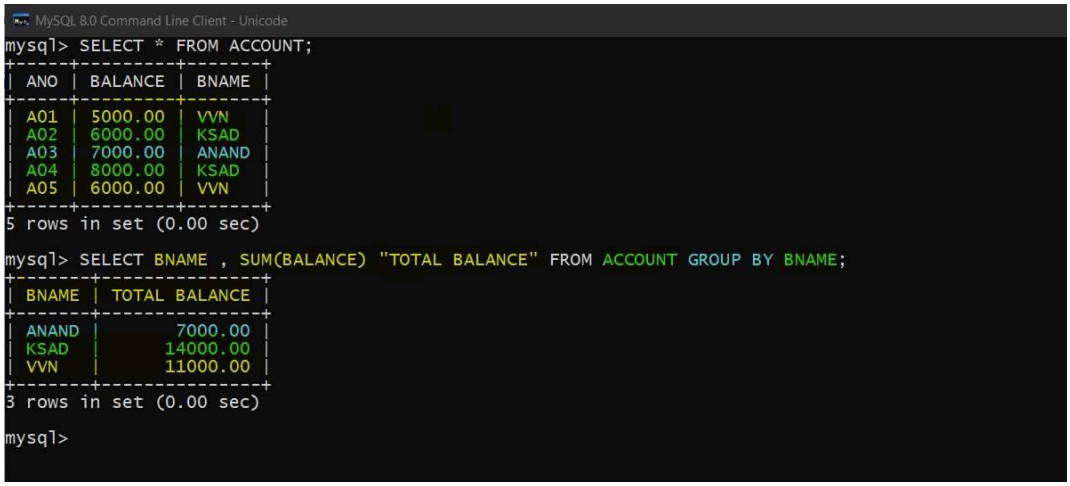
Practical 14: Write down SQL queries using GROUP By , HAVING clauses and concept of copying existing table.

Notebook:	Database Management Practicals		
Created:	4/20/2020 10:07 AM	**Updated:**	12/22/2022 10:57 AM
Author:	pagulearn.5334@gmail.com		

GROUP BY and HAVING

- The GROUP BY and HAVING clauses are used to **group the records of a table.**
- They are similar to ORDER BY and WHERE clause.
- The GROUP BY and HAVING clauses **act on set of records** , while ORDER BY and WHERE acts on **individual records.**

- **GROUP BY**
  - **Syntax :**
    - **SELECT Column1 , Column2 , ... , Aggregate_Function(argument) FROM Table_name GROUP BY Column1, Column2 ....,ColumnN**
  - **Description:**
    - The Group By clause groups records based on distinct values for the specified column.
    - It creates a dataset - containing several set of records grouped to gather based on condition.
  - **Example** : Display total balance of different account for each branch


MySQL 8.0 Command Line Client - Unicode
mysql> SELECT * FROM ACCOUNT;
+-----+---------+-------+
| ANO | BALANCE | BNAME |
+-----+---------+-------+
A01	5000.00	VVN
A02	6000.00	KSAD
A03	7000.00	ANAND
A04	8000.00	KSAD
A05	6000.00	VVN
+-----+---------+-------+
5 rows in set (0.00 sec)
mysql> SELECT BNAME , SUM(BALANCE) "TOTAL BALANCE" FROM ACCOUNT GROUP BY BNAME;
+-------+---------------+
| BNAME | TOTAL BALANCE |
+-------+---------------+
ANAND	7000.00
KSAD	14000.00
VVN	11000.00
+-------+---------------+
3 rows in set (0.00 sec)
mysql>

December 2016

03
Saturday

TABLE:- ACCOUNT

ANO	BALANCE	BNAME
A01	5000	VVN
A02	6000	KSAD
A03	7000	ANAND
A04	8000	KSAD
A05	6000	VVN

Query:- Display Total Balance of different account for each Branch.

- So, our output should look like This.

BNAME	Total Balance.
ANAND	7000
KSAD	14000
VVN	11000.

→ For this purpose, we need to Group the records of Account table Based on BNAME;-

04
Sunday

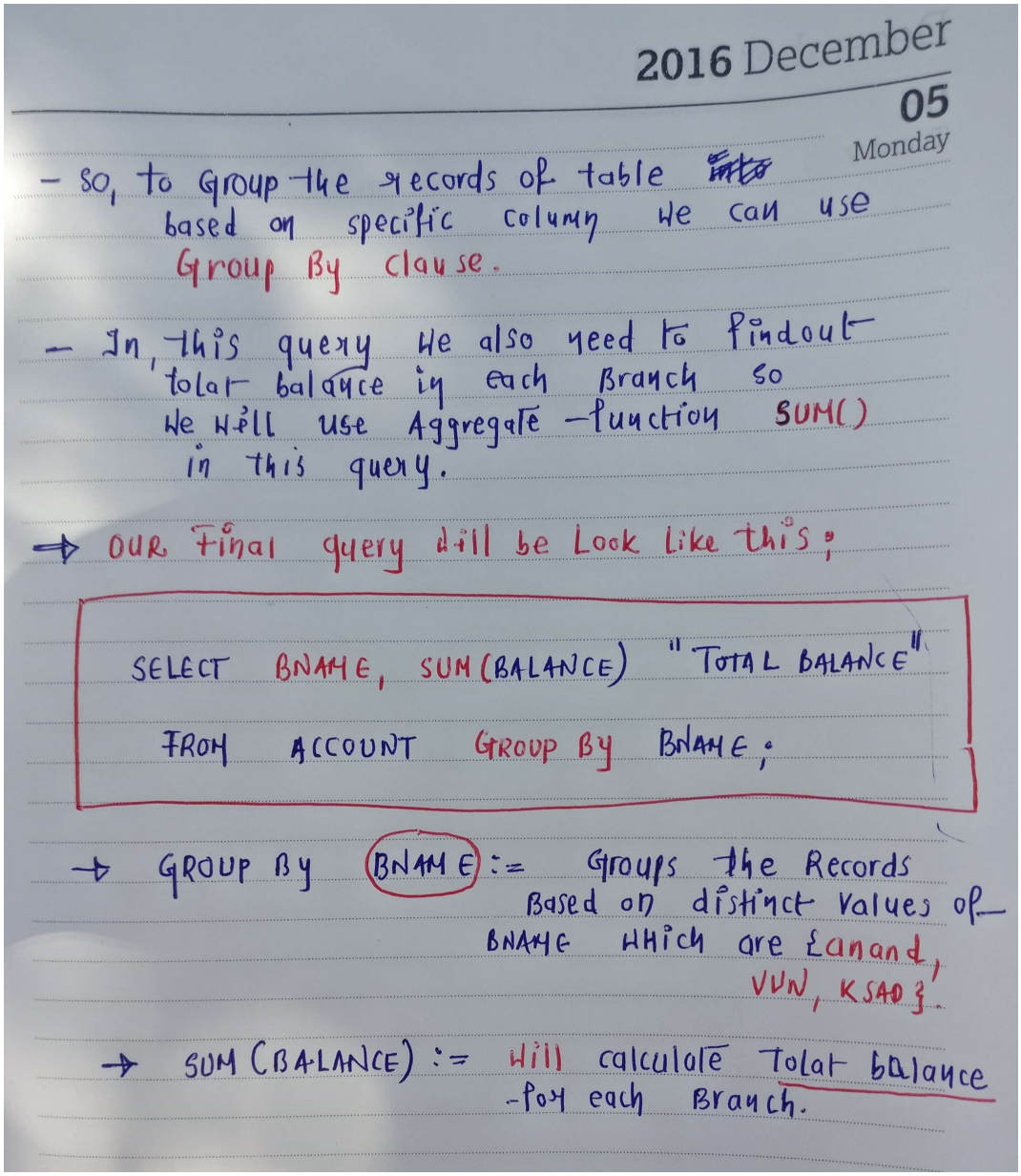
Like:

A01 A05	5000 6000	VVN VVN

A02 A04	6000 8000	KSAD KSAD

A03	7000	ANAND

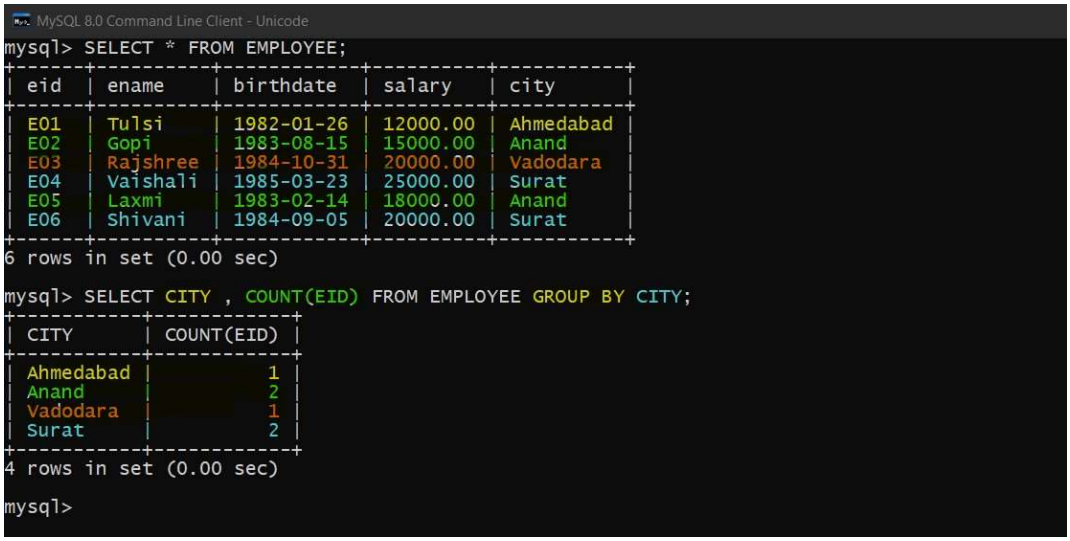
If your plans have failed, ask yourself if there's a better plan waiting to be discovered.



- Example : Display Total Salary of different employee for each city

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT * FROM EMPLOYEE;
+------+---------+------------+---------+-----------+
| eid  | ename   | birthdate  | salary  | city      |
+------+---------+------------+---------+-----------+
| E01  | Tulsi   | 1982-01-26 | 12000.00 | Ahmedabad |
| E02  | Gopi    | 1983-08-15 | 15000.00 | Anand     |
| E03  | Rajshree | 1984-10-31 | 20000.00 | Vadodara  |
| E04  | Vaishali | 1985-03-23 | 25000.00 | Surat     |
| E05  | Laxmi   | 1983-02-14 | 18000.00 | Anand     |
| E06  | Shivani | 1984-09-05 | 20000.00 | Surat     |
+------+---------+------------+---------+-----------+
6 rows in set (0.00 sec)

mysql> SELECT CITY , SUM(SALARY) "TOTAL SALARY" FROM EMPLOYEE GROUP BY CITY;
+-----------+--------------+
| CITY      | TOTAL SALARY |
+-----------+--------------+
| Ahmedabad |     12000.00 |
| Anand     |     33000.00 |
| Vadodara  |     20000.00 |
| Surat     |     45000.00 |
+-----------+--------------+
4 rows in set (0.00 sec)

mysql>
```

- **Example :** Display no of employee in each city.



- **HAVING**

  - **Syntax:**

    - **SELECT Column1 , Column2 , ... , Aggregate_Function(argument) FROM Table_name GROUP BY Column1, Column2 ....,ColumnN HAVING condition**

  - **Description:**

    - HAVING clause **filter the group** based on the specified condition.
    - Each Column specified in the HAVING clause must appear in the list of column specified in the GROUP BY clause or it must be appear within an aggregate function.
    - HAVING clause is similar to WHERE but ,
      - **WHERE** clause filters the **rows (records) based on the specified condition**
      - **HAVING** clause filters the **Set of records based on the specified condition.**

  - **Example :** Display total balance of different accounts for 'VVN' branch only.

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT * FROM ACCOUNT;
+-----+---------+-------+
| ANO | BALANCE | BNAME |
+-----+---------+-------+
| A01 | 5000.00 | VVN   |
| A02 | 6000.00 | KSAD  |
| A03 | 7000.00 | ANAND |
| A04 | 8000.00 | KSAD  |
| A05 | 6000.00 | VVN   |
+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> SELECT BNAME , SUM(BALANCE) "TOTAL BALANCE" FROM ACCOUNT GROUP BY BNAME HAVING BNAME='VVN';
+-------+---------------+
| BNAME | TOTAL BALANCE |
+-------+---------------+
| VVN   |      11000.00 |
+-------+---------------+
1 row in set (0.01 sec)

mysql>
```

- **Example :** Display Branch name and total balance for each branch having total balance greater than 12000.

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT * FROM ACCOUNT;
+-----+---------+-------+
| ANO | BALANCE | BNAME |
+-----+---------+-------+
| A01 | 5000.00 | VVN   |
| A02 | 6000.00 | KSAD  |
| A03 | 7000.00 | ANAND |
| A04 | 8000.00 | KSAD  |
| A05 | 6000.00 | VVN   |
+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> SELECT BNAME , SUM(BALANCE) "TOTAL BALANCE" FROM ACCOUNT GROUP BY BNAME;
+-------+---------------+
| BNAME | TOTAL BALANCE |
+-------+---------------+
| ANAND |       7000.00 |
| KSAD  |      14000.00 |
| VVN   |      11000.00 |
+-------+---------------+
3 rows in set (0.00 sec)

mysql> SELECT BNAME , SUM(BALANCE) "TOTAL BALANCE" FROM ACCOUNT GROUP BY BNAME HAVING SUM(BALANCE) > 12000;
+-------+---------------+
| BNAME | TOTAL BALANCE |
+-------+---------------+
| KSAD  |      14000.00 |
+-------+---------------+
1 row in set (0.00 sec)
```

- As You can see that , In the example of Group By only we have found out total balance for each Branch.
  - Now , if you want to filter out those (set of records) then we can use HAVING clause with Group By and Specific condition,
  - In this example I have added condition like ,
    - HAVING SUM(BALANCE) > 12000.
    - which indicates to filter out those branches whose total salary amount is greater than 12000.

Note :

- So with GROUP BY clause if you want to apply filter then you need to use HAVING clause with specific condition.

- Display City name along with total salary of employee it that city where city having total salary less than 30000.

Creating new Table from already existing one

- **Syntax:**

  - **CREATE TABLE newTableName (column1, column2 , ...,columnN) AS SELECT Column1,column2 , ...,columnN FROM sourceTableName WHERE condition.**

- **Description:**

  - this statement creates new table as **newTableName** from the already existing table **sourceTableName.**
  - You can provide different column name in newTableName table than the sourceTable.
  - The **column list** along with **newTableName** can be **omitted** , if **all columns of sourceTable** are being **copied without renaming.**
    - The meta character **'*'** can be used in **SELECT statement instead of specify column list.**
  - if WHERE clause is not provided in SELECT statement , then all the rows of sourceTable will be copied.
  - But to copy only selected rows from source table , a suitable WHERE clause should be provided.

- **Example :** Create a new table 'ACC' having all the rows and columns from the table Account.

```
MySQL 8.0 Command Line Client - Unicode
mysql> CREATE TABLE ACC AS SELECT * FROM ACCOUNT;
Query OK, 5 rows affected (0.13 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM ACC;
+-----+---------+-------+
| ANO | BALANCE | BNAME |
+-----+---------+-------+
| A01 | 5000.00 | VVN   |
| A02 | 6000.00 | KSAD  |
| A03 | 7000.00 | ANAND |
| A04 | 8000.00 | KSAD  |
| A05 | 6000.00 | VVN   |
+-----+---------+-------+
5 rows in set (0.00 sec)

mysql>
```

- **Example :** Create a new table 'ACC2' with only two columns ANO and BALANCE form already existing table ACOCUNT.

```
MySQL 8.0 Command Line Client - Unicode
mysql> CREATE TABLE ACC2 AS SELECT ANO, BALANCE FROM ACCOUNT;
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM ACC2;
+-----+---------+
| ANO | BALANCE |
+-----+---------+
| A01 | 5000.00 |
| A02 | 6000.00 |
| A03 | 7000.00 |
| A04 | 8000.00 |
| A05 | 6000.00 |
+-----+---------+
5 rows in set (0.00 sec)

mysql> CREATE TABLE ACC3 AS SELECT ANO, BALANCE FROM ACCOUNT WHERE BNAME='KSAD';
Query OK, 2 rows affected (0.04 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM ACC3;
+-----+---------+
| ANO | BALANCE |
+-----+---------+
| A02 | 6000.00 |
| A04 | 8000.00 |
+-----+---------+
2 rows in set (0.00 sec)
```

  - Here , second query with WHERE clause will copy only the records related to KSAD branch in ACC3 table.

- **Example :** Create a new table ACC4 with same structure as Account but does not contains any record from Account table

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT * FROM ACCOUNT;
+-----+---------+-------+
| ANO | BALANCE | BNAME |
+-----+---------+-------+
| A01 | 5000.00 | VVN   |
| A02 | 6000.00 | KSAD  |
| A03 | 7000.00 | ANAND |
| A04 | 8000.00 | KSAD  |
| A05 | 6000.00 | VVN   |
+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> CREATE TABLE ACC4 AS SELECT * FROM ACCOUNT WHERE BALANCE = 0;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM ACC4;
Empty set (0.00 sec)

mysql>
```

  - Here , In the ACCOUNT table doesnot contain any record with balance = 0.

- So , I have created new table ACC4 from Account with WHERE clause (balance = 0) , which will create ACC4 with same structure as ACCOUNT but no records will be copied as WHERE clause condition becomes false.
- That's Why SELECT * FROM ACC4;
  - Shows the result as Empty Set.

Inserting data into table from another table.

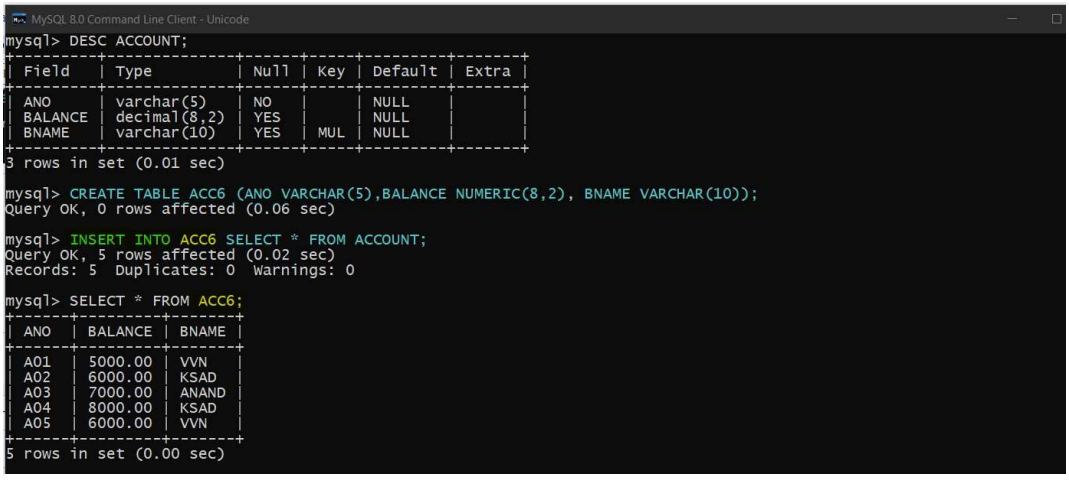
- **Syntax:**

```
INSERT INTO destinationTable
(column1,column2,.....,columnN)
SELECT column1,column2,.....,columnN FROM sourceTable
WHERE condition;
```

- **Description:**

- This statement copies records from source table to destination table.
- This statement does not create new table. So destination table must exist and it should have structure compatible with source table.
- The column list along with destination table can be omitted , if all columns of the source table are being copied.
- To insert only suitable rows , WHERE clause should be provided.

