Introduction to SQL.

In a database environment, the client or user interface should send a request to the server (database) and the server should respond to the client's request. For any communication, a language is necessary. The language used here is the Structured Query Language, commonly called SQL. It is a simple English-like language that is easy to understand and write. SQL is tied very closely with the relational model.

Languages in SQL

Data Retrieval Language (DRL) -- retrieves data from the server.
Key word

SELECT	----	Read records from the tables.

Data Definition Language (DDL) -- defines the structure of the objects in the database.
Key words

CREATE	----	creates a new object in the database.
ALTER	----	changes the structure of the objects
DROP	----	removes the object from the database
TRUNCATE	----	removes all the records and retains the structure
RENAME	----	changes the name of the table

Data Manipulation Language (DML) -- Manipulates the existing records
Key words

INSERT	-----	adds a new record or row to the table
UPDATE	-----	changes the values of the existing records
DELETE	-----	removes one or more records from the table.

Transaction Control Language (TCL) -- Controls the transactions
Key words

COMMIT ----- saves the changes in the database.
ROLLBACK ----- undo the change made.
SAVE POINT ----- intermediate buffer or a point between the transactions

Data Control Language (DCL) -- Controls the database actions and data access.
Key words

GRANT ----- provide privileges on the database objects to the users.
REVOKE ----- removes privileges to the database objects

Data Definition Language

It refers to the set of SQL commands, which are used to define the structure of a database object.
Various commands available under DDL are

- CREATE
- ALTER
- TRUNCATE
- RENAME
- DROP

i) CREATE command: It is used for creating objects in the database. Table is the base object in the database.
Syntax

```
CREATE TABLE table_name (
Column1_name data type [constraint] DEFAULT default_value,
Column2_name data type [constraint],
.....
Column_name data type, [Constraint(column_name)]);
```

In this syntax:

First, specify the table name.

Second, list all columns of the table within the parentheses. In case a table has multiple columns, then the columns must be separated by a comma (,).

Column definition is a combination of four things 1) column name followed by its 2) data type and size e.g., INT(3), VARCHAR2, and 3) an optional default value 4) column constraint such as NOT NULL, PRIMARY KEY, CHECK etc..

Third, add table constraints if applicable e.g: PRIMARY KEY, FOREIGN KEY, CHECK etc

Note : Keywords given within [ ] are optional.

Example:

```
SQL> CREATE TABLE students (
        Student_id int PRIMARY KEY,
        Student_name VARCHAR2(10),
        Date_of_birth DATE,
        Mobile_number int(10);
```

The above command creates a table named students with columns Student_id, Student_name etc, having data types NUMBER, VARCHAR2 respectively.

PRIMARY KEY is a constraint, which is specified for a column or combination of columns that ensures unique values. A table can have only one primary key.

Data Types in MySQL

MySQL supports several standard SQL data types. Each column can contain only one data type. In MySQL, data types are grouped in different categories:

.

Numeric

- Date and time
- String

Numeric Data Types (Number Formats)

MySQL supports numeric data types such as integers, decimals, and floating-point data types

Integer data types

TINYINT :	Allowable range from 0 to 255 and width max 4 digits.
SMALLINT	Allowable range from 0 to 65535 and width max 5 digits
MEDIUMINT	Allowable range from 0 to 16777215 and width max 9 digits
INT	Allowable range from 0 to 4294967295 and width max 11 digits
BIGINT	Allowable range from 0 to 18446744073709551615 and width max 20 digits.
FLOAT(m,d)	Display length (m) and decimal (d) max 24 decimal places
DOUBLE(m,d)	max. 53 decimal place.

Date and time data types

DATE	Default format 'yyyy-mm-dd'
TIME	Default format 'HH:MI:SS'
DATETIME	Default format 'yyyy-mm-dd hh:mm:ss'
TIMESTAMP	Default format 'YYYY-MM-DD HH:MI:SS'

String data types

CHAR(n)	Fixed length string	max of 255 characters.
VARCHAR(n)	Variable length string	max 255 characters.
TINYTEXT	Variable length string	max 255 characters