- **Example :** insert all the records of an account table into table named ACC6.



Write down SQL Queries for Followings .

- **Note**
  - **In For below query you need to refer BRANCH, CUSTOMER which we have used previously**
  - **In Some of the queries you need use new table called : Employee which is given below.**

Insert following values in the table **Employee**.

emp_n	emp_name	emp_sal	emp_comm	dept _no
101	Smith	800		20
102	Snehal	1600	300	25
103	Adama	1100	0	20
104	Aman	3000		15
105	Anita	5000	50,000	10
106	Sneha	2450	24,500	10
107	Anamika	2975		30

1. Count total no of customers in each city. [hint : City No of Customer]
2. Display total salary of Employee for each department. [hint : dept_no total_emp_salary]
3. Display total count of no of branch located in city named 'BOMBAY'.
4. Create new table Supplier from Employee with all the columns.
5. Insert data into 'Sup2' table from Employee table with only those records where Employee name second character is 'n'.
6. Create new table 'Sup1' Employee table with only first two columns.

Practical 18: WriteDown SQL queries to perform various Join Operations.

Notebook:	Database Management Practicals		
Created:	4/24/2020 10:10 AM	**Updated:**	5/22/2021 8:51 AM
Author:	pagulearn.5334@gmail.com		

Joins

- **It is possible to retrieve information from multiple tables using Join Operation.**
- In Relational Model table is used to represent relationship among data stored in database.
- For that , some columns are used as common columns in multiple tables.
- Such column act as a primary key in one table and as a foreign key in another table .
- Multiple tables can be joined using such columns. [datatype and size should be same in both tables for such columns.]
- In Set Operations rows are combined from two different tables.
- **The outcome of join operation is a wider table whose each row is a concatenation of two rows , one from each table.**

- Multiple Tables can be combined in different manners based on requirements.

- Join Operations can be classified into following categories.
  - **Cross Join [Cartesian Product /Simple Join]**
  - **Inner Join [Equi Join / Non -equi join]**
  - **Self Join**
  - **Outer Join**

- For Performing Join Operation we are going to use following two tables.

Account:

ano	balance	bname
A01	5000	vvn
A02	6000	ksad
A03	7000	anand
A04	8000	ksad
A05	6000	vvn

Branch:

bname	baddress
vvn	Mota bazaar, VVNagar
ksad	Chhota bazaar, Karamsad
anand	Nana bazaar, Anand

Figure 8.6: An Account and a Branch table

- **CROSS JOIN**

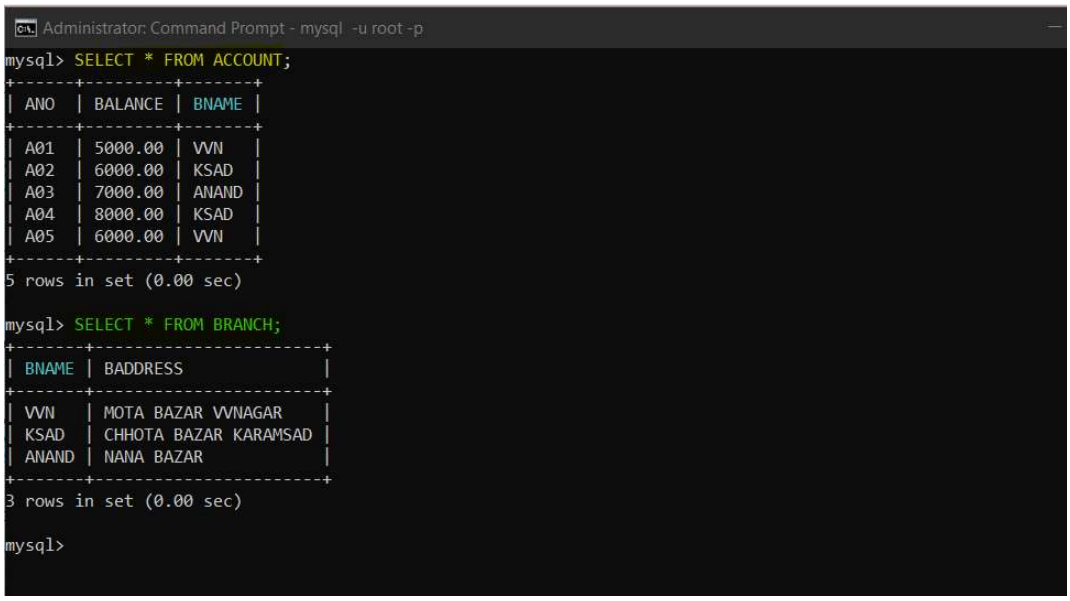
  - **The Cross Join combines every row from one table with every row in another table.**
  - The Output of Cross Join is similar to Cartesian Product.
  - This Operation is also called as Cartesian Product or Simple Join.
  - The Cross Join can be used when it is required to retrieve all possible combinations of rows and columns from both the table.

  - **Syntax**

    - **SELECT Column1,Column2,.....,ColumnN FROM table1 ,table2;**

    - Here , both the table contains **column with same name** then they can be differentiated by **tablename.columnName**

  - **Example :** Combine Information From ACCOUNT and BRANCH Table



- Output of Cross Join Operation Produces Inconsistent data
  - it also contains records in which BNAME of ACCOUNT table does not match with BNAME of Branch.
- But , Cross Join provides base for Inner Join , through Which we can retrieve consistent records.

- Here , Records highlighted with yellow lines are consistent records , all other records are inconsistent.

- **INNER JOIN:**

  - The Inner Join combines only those records which contains common values in both the tables.
  - So , It produces only consistent records in the output.

  - **Syntax:**

    - **SELECT Column1,Column2 ,...,ColumnN FROM table1,table2 WHERE table1.column1 OP table2.column2;**

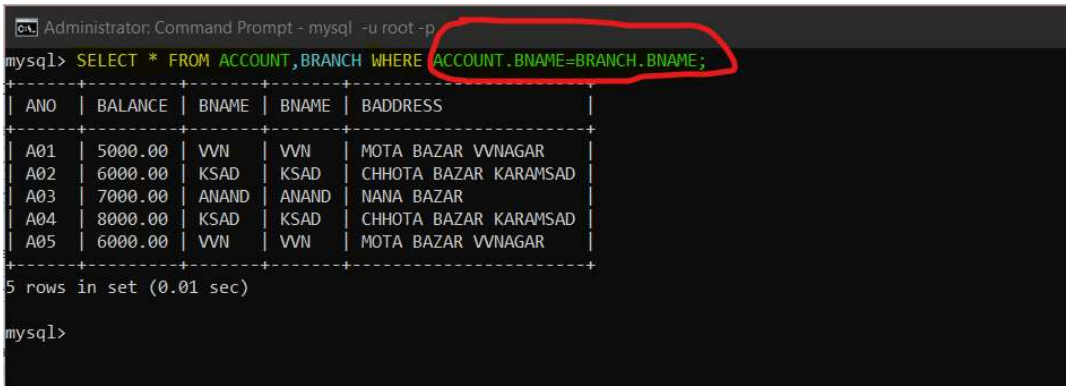
    - **SELECT Column1,Column2 ,...,ColumnN FROM table1 INNER JOIN table2 ON table1.column1 OP table2.column2;**

    - Here , **Column1 of table1 is usually Primary Key for table1 , While column2 of table2 is Foreign key which refer to column1 of table1.**
    - **OP** represent a **relational operator** which is used to compare values of column1 and column2.

  - **In Inner Join , if both the columns are compared for equality , OP is "=" then operation is referred as an Equi-Join.**

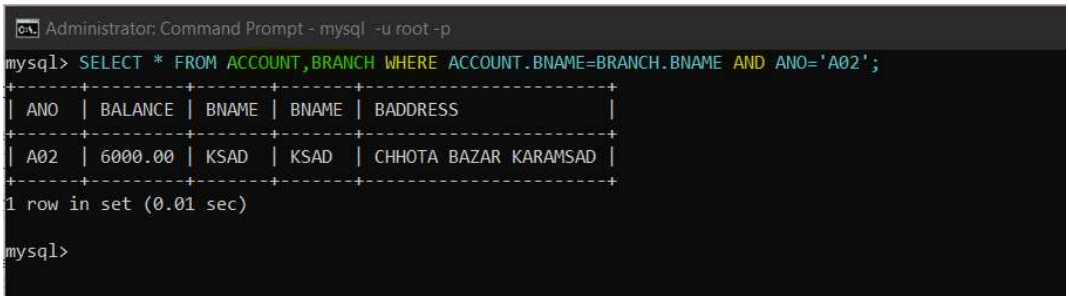
        - this join produce only those records which have common(same) values for both of these columns.
    - **In Inner Join, if both the columns are compared for non equality , then join operation is referred as Non-equi Join.**
        - Non equality comparison may include all other relational operator other than "=".

- **Example :** Combine only consistent information from Account and Branch table [Equality]



- Same INNER JOIN can also be performed using following Syntax.

- Here , we have used the "=" operator so it is an example of Equi Join.



- With Equi Join condition we can also use other condition along with Logical operator to retrieve information based on our

requirement.

- This can be considered as Non Equi join operation as we have used "<>" which is producing inconsistent data for this example.

- **SELF JOIN**

- **Self Join is the process of Combining(Joining) a table to it self**
- In Self join each row of a table is combined with other rows of the same table to produce result.
- Two different copies of same table is opened in memory . **So in FROM clause** table name is **specified twice** . To **distinguish two different instances , alias(temporary alternative name) of tables are created.**

- **Syntax:**

  - **SELECT Column1,Column2, ...,ColumnN FROM table alias1 , table alias2 [WHERE condition.]**

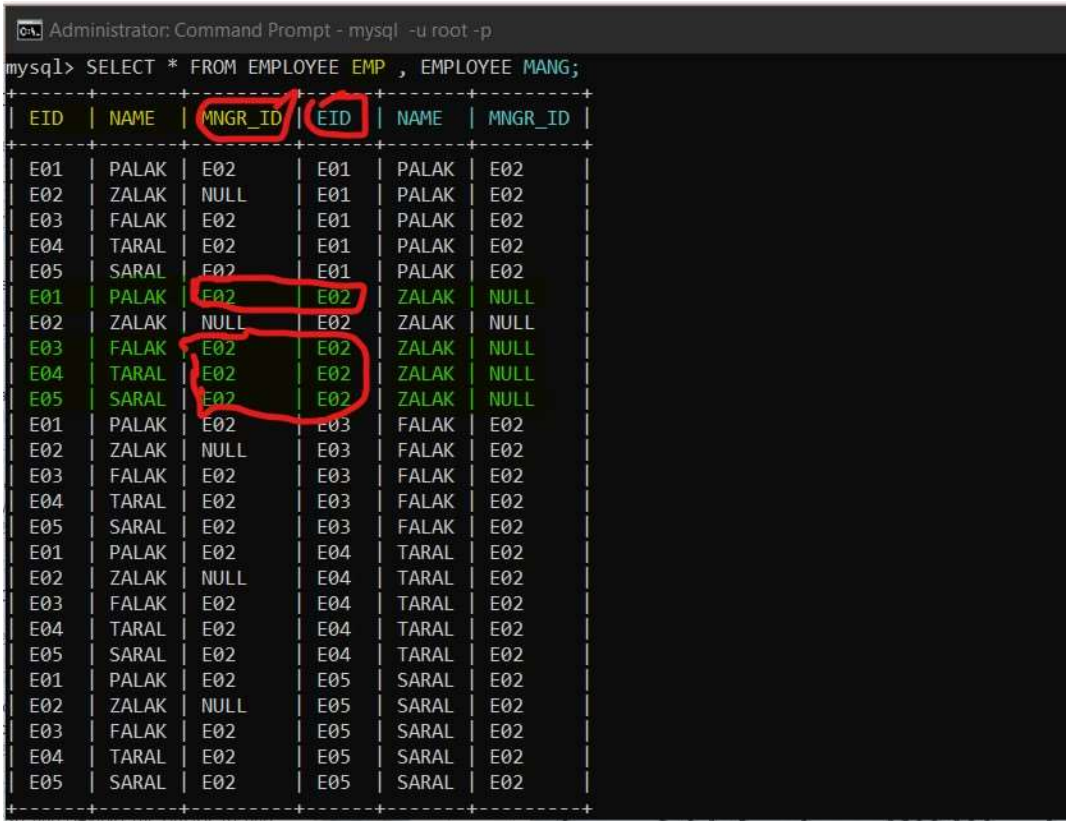
- **Example:** Display Employee id , name along with their manager Name.

Employee:

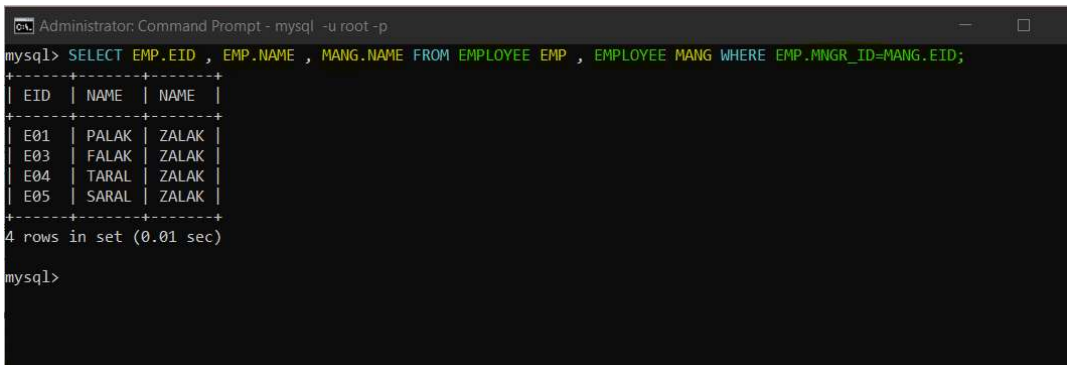
eid	name	mngr_id
E01	Palak	E02
E02	Zalak	null
E03	Falak	E02
E04	Taral	E02
E05	Saral	E02

nd Employee table

- To Solve above query , as we can see that eid of each employee's manager can be directly available.
- But need to find out name of that particular eid , which is "ZALAK" in our example.
- So we need to do SELF JOIN of EMPLOYEE table with it self
- Result of this self join operation would look like this



- As you can see that in this SELF JOIN operation We have provided alias name EMP and MANG to EMPLOYEE tables.
- So In the Result
  - First three columns highlighted with Yellow indicates EMP
  - LAST three Columns highlighted with Blue indicates MANG.
- Now , As you can see that highlighted Green records ,
  - This are the actual consistent records which indicates ,
    - Employee having Eid 'E01' , Name : PALAK , MNGR_ID : E02 [EMP] and that Manager having EID : 'E02' has Name : ZALAK [MANG]
- To get this consistent result we need to add WHERE clause with following condition
  - EMP.MNGR_ID =MANG.EID
  - Here , reason for this condition is ,
    - yellow MNGR_ID is Column of EMP
    - blue EID is column of MANG.



- Here Observe that you did not get E02 employee record because she is her self MANAGER.

OUTER JOIN

- Though Inner Join yields useful and consistent information it may omit some information in some situations , which result in information lost.
- Outer Join will deal with such kind of missing information.

- **Example :** To Understands drawback of INNER JOIN in some situation [Consider following two tables.]

College:

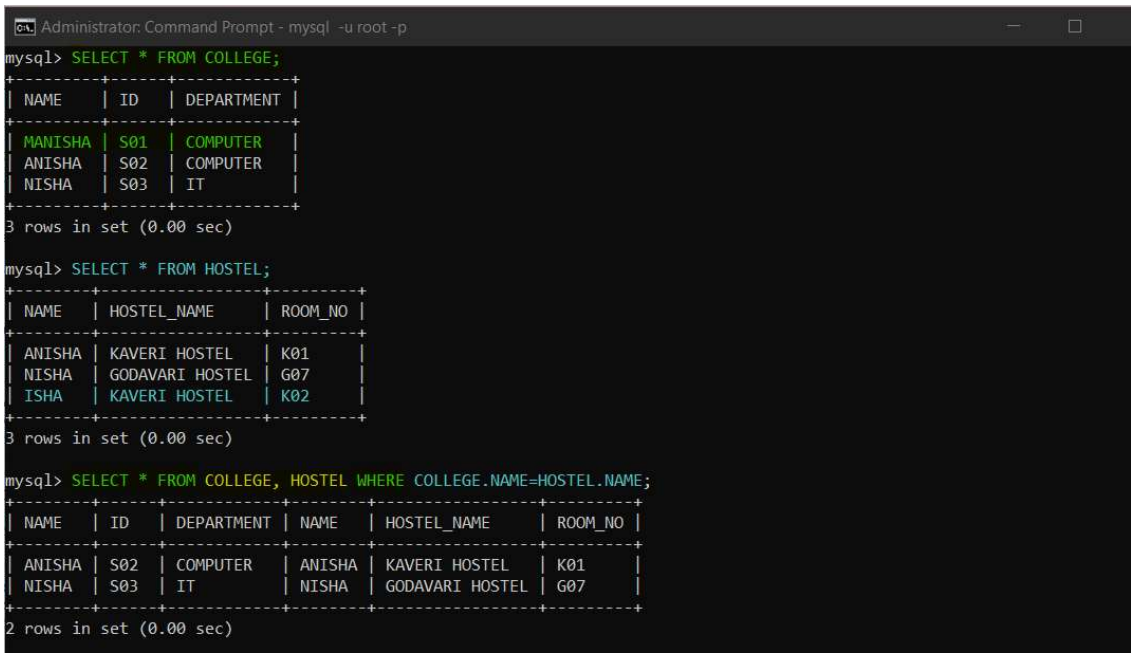
name	id	department
Manisha	S01	Computer
Anisha	S02	Computer
Nisha	S03	I.T.

Hostel:

name	hostel_name	room_no
Anisha	Kaveri Hostel	K01
Nisha	Godavari Hostel	G07
Isha	Kaveri Hostel	K02

Figure 8.8: Sample tables – College and Hostel

- Performing Inner Join on above tables yields following result.



- As You can see that while performing INNER JOIN on COLLEGE and HOSTEL , some information may lost in result.
  - In above INNER JOIN result information about "MANISHA" from COLLEGE table and "ISHA" from HOSTEL is lost.
- So OUTER JOIN can deal with such kind of information loss.

- OUTER JOIN can be classified into following categories.
  - **LEFT OUTER JOIN**
  - **RIGHT OUTER JOIN**
  - **FULL OUTER JOIN**

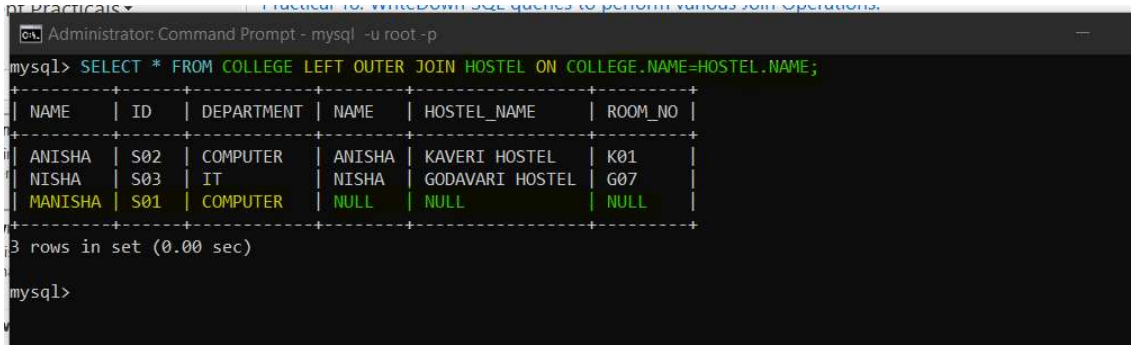
- **LEFT OUTER JOIN**

  - The LEFT OUTER JOIN retains all the records of the left table even though there is no matching record in the right table.
  - For such kind of records columns of Right table will be padded with NULL in result.

- **Syntax:**

  - **SELECT Column1,Column2 ,...,ColumnN FROM table1 LEFT OUTER JOIN table2 ON table1.column1 = table2.column2;**

- **Example:**



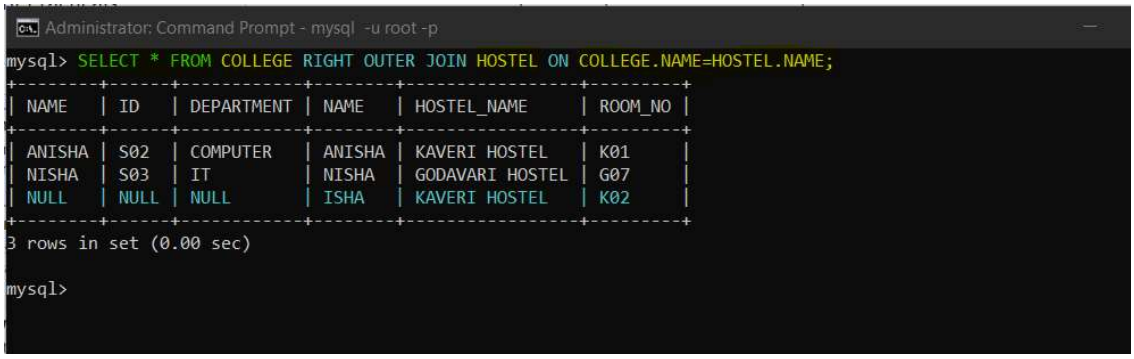
- **RIGHT OUTER JOIN**

  - The RIGHT OUTER JOIN retains all the records of the right table even though there is no matching record in the left table.
  - For such kind of records columns of left table will be padded with NULL in result.

- **Syntax:**

  - **SELECT Column1,Column2 ,...,ColumnN FROM table1 RIGHT OUTER JOIN table2 ON table1.column1 = table2.column2;**

- **Example:**



- **FULL OUTER JOIN**

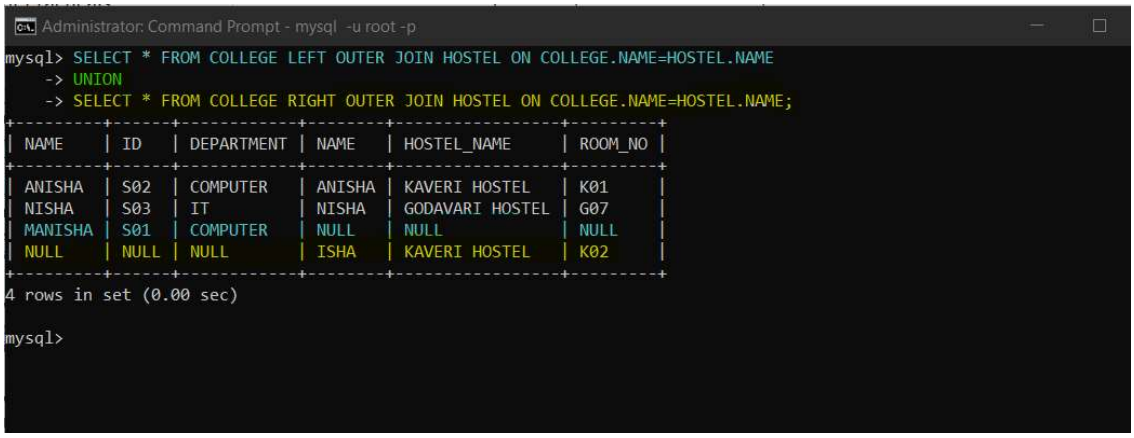
- FULL OUTER JOIN retains all the records of both the tables.
- it also paddes NULL values whenever required.

- **Syntax :**

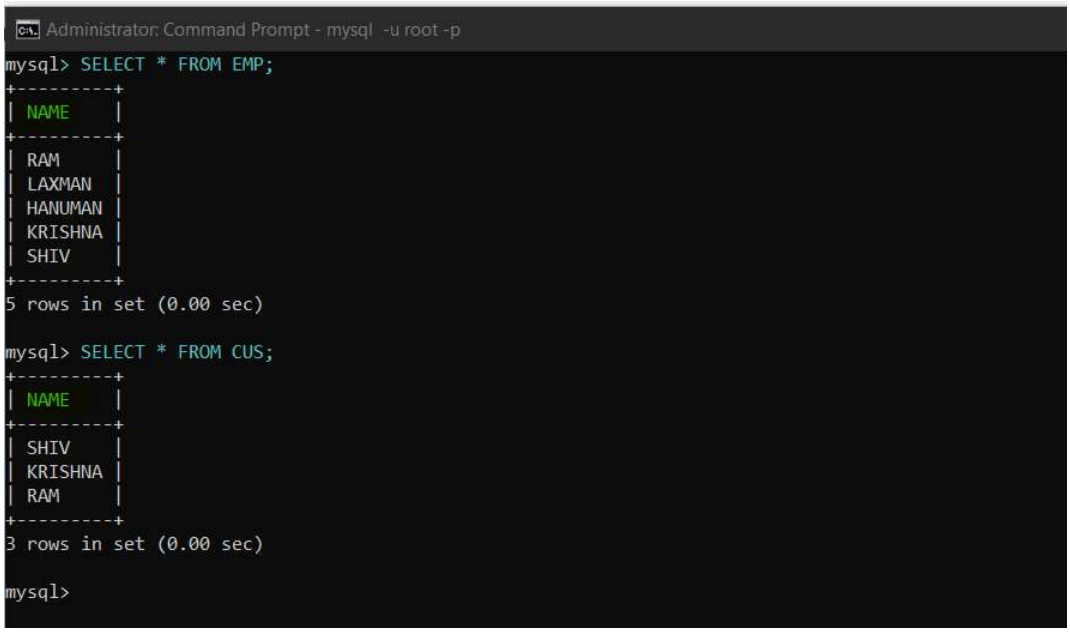
  - **SELECT Column1,Column2 ,...,ColumnN FROM table1 FULL OUTER JOIN table2 ON table1.column1 = table2.column2;**

- **Example : [HERE FULL OUTER JOIN DOES NOT WORK IN MYSQL SO WE CAN USE UNION TO PERFORM FULL OUTER JOIN]**

- **SELECT Column1,Column2 ,...,ColumnN FROM table1 LEFT OUTER JOIN table2 ON table1.column1 = table2.column2**
- **UNION**
- **SELECT Column1,Column2 ,...,ColumnN FROM table1 RIGHT OUTER JOIN table2 ON table1.column1 = table2.column2;**

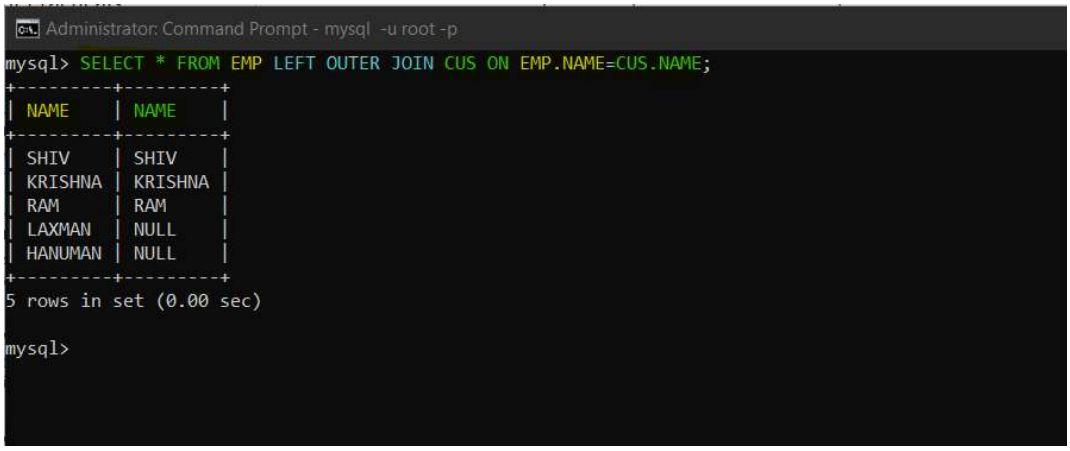


SET DIFFERENCE (MINUS OPERATION) USING LEFT OUTER JOIN



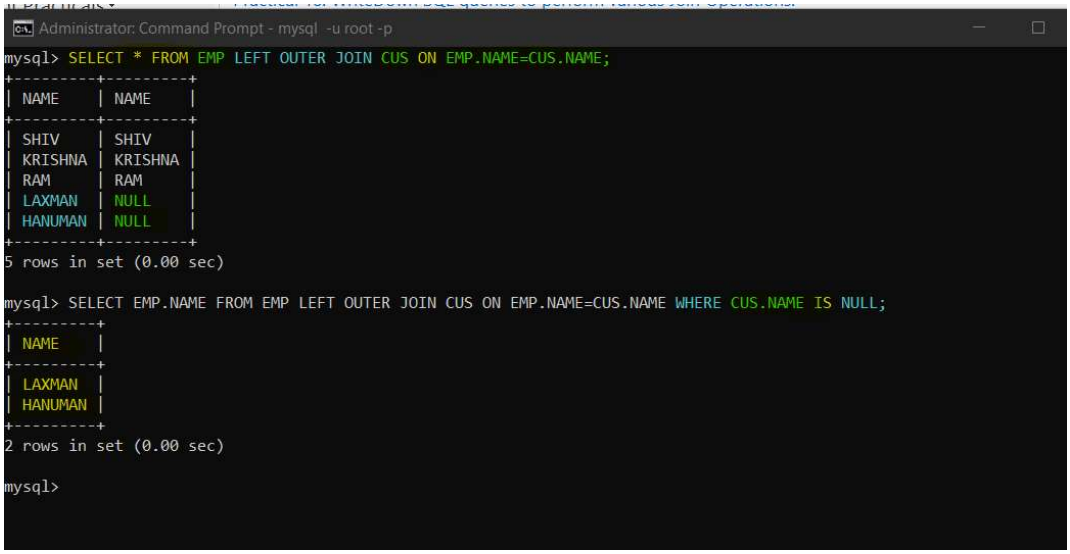
- Assuming above two tables
  - **EMP(NAME)**
  - **CUS(NAME)**
- Now ,We want to perform: **SELECT NAME FROM EMP MINUS SELECT NAME FROM CUS**
- WHOSE RESULT SHOULD BE : **{LAXMAN,HANUMAN}**

- SO FIRST, We will try to do LEFT OUTER JOIN of EMP table With CUS table.



  - Here , in above output NAME with
    - yellow highlight indicates EMP.NAME(Left table)
    - green highlight indicates CUS.NAME(right table)
    - FROM EMP table for (LAXMAN , HANUMAN) we are not able to find out corresponding from CUS , So WE GET NULL for CUS.NAME
    - Now , to get only records {LAXMAN,HANUMAN} we will add some condition with existing query ,which performs

similar to MINUS(SET DIFFERENCE Operation)



Practical 17 : WriteDown SQL Queries to perform various Set Operations.

Notebook:	Database Management Practicals		
Created:	4/23/2020 10:05 AM	**Updated:**	1/13/2023 12:22 PM
Author:	pagulearn.5334@gmail.com		

- A table in relational database is a subset of cartesian product among domains of its attributes.
- So, Various Set Operation can be performed on it
  - **UNION**
  - **INTERSECTION**
  - **DIFFERENCE**

- To Explain Set Operation We are considering Following Two Tables.

Customer:

cid	name
C01	Riya
C02	Jiya
C03	Diya
C04	Taral
C05	Saral

Employee:

eid	name	mngr_id
E01	Palak	E02
E02	Zalak	null
E03	Falak	E02
E04	Taral	E02
E05	Saral	E02

Figure 8.1: Customer and Employee table

- **UNION**

  - **Syntax**

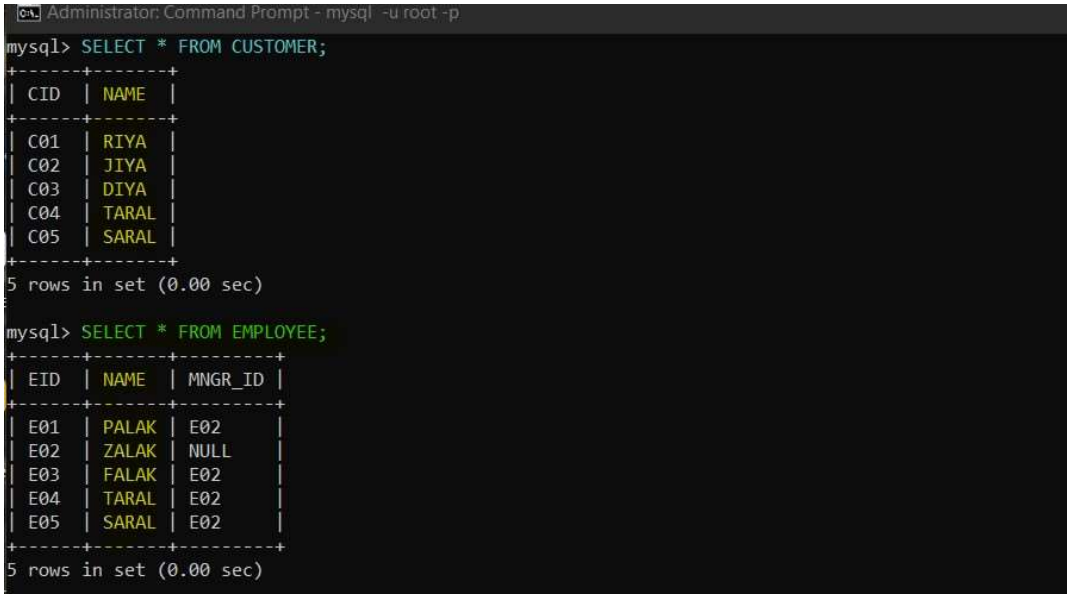
    - **Query1 UNION Query2**

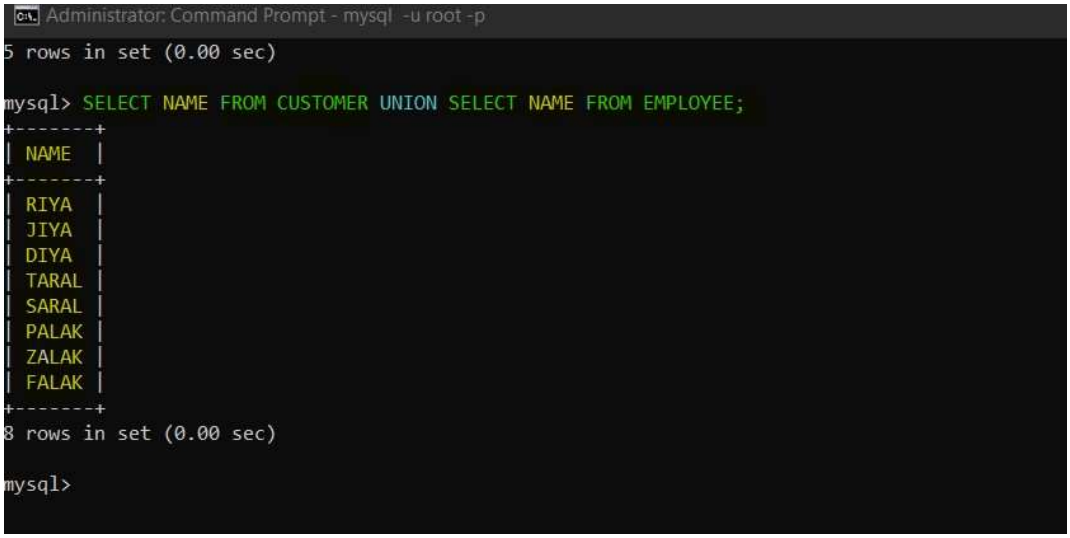
  - **Description**

    - UNION clause Merges the output of Query1 and Query2
    - The output will be as a single set of rows and columns.
    - Here, output will **contain all the records from Query1 and Query2** , but **common records will appear only once.**

- **The number of columns , name of each column and their data types in both the query need to be same.**
- **NULL values will not be ignored while comparing records in both query.**

- **Example :** List out name of persons who are either Customer or Employee.





Note :

- If we use **UNION ALL** then common records appears more than once in the resultant set.

- **INTERSECT**

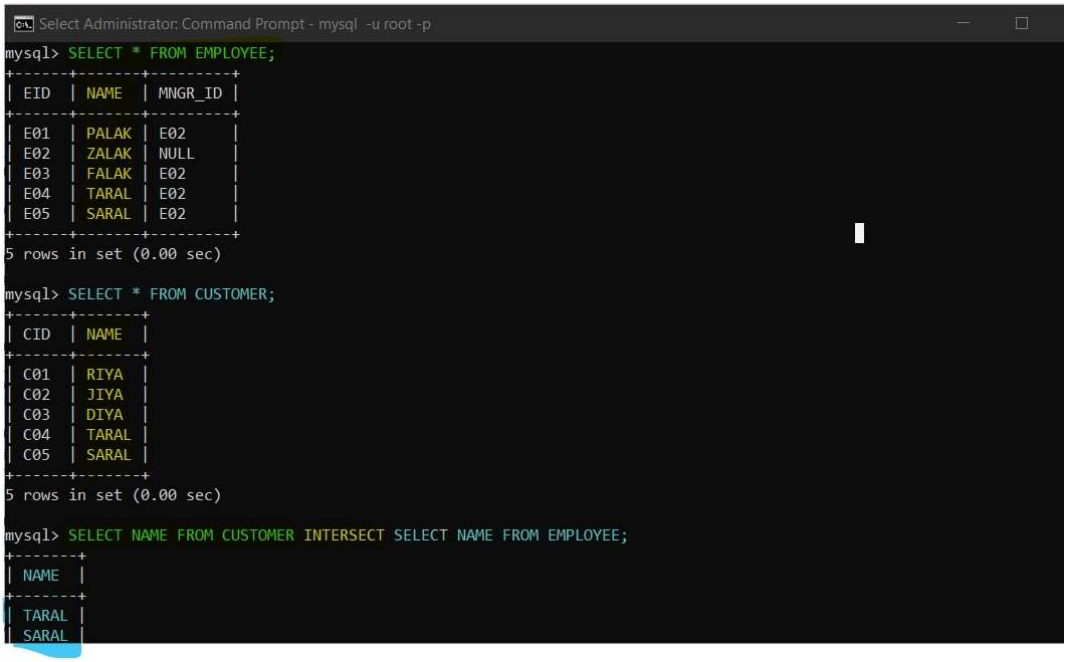
- **Syntax**

- **Query1 INTERSECT Query2**

- **Description**

  - The INTERSECT clause combines the output of Query1 and Query2
    - **The output will contains common records from both the Query1 and Query2.**
  - **The number of columns , name of each column and their data types in both the query need to be same.**
  - **The output will be same even if the orders of Query1 and Query2 are altered.**
  - **NULL values will not be ignored while comparing records in both query.**

- **Example** : List out name of persons who are Customer as well as Employee.



- **MINUS (DIFFERENCE)**

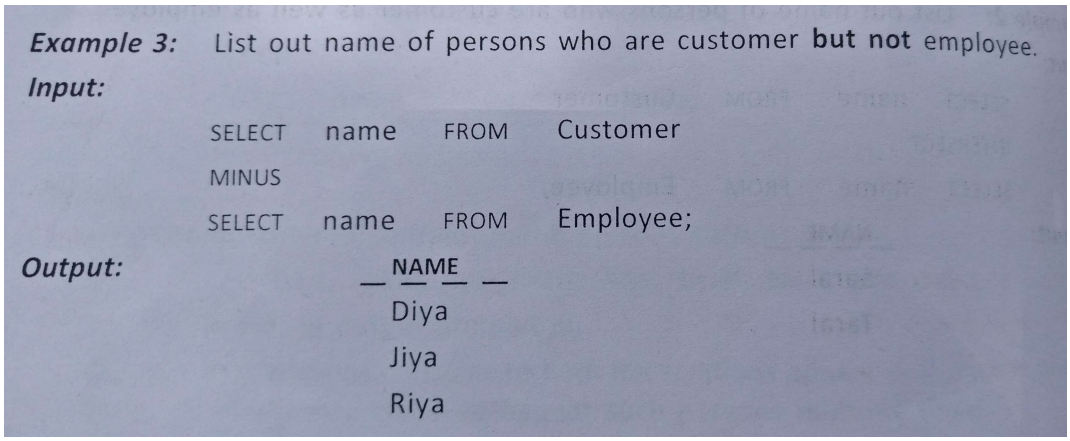
  - **Syntax**

    - **Query1 MINUS Query2**

  - **Description**

- The Minus clauses combines the output of Query1 and Query2 in such a way that , **output will contains those records from Query1 which are not present in Query2.**
- **The number of columns , name of each column and their data types in both the query need to be same.**
- **The output will not be same even if the orders of Query1 and Query2 are altered.**
- **NULL values will not be ignored while comparing records in both query.**

- **Example:** List out Name of Persons who are Customer but not Employee



Example 3: List out name of persons who are customer **but not** employee.

Input:

```
SELECT name FROM Customer
MINUS
SELECT name FROM Employee;
```

Output:

NAME
Diya
Jiya
Riya

- **Example :** List out Name of persons who are Employee but not Customer

- **SELECT NAME FROM EMPLOYEE MINUS SELECT NAME FROM CUSTOMER**

- Here , Result of above query will be **{PALAK,ZALAK,FALAK}**

Note :

- The Minus operation does not works in MySQL .
- It can be implemented using Left Join operation which we will see later.

Sub Queries

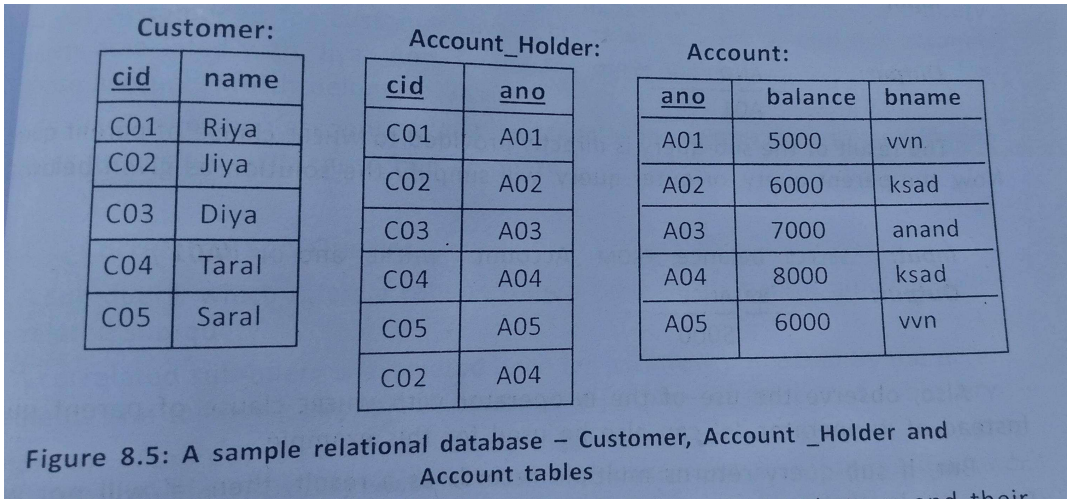
- **A sub query is an SQL statement which appears inside another SQL statement.**

- Syntax :

  - **SELECT ColumnName FROM TableName WHERE columnName OPERATOR (SELECT .....)**

  - Here , **SELECT statement appears as a part of WHERE clause of some other SQL statement is referred as a Sub Query.**
  - A statement containing a Sub Query is called a **Parent or Outer Query.**
  - **A result of Parent Query depends upon the result of the Sub Query.**
  - A Sub Query generally appears as a part of where clauses in various SQL statements.
  - A sub query can also be referred as a Nested Query.
  - There can be a chain of sub queries.

- To Explain The concept of Sub Queries we are going to use following tables.



Customer:

cid	name
C01	Riya
C02	Jiya
C03	Diya
C04	Taral
C05	Saral

Account_Holder:

cid	ano
C01	A01
C02	A02
C03	A03
C04	A04
C05	A05
C02	A04

Account:

ano	balance	bname
A01	5000	vvn
A02	6000	ksad
A03	7000	anand
A04	8000	ksad
A05	6000	vvn

Figure 8.5: A sample relational database – Customer, Account _Holder and Account tables

- Example : Find out balance of an account which belongs to Customer 'C01'

November **2016**

19
Saturday

CUSTOMER

Cid	name
C01	Riya
C02	Jiya
C03	Diya
C04	Taral
C05	Saral

Account_Holder

Cid	ano
C01	401
C02	402
C03	403
C04	404
C05	405
C02	404

Account :-

ano	Balance	bname
401	5000	vvn
402	6000	ksad
403	7000	anand
404	8000	ksad
405	6000	vvn

20
Sunday Query := Find out Balance of an Account which belongs to customer 'c01'.

A loving soul is a loved soul.

Solution :- 2016 November

21 Monday

1. Balance of an account is available in table : Account.

2. But He need to findout Account No of Customer Having Customer id : 'C01' - such type of information is available in table called : Account_Holder.

So, to solve our original query first we need to findout Account No of customer 'C01'.

→ For this task our query Hould look like,

① → SELECT ano FROM Account_Holden WHERE cid='C01';

The Result of above query Hould be '401'

NOW, He can use this Data '401' to findout Balance FROM Account:

② → SELECT Balance FROM Account HHERE ano='401';

Which Hill return final Result : 5000

November 2016

22
Tuesday

NOW; We can combine above two Process Write DOWN a sql query which uses the concept of sub query.

```
SELECT Balance FROM Account
    WHERE ano IN (SELECT ano FROM Account_Holder
    WHERE cid='c01');
```

Parent/outer query

Operator.

Sub query :- bcoz it is a part of other SQL statement's WHERE clause.

→ in the above query execution will be done in following manner.

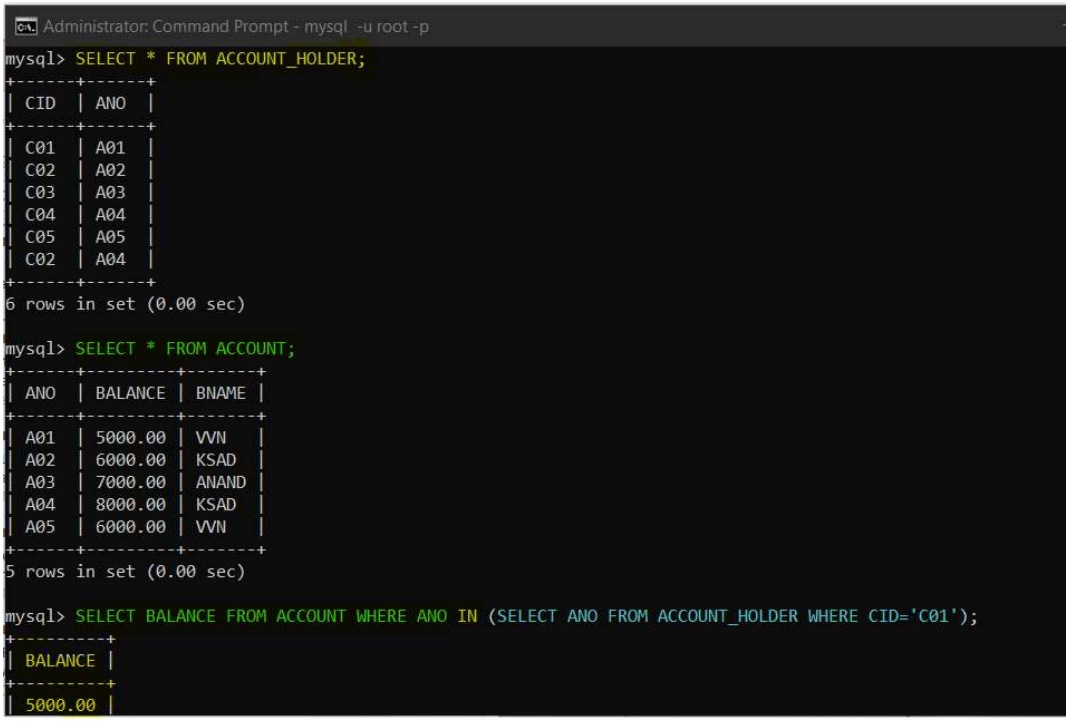
1. Sub query will return the result : '401'
   So, New query will look like this

```
SELECT Balance from Account
    WHERE ano IN ('401');
```

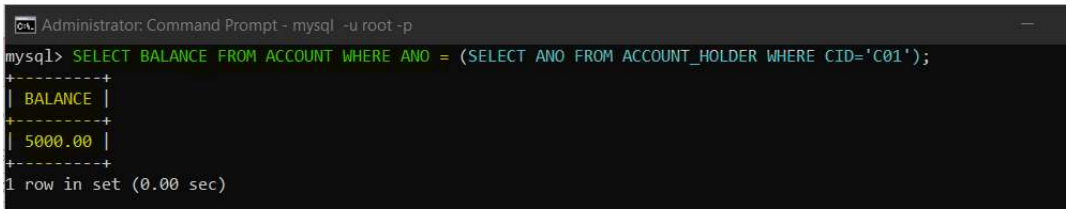
Result of sub query.

↳ this query will provide the final result : 5000

Procrastination will be overcome when you take little steps toward the goal. With each step you gain momentum, and with consistency, you'll be crossing the finish line before you know it.

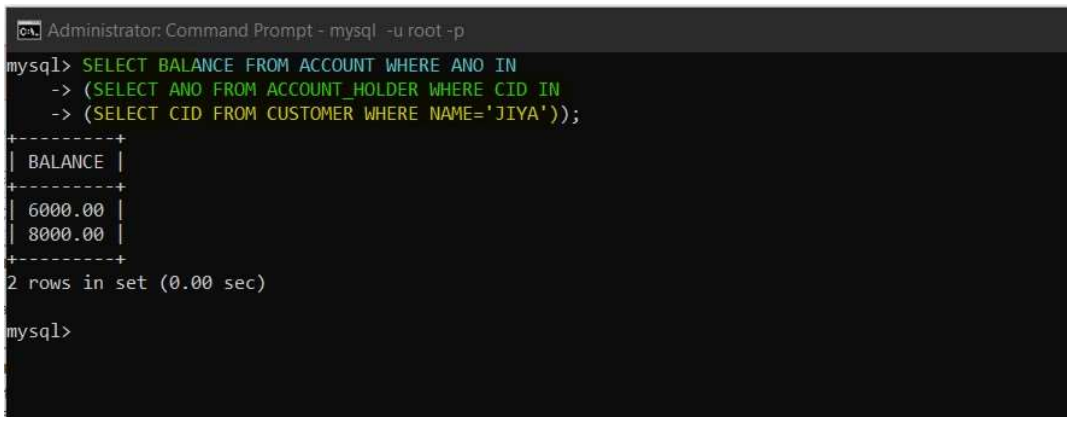


- In above query Blue highlighted SQL statement is Sub query whose result 'A01' is used by Outer Query highlighted with green color.
- Sub query is part of WHERE clause of Outer Query with Operator IN.
- Same Query can be written using = operator rather than IN operator also.



- **Example :** Find out the balance of accounts which belongs to Customer 'JIYA'

  - This query is an example of chain of sub queries.
    - Here first we need to find out Cid of customer 'JIYA' which is 'C02'
    - Then we need to find out Account No of customer having Cid 'C02' which is (A02,A04) because 'C02' has multiple Account.
    - Then we need to find out Balance of both Account no (A02,A04).
  - So above query can be written as follows.



- Here , SQL query highlighted with yellow is the inner most sub query which will return the result : 'C02'
- SQL query highlighted with green color use 'C02' as its WHERE clause input and produce the result ('A02','A04')
- SQL query highlighted with blue color use ('A02','A04') as its WHERE clause input and produce the output (6000,8000).

- This shows that Customer 'JIYA' having Customer ID 'C02' has Two Accounts 'A02','A04' with balance 6000,8000 respectively.

Correlated Sub Queries.

- **A Sub query which refers a column from a table in parent query is called correlated sub query.**
- **A correlated query is evaluated once for each record (row) processed by parent query statement.**

- **Example :** Find out the accounts having maximum balance in branch to which it belongs. Display account no , balance , bname for such accounts.

ACCOUNT :-

ANO	BALANCE	BNAME
401	5000	VVN
402	6000	KSAD
403	7000	ANAND
404	8000	KSAD
405	6000	VVN.

Query :- FIND OUT the accounts having max balance in branch to which it belongs. Display account no, balance, bname for such account.

SQL statement :-

```
SELECT ano, balance, bname FROM ACCOUNT ACC
WHERE balance IN (
        SELECT MAX(balance) FROM
ACCOUNT WHERE bname = ACC.bnam
```

- Here; sub query is refering column of Parent query Through the statement

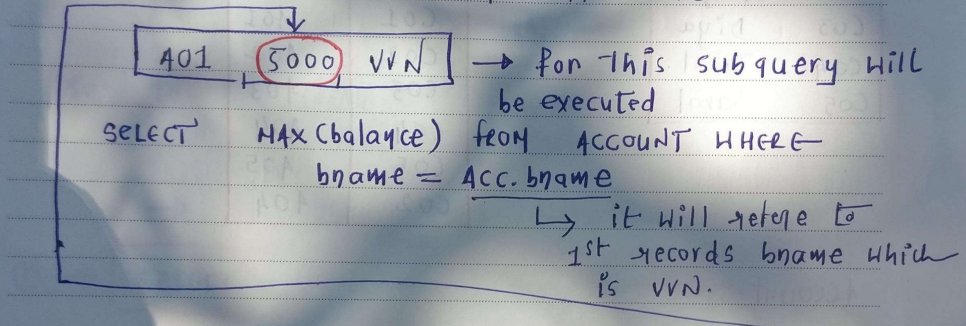
bname = ACC.bname → if this sub query is called correlated.

Some people own a lot of things, but many people's things own them.

Execution of this query will be done in following manner.

→ for each record in parent query table sub query will be executed.

1. so; first record in account table is



→ so; sub query becomes;

SELECT MAX(balance) from account WHERE bname = 'VVN'.

- Result of it will be (6000)

→ so; NON PARENT query will become;

SELECT ano, balance, bname FROM ACCOUNT ACC WHERE balance IN (6000).

→ Remember this parent query is executed for first Record having balance = 5000.

→ So, Result of Parent query execution is false.
- it will return nothing.

2. for second record in Parent query table

(402, 6000, KSAD) → sub query will be executed → where bname will refer to KSAD

⇒ SELECT MAX(balance) FROM account WHERE bname = 'KSAD'

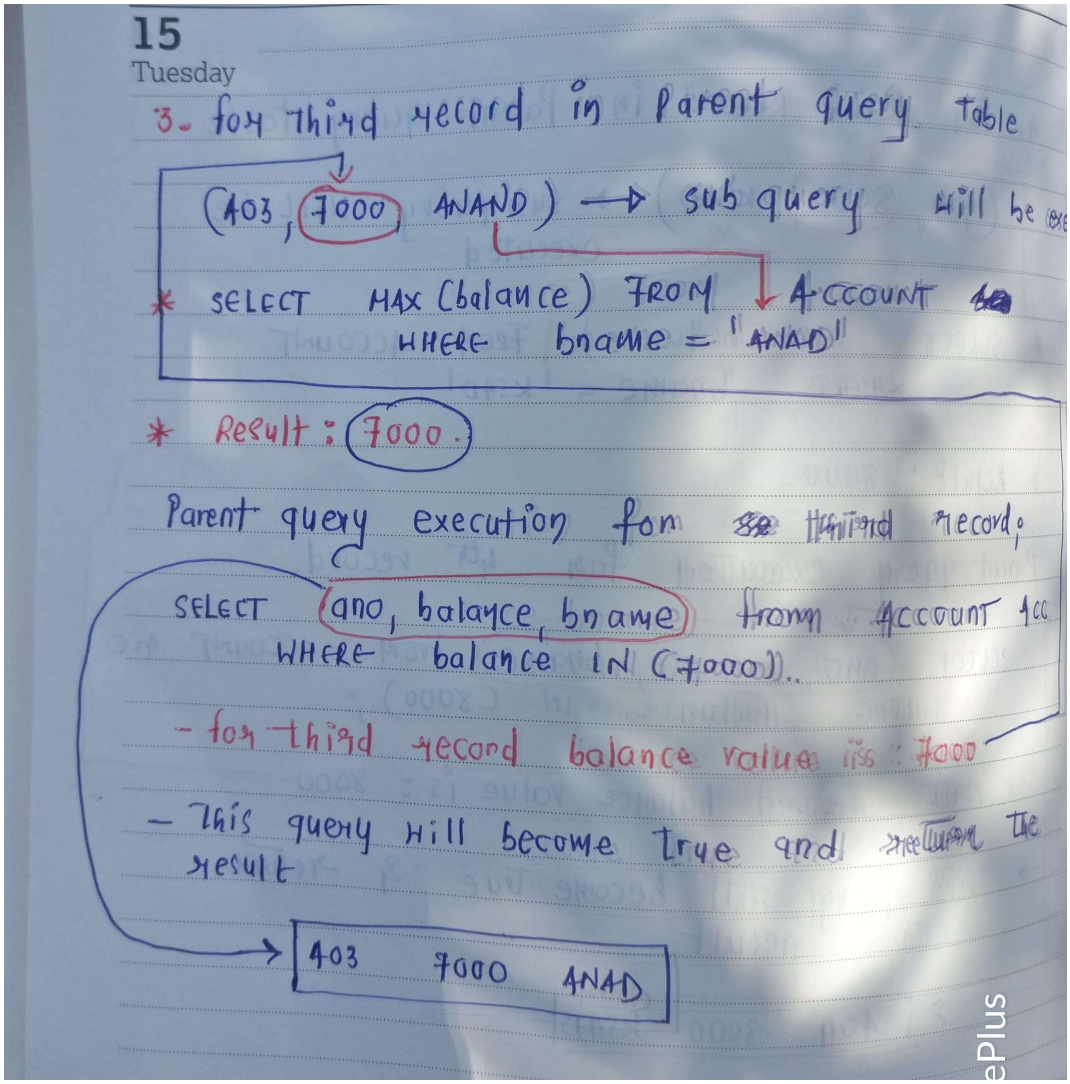
⇒ Result: 8000.

Parent query executed for second record;

SELECT ano, balance, bname FROM account acc WHERE balance IN (8000).

→ But for second record : Balance value is : 6000

→ This does not match with 8000 so, result of the query will fail.


15
Tuesday
3. for third record in Parent query table
(403, 7000 ANAND) → sub query will be exe
* SELECT MAX (balance) FROM Account
WHERE bname = 'ANAD'
* Result : 7000.
Parent query execution for third record.
SELECT (ano, balance, bname) from Account
WHERE balance IN (7000)).
- for third record balance value is : 7000
- This query will become true and
result
403 7000 ANAD

4. for forth record in Parent query table

(404, 8000, KSAD) → subquery will be executed

* SELECT MAX(balance) FROM ACCOUNT WHERE bname = 'KSAD'

* Result : 8000.

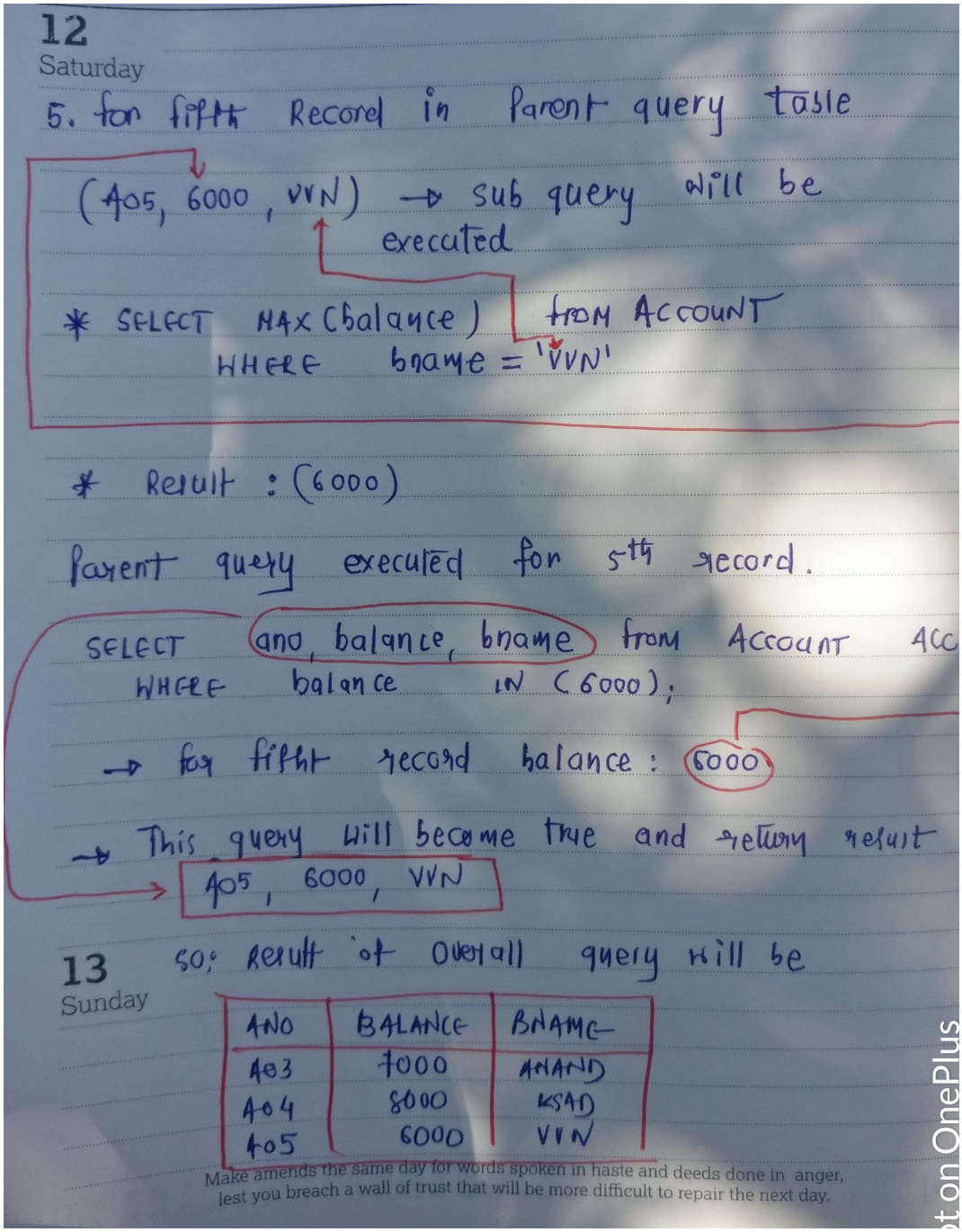
Parent query executed for 4th record.

SELECT ano, balance, bname from ACCOUNT ACC WHERE Balance IN (8000);

→ forth Record balance value is : 8000

→ This query will become true & return the result

→ | 404 8000 KSAD |




```
Administrator: Command Prompt - mysql  -u root -p
mysql> SELECT ANO,BALANCE,BNAME FROM ACCOUNT ACC WHERE BALANCE IN
    -> (SELECT MAX(BALANCE) FROM ACCOUNT WHERE BNAME=ACC.BNAME);
+------+---------+-------+
| ANO  | BALANCE | BNAME |
+------+---------+-------+
| A03  | 7000.00 | ANAND |
| A04  | 8000.00 | KSAD  |
| A05  | 6000.00 | VVN   |
+------+---------+-------+
3 rows in set (0.01 sec)

mysql>
```

- **Here , ACC is the alternative name Given to ACCOUNT table to use it in sub query**

Practical 12: Write down SQL queries that demonstrate the use of Aggregrate Function.

Notebook:	Database Management Practicals		
Created:	4/16/2020 10:18 AM	Updated:	12/13/2022 11:22 AM
Author:	pagulearn.5334@gmail.com		

SQL Functions

- SQL functions **accepts arguments as an input** and return **result as an output.**
- Arguments can be some **constant values , expressions or column names** related to **tables.**
- The general form of SQL functions is:
  - **Function_name(argument1,argument2,.....,argumentN)**

- Sql Functions can be categories into two parts
  - **Aggregate function**
  - **Scalar function**
- **based on whether they operate on a set of rows or single row.**

- **Aggregate Function (Group Functions)**

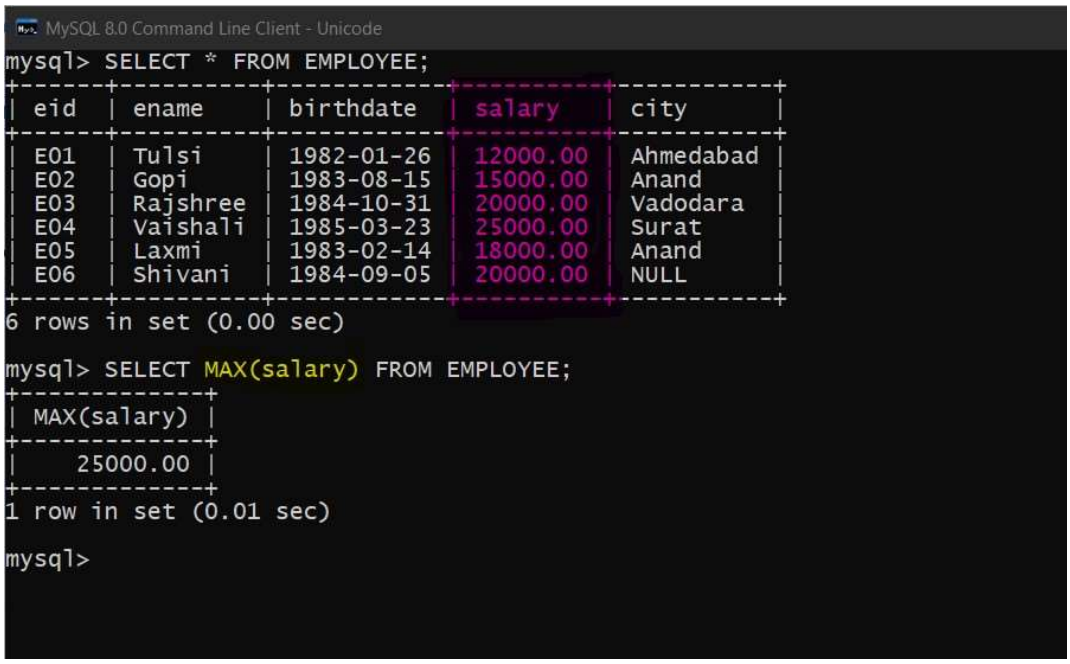
  - A function that operate **on a set of rows (or values)** are called **aggregate or group function.**
  - These function accept a **set of rows (a group) as an input.**
  - They return **only a single row as an output.**
  - So, if 5 rows are given as an input , there will be only single row as a result.

  - **Example :**

  - **MAX**

    - **MAX (columnName)**
    - **it will returns maximum value for a given column**

  - **Find out Maximum Salary from Employee Table**

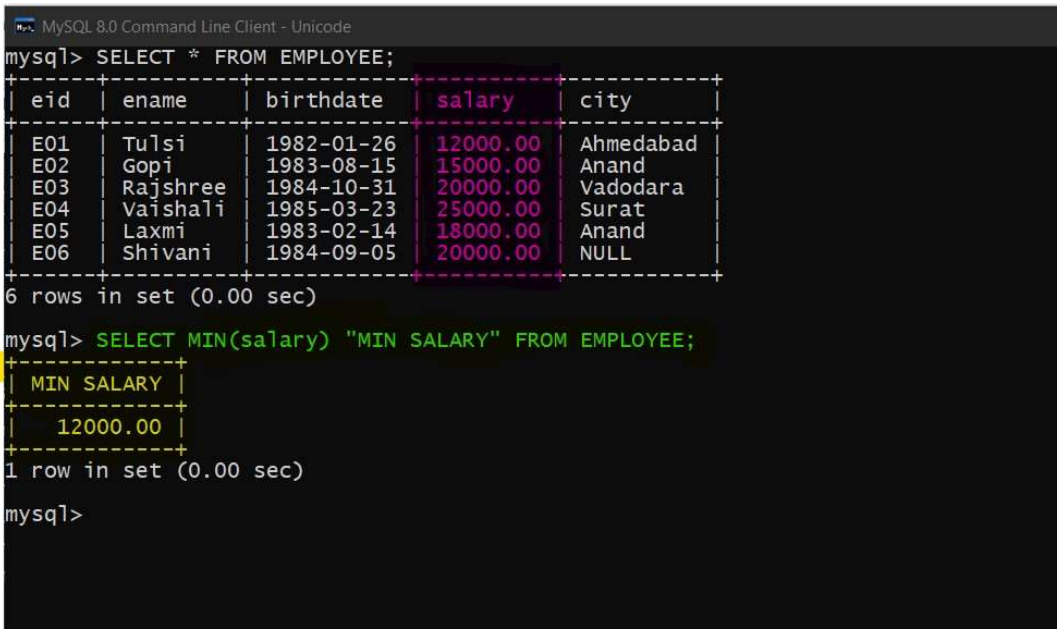


- Here MAX() function takes **set of values of salary column** take as an input and return **the Maximum salary among them.**

- **MIN**

  - **MIN(columnName)**
  - it returns minimum value for a given column.

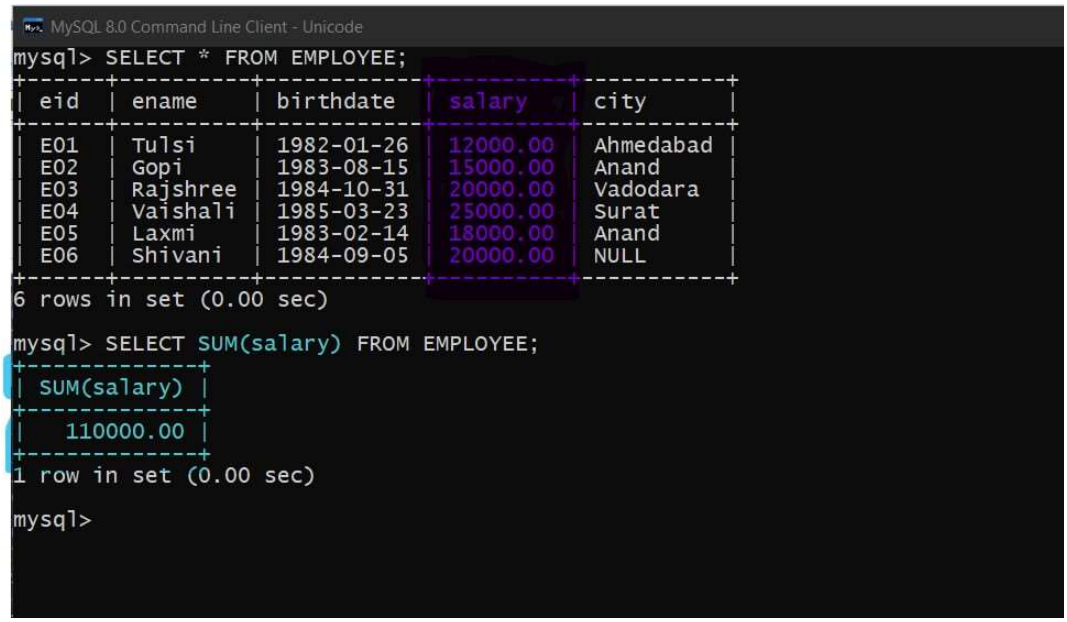
- **Find out minimum salary from the Employee and assign new name "MIN SALARY" resultant column.**



- **SUM**

- **SUM([Distinct | All] columnName)**
- **Returns sum of all values for a given column**
- if "Distinct" keyword is provided then duplicate values are considered only once.
- By default "All" is considered.

- Find Sum of all the salaries of Employee in Employee table.



- SUM function takes set of values of Salary column and perform sum if all the salaries and return sum of salary as a result.

- **Find Sum of the distinct(unique) salaries of Employee in Employee table.**

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT * FROM EMPLOYEE;
+------+----------+------------+----------+-----------+
| eid  | ename    | birthdate  | salary   | city      |
+------+----------+------------+----------+-----------+
| E01  | Tulsi    | 1982-01-26 | 12000.00 | Ahmedabad |
| E02  | Gopi     | 1983-08-15 | 15000.00 | Anand     |
| E03  | Rajshree | 1984-10-31 | 20000.00 | Vadodara  |
| E04  | Vaishali | 1985-03-23 | 25000.00 | Surat     |
| E05  | Laxmi    | 1983-02-14 | 18000.00 | Anand     |
| E06  | Shivani  | 1984-09-05 | 20000.00 | NULL      |
+------+----------+------------+----------+-----------+
6 rows in set (0.00 sec)

mysql> SELECT SUM(distinct salary) FROM EMPLOYEE;
+----------------------+
| SUM(distinct salary) |
+----------------------+
|             90000.00 |
+----------------------+
1 row in set (0.00 sec)

mysql>
```

- **AVG**

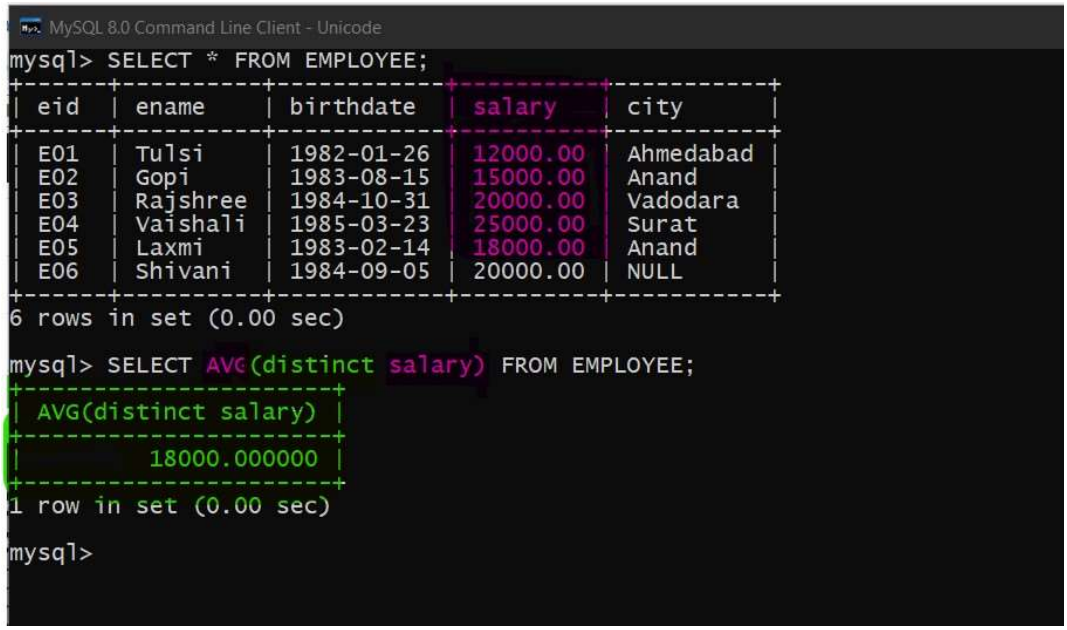
  - **AVG ([Distinct | All] ColumnName)**

  - **It will return average of all values for a given column**

- **Find average of all salaries from Employee table.**

```
MySQL 8.0 Command Line Client - Unicode
mysql> SELECT * FROM EMPLOYEE;
+------+----------+------------+----------+-----------+
| eid  | ename    | birthdate  | salary   | city      |
+------+----------+------------+----------+-----------+
| E01  | Tulsi    | 1982-01-26 | 12000.00 | Ahmedabad |
| E02  | Gopi     | 1983-08-15 | 15000.00 | Anand     |
| E03  | Rajshree | 1984-10-31 | 20000.00 | Vadodara  |
| E04  | Vaishali | 1985-03-23 | 25000.00 | Surat     |
| E05  | Laxmi    | 1983-02-14 | 18000.00 | Anand     |
| E06  | Shivani  | 1984-09-05 | 20000.00 | NULL      |
+------+----------+------------+----------+-----------+
6 rows in set (0.00 sec)

mysql> SELECT AVG(salary) FROM EMPLOYEE;
+--------------+
| AVG(salary)  |
+--------------+
| 18333.333333 |
+--------------+
1 row in set (0.00 sec)

mysql>
```

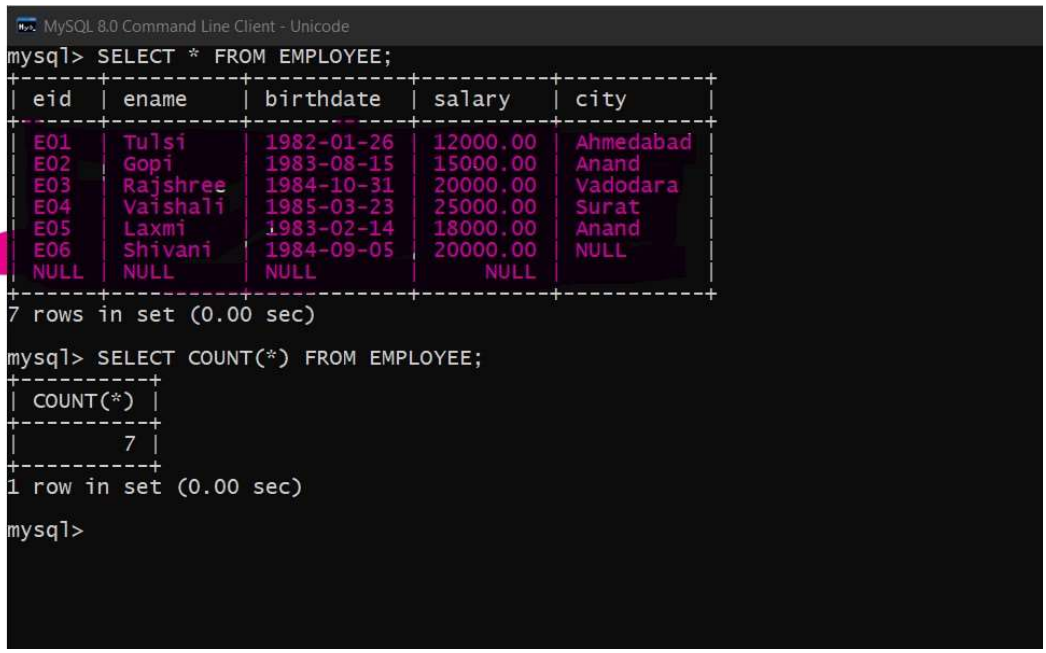
- **Find average of distinct salaries from Employee table.**



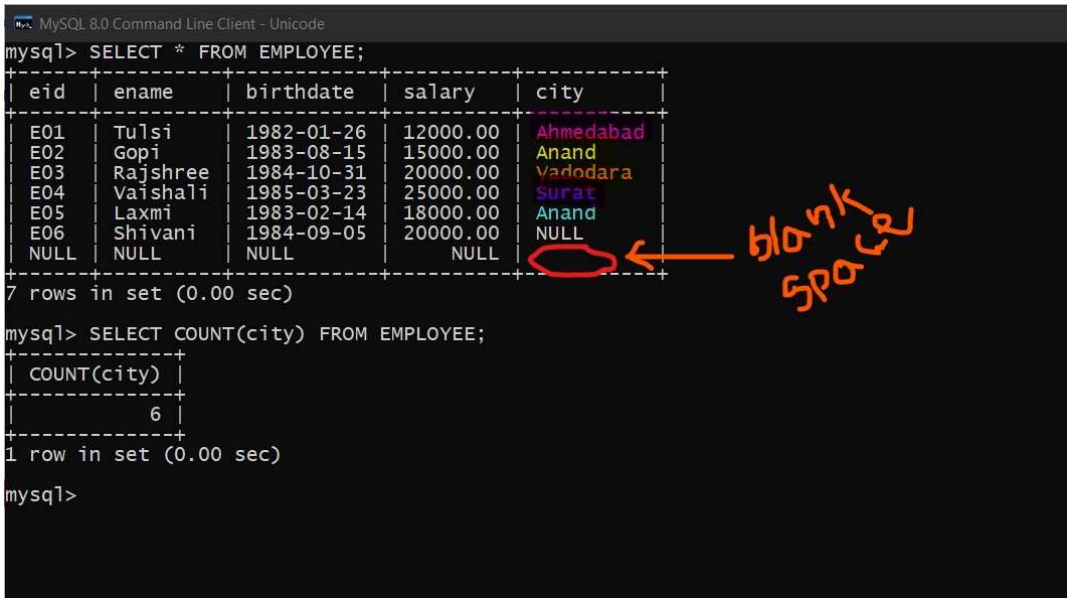
- **COUNT**

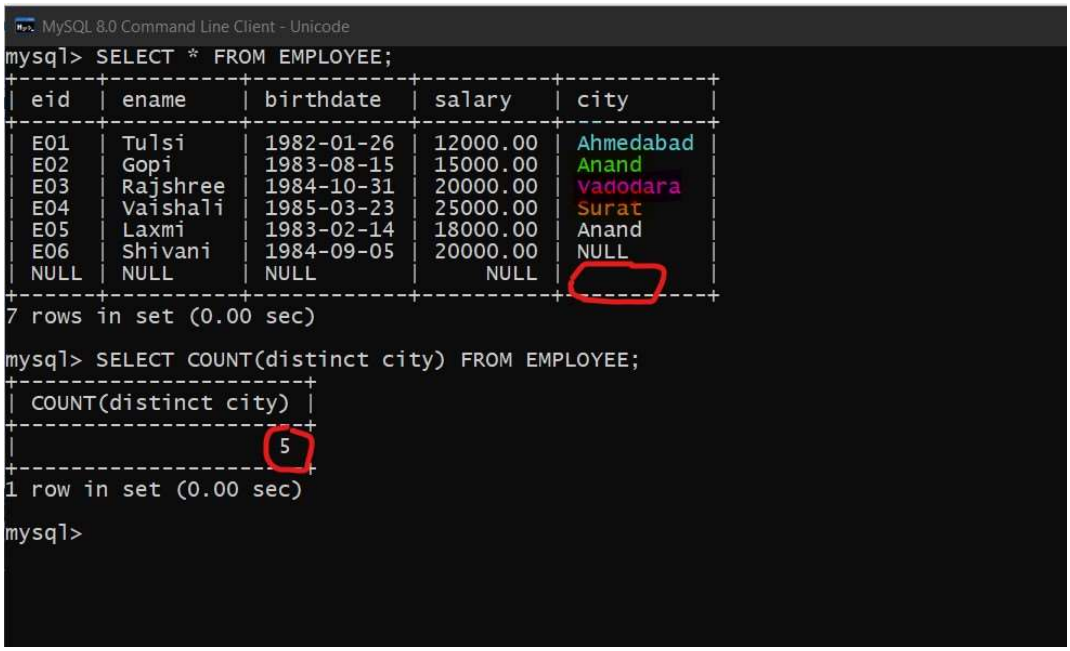
  - **COUNT(*)**
    - It will return **no of rows** in a tables including **duplicate and having null values.**

- Find out total no of Employees.



- COUNT([Distinct / All] columnName)
  - It returns **number of rows where column does not contain null value**
  - Remember **it will count black space value if column contain blank space**.





Write down Sql Queries Following Tables.

DEPOSIT

ACTNO	CNAME	BNAME	AMOUNT	ADATE
100	ANIL	VRCE	1000.00	1-MAR-95
101	SUNIL	AJNI	5000.00	4-JAN-96
102	MEHUL	KAROLBAGH	3500.00	17-NOV-95
104	MADHURI	CHANDI	1200.00	17-DEC-95
105	PRMOD	M.G.ROAD	3000.00	27-MAR-96
106	SANDIP	ANDHERI	2000.00	31-MAR-96
107	SHIVANI	VIRAR	1000.00	5-SEP-95
108	KRANTI	NEHRU PLACE	5000.00	2-JUL-95
109	MINU	POWAI	7000.00	10-AUG-95

VRCE	NAGPUR
AJNI	NAGPUR
KAROLBAGH	DELHI
CHANDI	DELHI
DHARAMPETH	NAGPUR
M.G.ROAD	BANGLORE
ANDHERI	BOMBAY
VIRAR	BOMBAY
NEHRU PLACE	DELHI
POWAI	BOMBAY

CUSTOMERS

ANIL	CALCUTTA
SUNIL	DELHI
MEHUL	BARODA
MANDAR	PATNA
MADHURI	NAGPUR
PRAMOD	NAGPUR
SANDIP	SURAT
SHIVANI	BOMBAY
KRANTI	BOMBAY
NAREN	BOMBAY

BORROW

LOANNO	CNAME	BNAME	AMOUNT
201	ANIL	VRCE	1000.00
206	MEHUL	AJNI	5000.00
311	SUNIL	DHARAMPETH	3000.00
321	MADHURI	ANDHERI	2000.00
375	PRMOD	VIRAR	8000.00
481	KRANTI	NEHRU PLACE	3000.00

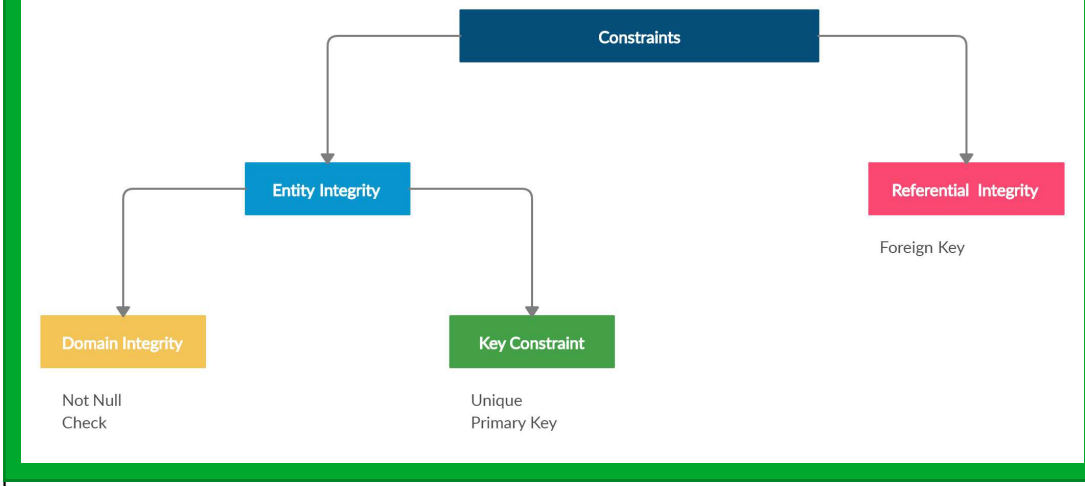
1. **List total deposit amount from deposit. [use SUM()]**
2. **Count total no of customers from Customer.**
3. **Find out Minimum loan amount from Borrow.**
4. **Find out count of distinct cities where different branch is located.**
5. **Find out average of amount of loan taken from branches.**
6. **Find out maximum amount deposited by a depositor.**

CHAPTER 9 : Introduction To Constraints

Notebook:	CHAPTER 9 : SQL CONSTRAINTS		
Created:	4/1/2020 9:10 AM	**Updated:**	10/11/2023 3:40 PM
Author:	pagulearn.5334@gmail.com		
URL:	https://inc-onenote.officeapps.live.com/o/onenoteframe.aspx?ui=en%2DUS&rs=en%2...		

Introduction

- A Constraint is a rule that restricts the data values that may be present in database.
- The Data stored in data should be valid , correct and consistent .
- So , When we do data manipulation in database ,these constraints should be followed..
- Oracle , provides various types of constraints to ensure validity, correctness and consistency of data in database.
- As Oracle is based on the Relational Data Model , it provides following general types of constraints.



- The Oracle constraints can be broadly classified into following categories..
  - **Entity Integrity Constraint**
  - **Referential Integrity Constraint**

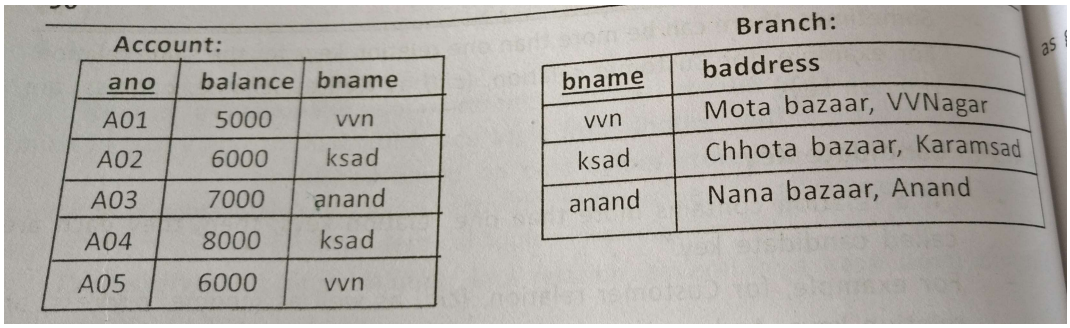
Entity Integrity Constraint

- Entity Integrity Constraints are those constraints which **restricts the data values in a row of an individual table.**
- This can be done in two ways
  - **Domain Integrity Constraints**

        - It specifies that the **value of each column must be belong** to the **domain** of that column
        - each value in column must follow some rules.
            - Balance for any account in Banking should not be negative value.
            - Account no of a customer can not be null value.
        - To provide Domain Integrity Oracle offers two constraint :
            - **NOT NULL**
            - **CHECK**
    - **Key Constraint**
        - It Ensure that there must be some way through which we can **distinguish two different rows** of the **same table uniquely.**
        - Key Constraint is achieved by Constraints
            - **UNIQUE**
            - **PRIMARY KEY**

Referential Integrity Constraint

- Referential Integrity constraint specifies that the **row in one table which refers** to the **other row** must **refer to the existing row in that table.**
- Referential Integrity constraint can be achieved by Constraint.
    - **FOREIGN KEY**



Account:

ano	balance	bname
A01	5000	vvn
A02	6000	ksad
A03	7000	anand
A04	8000	ksad
A05	6000	vvn

Branch:

bname	baddress
vvn	Mota bazaar, VVNagar
ksad	Chhota bazaar, Karamsad
anand	Nana bazaar, Anand

Example:

- Take an example of above to table Account and Branch
- Let say here , first row in Account table with bname=vvn refer to the row ={ "vvn", "Motabazar" } of the branch table. from the branch table we say that such kind of branch is exist.
- But if we have record like { A06 , 6000, surat} in Account table then it violates the referential integrity rule because such kind of branch doesnot exist in Branch table.

- In Oracle , Constraint can be implemented using two ways
    - Using CREATE TABLE statement while creating a table

- Using ALTER TABLE statement to add or modify constraint in already existing table.

- Once constraint has been implemented , the DBMS itself applies that constraints while executing various SQL statements.
- If any SQL statement tries to violate those constraint , oracle cancel those operation.

- Constraints are associated with specific table and stored in data dictionary as a part of structure or schema of a table.

How to Define Constraint In Oracle?

Column Level Constraint

- Constraints can be defined along with **Column definition**
- **Syntax**
  - **columnname datatype(size) constraint-definition**
- Column level **constraint can not be applicable** when constraint **span across multiple columns** in table.
  - like defining PRIMARY KEY which is a combination more than one column.

Table Level Constraint

- In this type constraints are defined after defining all the columns of the table.
- **Syntax**
  - **CREATE TABLE tablename (column1 datatype (size) , ...,columnN datatype(size) ,Constraint definition1 , ..., Constraint definitionN);**
- Table level Constraint can be applicable when constraint span across multiple columns of a tables.

Now ,We will see each constraint in details for that following table schema should be taken into consideration.

An Account Table

Column name	Data type	Size
ano	Char	3
balance	Number	9
bname	Varchar2	10

A Branch Table

Column name	Data type	Size
bname	Varchar2	10
baddress	Varchar2	20

Figure 9.2: Structure of an Account and a Branch table

NOT NULL Constraint

- There may a situation when , records in the table do not contain any value for some fields.
- In Oracle , NULL can be stored in such fields.

- A NULL value in table indicates "not applicable" , "missing" , "not known"
- A NULL value is distinct from Zero or other numeric value for numerical data
- A NULL value is distinct from a blank space for character data
- A NULL value will be evaluated to NULL in any expression.
- The result of any condition including NULL value is unknown and treated as FALSE.

- But sometime it is required that a field cannot be left empty in table , which means column must have some value other than NULL.
- Example :
  - Balance column in Account can not be NULL for any account.
  - Or Account No cannot be NULL value.
- To ensure such type of restriction on field of tables a NOT NULL constraint can be used.

Enforced Restriction :

- **A column defined as a NOT NULL , cannot have NULL values.**
- **Such columns becomes a mandatory and can not left empty for any record.**

NOT NULL constraint defined at Column Level

- **Syntax :**
  - **columnName datatype (size) NOT NULL**

- Example :
  - I am applying NOT NULL constraint during CREATING a Table..

```
mysql> CREATE TABLE ACCOUNT(ANO VARCHAR(5),BALANCE NUMERIC(8,2) NOT NULL ,bname VARCHAR(5));
Query OK, 0 rows affected (0.12 sec)

mysql> DESC ACCOUNT;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| ANO     | varchar(5)   | YES  |     | NULL    |       |
| BALANCE | decimal(8,2) | NO   |     | NULL    |       |
| bname   | varchar(5)   | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
3 rows in set (0.04 sec)

mysql>
```

- Now if you try to insert a NULL value in balance column You will get following Type of Error..

```
mysql> INSERT INTO ACCOUNT VALUES('A01',NULL,'VVN');
ERROR 1048 (23000): Column 'BALANCE' cannot be null
mysql>
```

Note :

- **NOT NULL constraint cannot be applied at Table level.**

CHAPTER 9 : CHECK CONSTRAINTS

Notebook:	CHAPTER 9 : SQL CONSTRAINTS		
Created:	4/6/2020 9:50 AM	**Updated:**	10/10/2023 10:12 PM
Author:	pagulearn.5334@gmail.com		

CHECK Constraint

- The Check Constraint is used to implement **business rules.**
- This Constraint is also referred as **business rule constraint.**
- Example: Some business rules specific to a bank system are
  - A balance in any account should not be a negative value
  - A bname must be 'vvn','ksad','anand'
  - An account no must start with 'A'.
- **This Business rule defines a domain for a particular column** , : And Domain is set of permitted values for particular column.

Enforced Restriction:

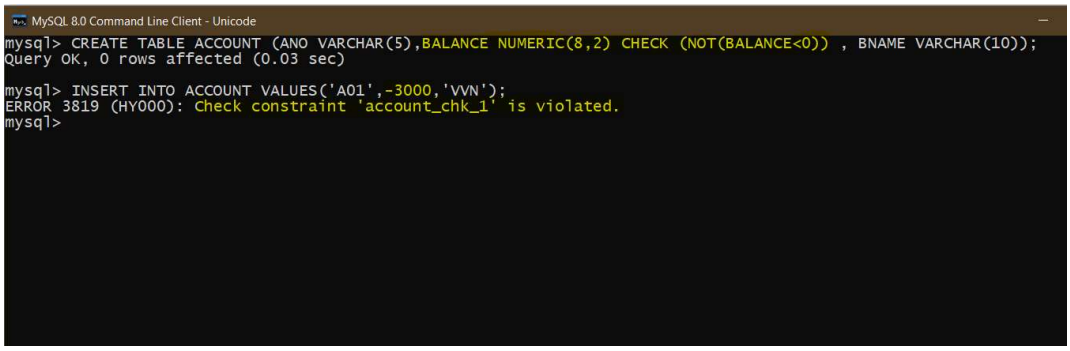
- Check constraint is **bound to a particular column.**
- Once Check constraint is implemented , any insert or update operation on that table must follows this constraints.
- If any operation violates conditions, it will be rejected.

A CHECK Constraint defined at Column Level

Syntax :

- **ColumnName datatype (size) CHECK (condition)**

Example:

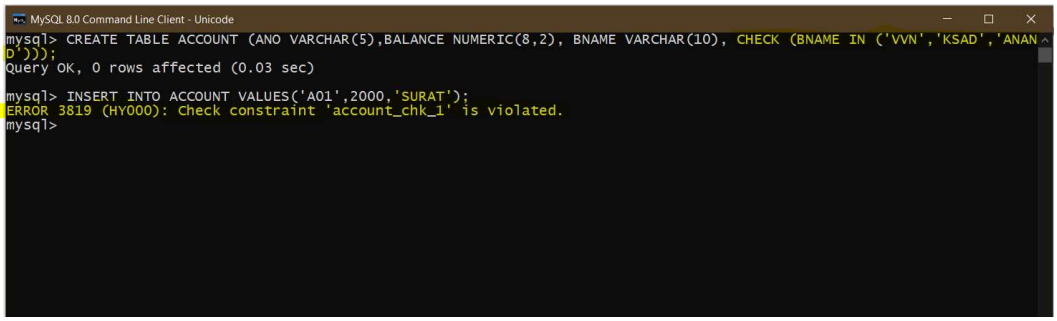


A CHECK Constraint defined at Table Level

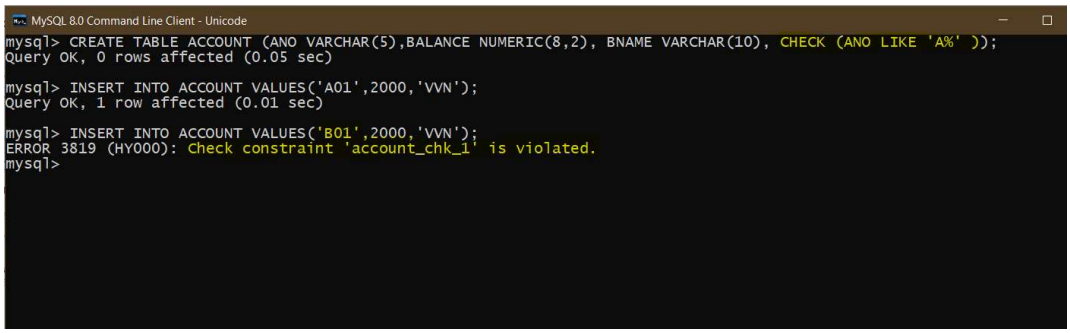
Syntax:

- **CHECK (condition)**

Example : Bname must be 'VVN' , 'KSAD', 'ANAND'



Example 2 : Account no must start with 'A'



Features:

- Any Business rule validation can be applied using this constraint.
- A CHECK constraint **takes longer time to execute** . so it should be avoided if possible.
- If condition encounters **NULL values** the result will be unknown. it means **this constraint will be ignored while encountering NULL values.**
- The condition specified with CHECK constraint must be a valid logical expression.
- If Constraint is violated , it displays the error message.

Naming a Constraint

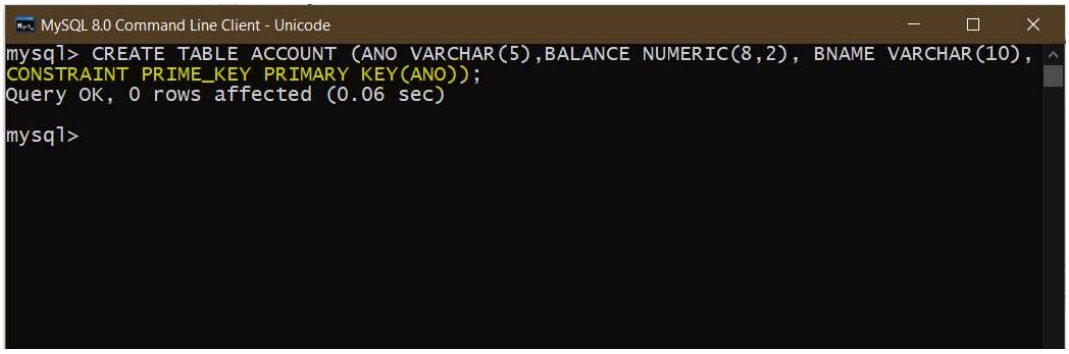
- It is possible to provide **some user defined name to a constraint** we have created so far.
- It will be **useful to modify or drop** this constraint later on.

Syntax :

- **CONSTRAINT constraint_name constraint_definition**

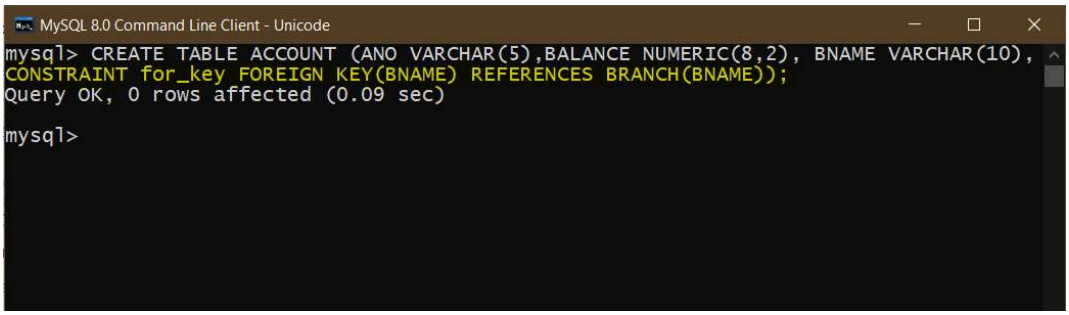
Example : Create Account table with ANO as PRIMARY KEY and assign

name 'Prime_key' to this constraint.



Example : Create a table Account having constraint name "for_key" defining BNAME as foreign key , referring to BRANCH table.

Note: for below query i have already created Branch table with BNAME as primary key first.oo



Altering Table schema

- Add , modify or drop constraints can be considered as a part of altering table schema.
- ALTER command can be used to these tasks.

ADD Constraint:

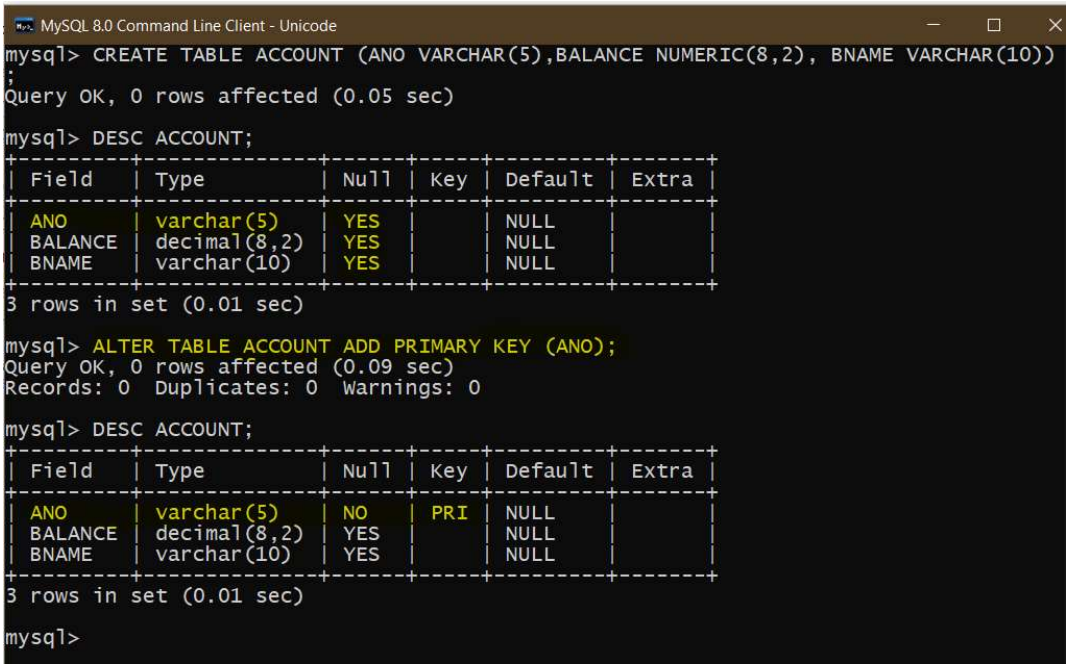
Syntax:

- **ALTER TABLE tableName ADD constraint-definition;**

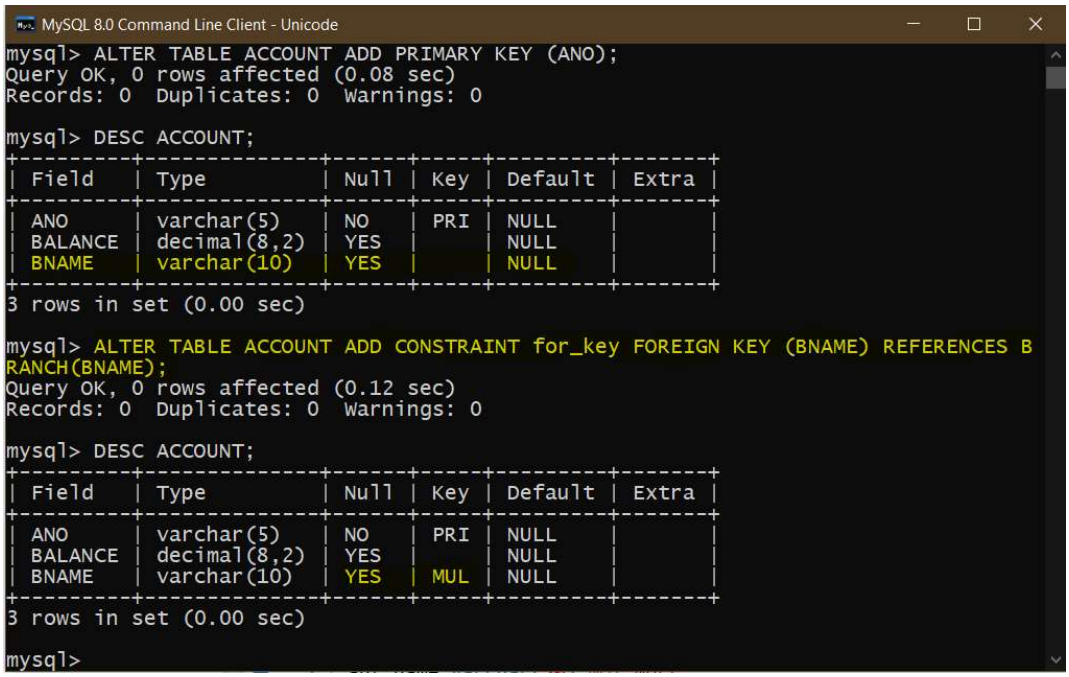
Description:

- This command adds new constraint in an **existing table.**
- If above command violates by the data previously placed in the table then this command will be rejected.

Example: Adding Primary key Constraint To already created Account table



Example : Adding Foreign Key Constraint with name 'for_key' in Account table using ALTER



DROPING CONSTRAINT

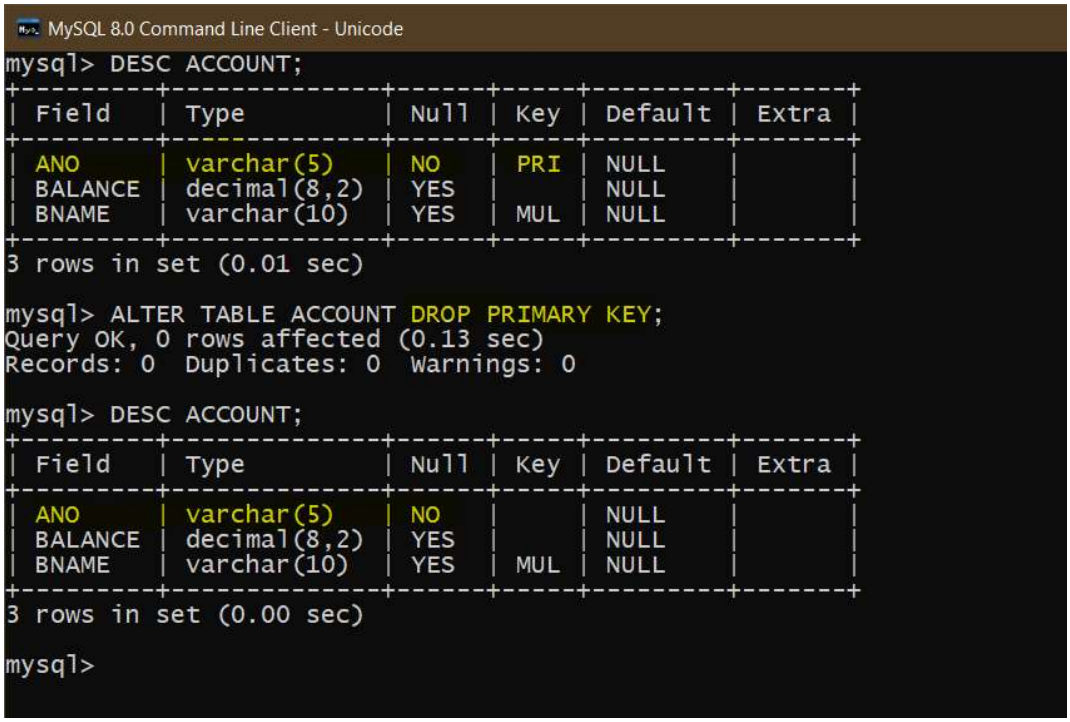
Syntax

- **ALTER TABLE tableName DROP Constraint**

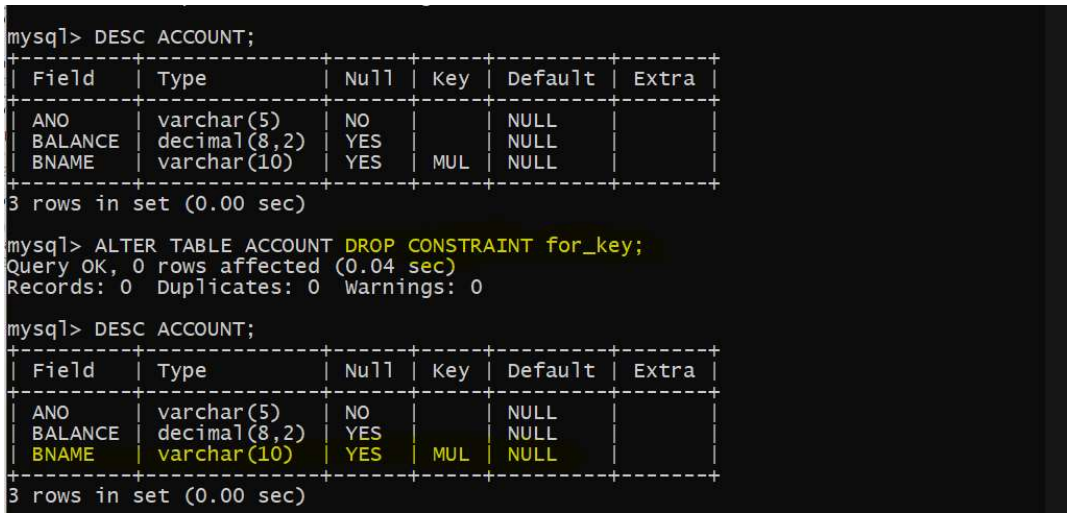
Description

- A Constraint can be dropped directly or can be dropped using pre-assigned name to that constraints .

Example : Drop PRIMARY KEY constraint From Account table directly.



Example : Altering Account table by dropping foreign key named 'for_key' earlier.



Note :

- As you can see that DESC ACCOUNT still showing some value for Key field:
- it may be MySql error , but when we use DROP CONSTRAINT for_key it will delete that constraint.

CHAPTER 9 : UNIQUE , PRIMARY AND FOREIGN KEY CONSTRAINT

Notebook:	CHAPTER 9 : SQL CONSTRAINTS		
Created:	4/3/2020 9:37 AM	**Updated:**	5/11/2020 12:55 PM
Author:	pagulearn.5334@gmail.com		

UNIQUE Constraint

- In Database there may be a requirement that column must have unique values for records in table.
- Which mean no duplicate values should be allowed for that column.
- Example :
  - In Account table , all account numbers must be unique , this is required to identify all records stored in tables uniquely.
- Such kind of restriction can be ensured using a UNIQUE constraint.

Enforced Restriction

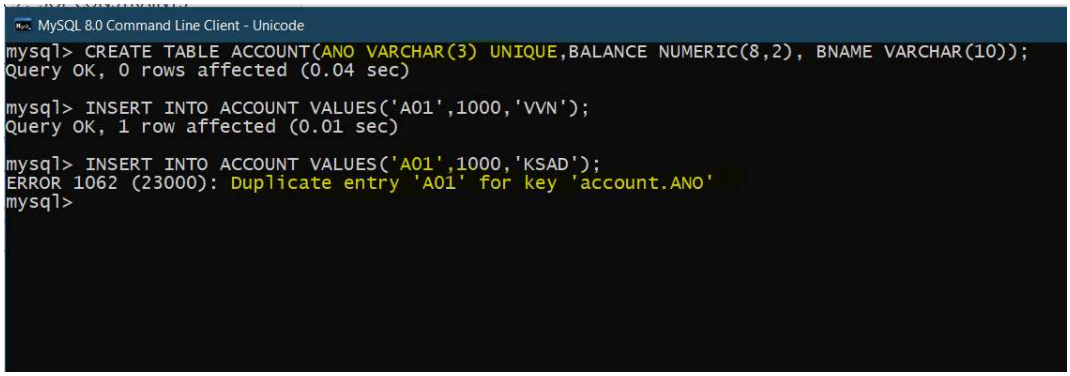
- A column , defined as a UNIQUE cannot have duplicate values across all records.
- Such column must contains unique values.

UNIQUE Constraint defined at Column Level

Syntax

- **columnName datatype (size) UNIQUE**

Example

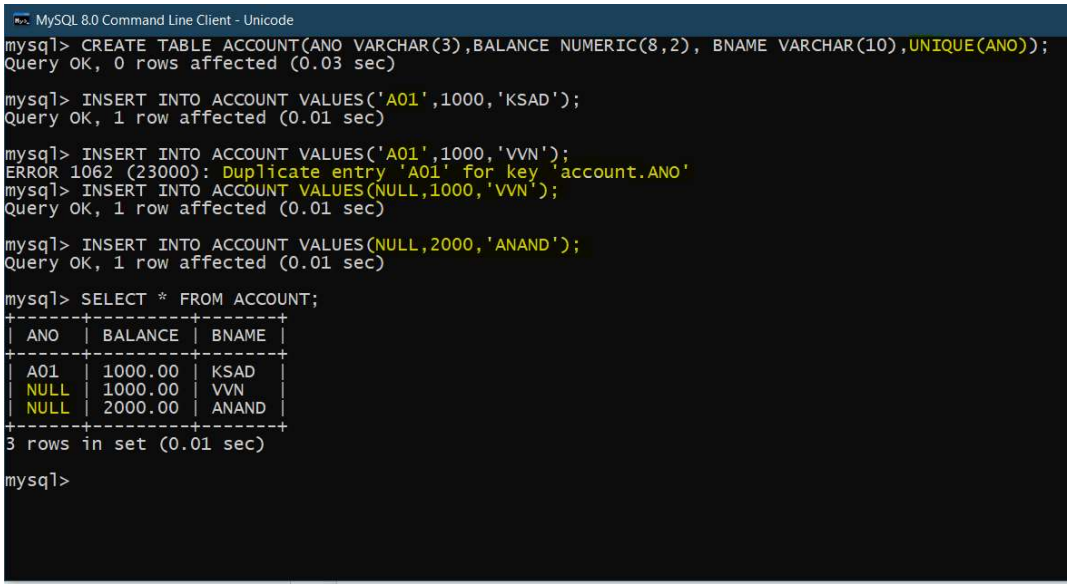


UNIQUE Constraint defined at Table Level

Syntax

- **UNIQUE (columnName [, columnName .....])**

Example :



Features

- UNIQUE constraint **does not allow duplicate values** , but **NULL values can be duplicated in a Column defined as a UNIQUE column.**
- **NULL** values are **ignored by UNIQUE** constraint.
- A Table can have **more than one column** defined as a **UNIQUE column.**
- If **multiple columns** need to defined as **composite UNIQUE columns** , then only **table level definition** is applicable.
    - Example : UNIQUE(ANO , NAME)
    - here , assume that ANO and NAME is the column of Table.
- A column having **LONG** and **LONG RAW data type** , can not be defined as a **UNIQUE** column.

PRIMARY KEY Constraint

- A Primary key is a set of one or more columns used to identify each records uniquely in a table.
- Database should be designed in such a way that each table in a database must contain primary key.
- A single column primary key is called a simple key , multi column primary key is called composite key.
- Oracle provides PRIMARY KEY Constraint to define primary key for a table.

Enforced Restrictions

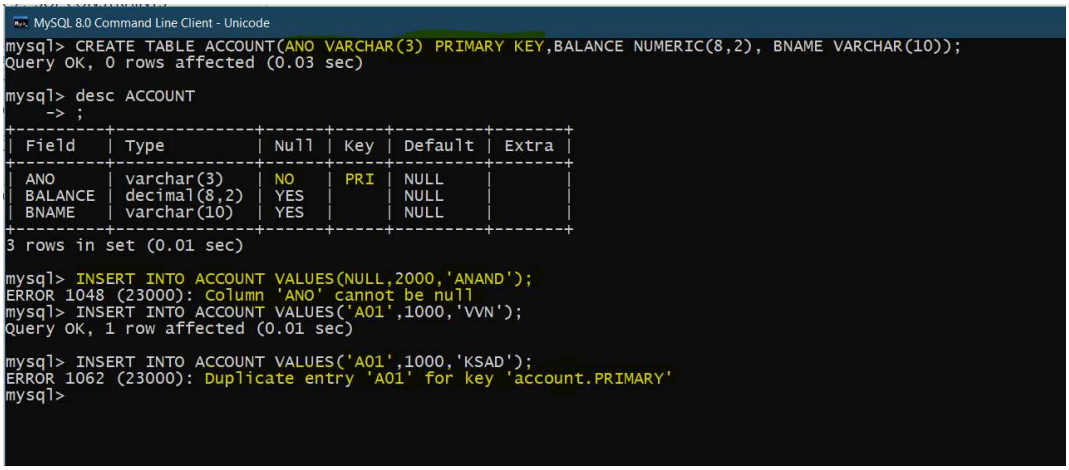
- A Column , defined as a **PRIMARY KEY cannot have duplicate values** across all records. so such column must contain unique values.
- A Column defined as a **PRIMARY KEY cannot have a NULL value.** So such column is a mandatory column.
- Thus , we can say that **PRIMARY KEY constraint is a combination** of **NOT NULL** and **UNIQUE** constraints.

PRIMARY KEY Constraint defined at Column level

Syntax

- **ColumnName datatype(size) PRIMARY KEY**

Example :

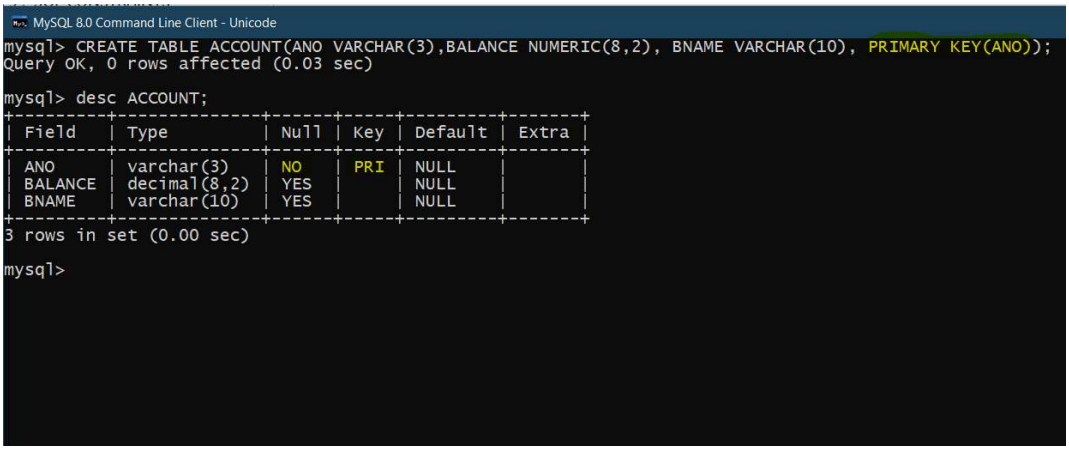


PRIMARY KEY Constraint defined at Table Level

Syntax

- **PRIMARY KEY (columnName [, columnName .....])**

Example:



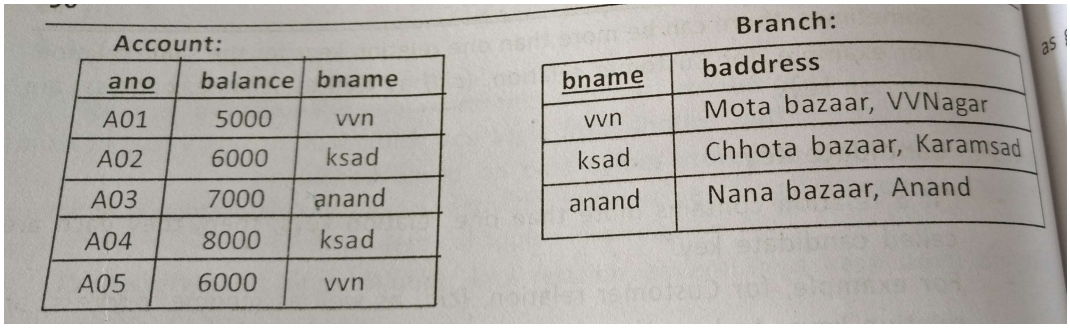
Features:

- A PRIMARY KEY constraint can be considered as a combination of NOT NULL and UNIQUE constraints.
- A PRIMARY KEY is not **compulsory** but it is recommended.
- A table **can not have more than one primary key.**
- A **composite key** can be defined only **at table level.**
- A composite key **do not allow duplicate values in combination** for any records, and **NULL** values are **not allowed** in **individual column** also.
- Maximum 16 columns can be combined as a composite primary key in a table
- A column having LONG or LONG RAW data type can not be defined as a PRIMARY KEY.

FOREIGN KEY Constraints

- Foreign Key constraint also referred as referential integrity constraint , is specified between two tables.
- This constraint is used to ensure consistency among the records of the two tables.

Example :



Account:

ano	balance	bname
A01	5000	vvn
A02	6000	ksad
A03	7000	anand
A04	8000	ksad
A05	6000	vvn

Branch:

bname	baddress
vvn	Mota bazaar, VVNagar
ksad	Chhota bazaar, Karamsad
anand	Nana bazaar, Anand

- In above example bname is the common field between both the tables , which provides link between Account and Branch . it represent which account is related to which branch.
- To maintain consistency between account and its branch , which means account table must have only those branch name in its record which is actually present in Branch table.
- Such kind of consistency can be ensured using FOREIGN KEY constraint. To serve this purpose we will use the concept of foreign key.

FOREIGN KEY

- A foreign key is a set of one or more columns whose values are derived from the primary key or unique key of other table.
- In our example , an attribute bname in Account table is referred as foreign key as its values are derived from the Branch table where it is defined as a primary key.
- A table in which a foreign key is defined is called a foreign table / detail table / child table. [Account table]
- A table of which primary key or unique key is referred is called a primary / master /parent table. [Branch table]

Enforced Restrictions

- The Foreign key constraints enforces different restrictions on detail table and master table .
- Let's assume if bname is defined as a foreign key in Account table referring to bname in Branch table , then there will be following restriction on both these tables.

- **Restrictions on Detail Table [Account]**
  - Detail table contains a foreign key which is related to Master table
  - **Insert / update operation involving value of a foreign key are not allowed , if corresponding value does not exist in the Master table.**
  - So it is not possible to insert new record in Account table having bname = "Surat" because such kind of bname does not exist in Master table (Branch) , even we can not change bname of any existing record in Account table to "surat".

- **Restrictions on Master Table [Branch]**
  - Master table contains a primary key or unique key , which is referred as foreign key in Detail table.
  - **Delete or update operation on records in master table are not allowed , if corresponding record is present in Detail table.**
  - In our Example , it is not possible to delete a record having branch name "vvn" from Branch table , even we can not change branch name from "vvn " to "surat" in Branch table because corresponding record having branch name "vvn" exist in Account table.

FOREIGN KEY constraint defined at Column level

Syntax

- **columnName datatype(size) REFERENCES tableName (columnName) [ON DELETE CASCADE]**
  - Here table name in REFERENCES clause refers to the Master table
  - if columnName from REFERENCES clause is omitted , then the primary key is automatically referred of a table specified in this clause.

Note :

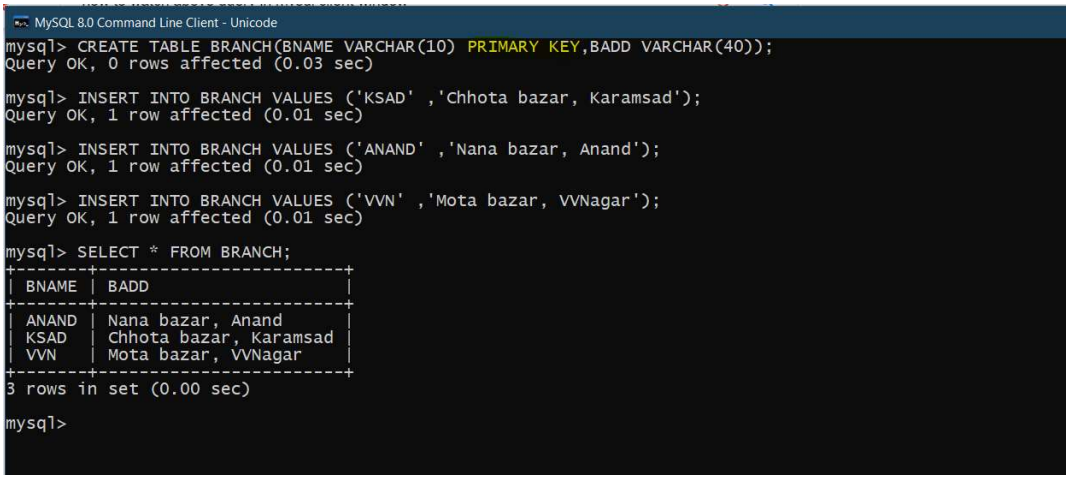
- Above syntax given in the book is for Oracle database which is not working in MySQL.
- So syntax for foreign key constraint defined at column level does not work in Mysql , we will add foreign key constraint using Table level syntax.

FOREIGN KEY constraint defined at Table level

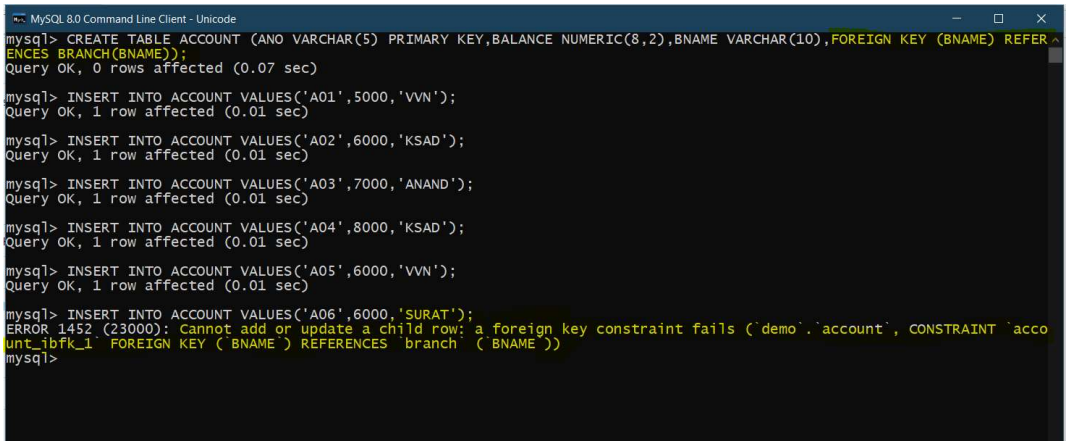
Syntax

- **FOREIGN KEY (columnName [, columnName.....]) REFERENCES tableName (columnName [, columnName....]) [ON DELETE CASCADE]**

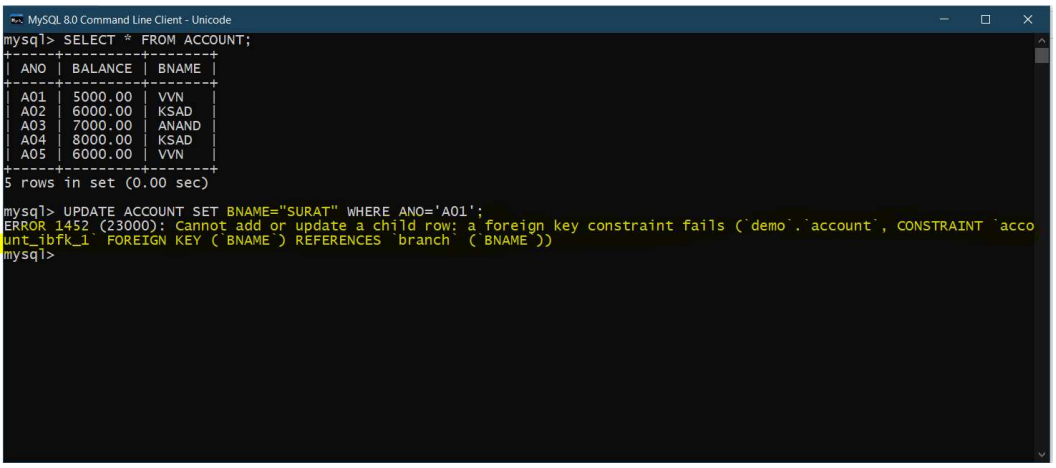
Example



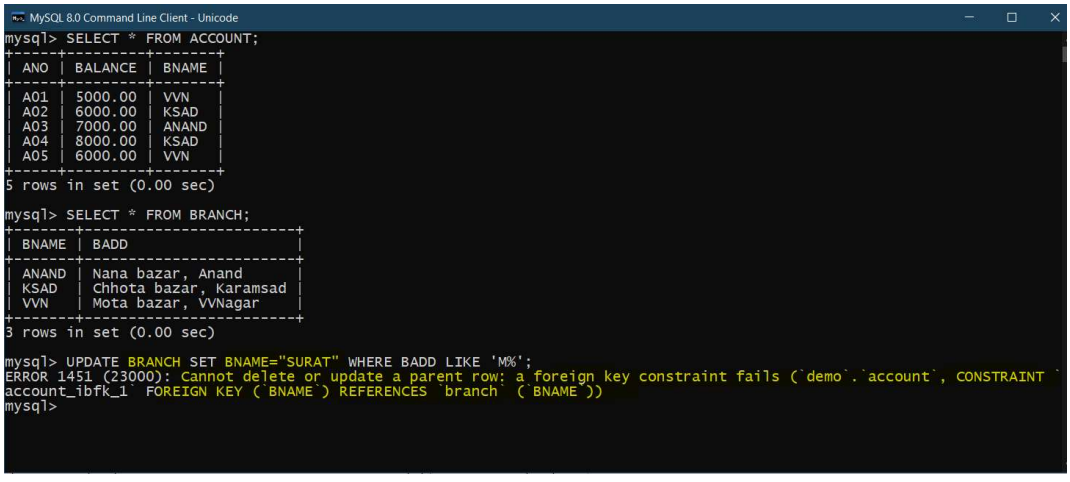
Restriction on Detail Table [Account] while Inserting a Record



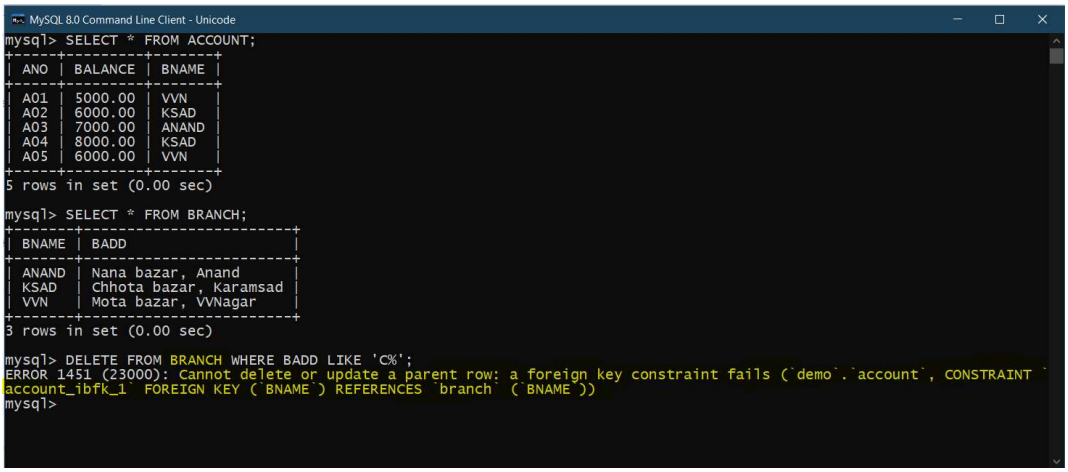
Restriction on Detail Table [Account] while Updating Data



Restriction on Master Table [Branch] While Updating Data



Restriction on Master Table [Branch] While Deleting Data



ON DELETE CASCADE option:

- This option overcomes the restriction enforced on Master table for Delete operation.
- **If we specify this option , then any record we delete from the Master table , all the corresponding records from detail table are also deleted automatically.**
- If this option is on and we delete branch name " vvn" record from the Branch table ,all the record corresponding to "vvn" bname in Account table will also be deleted.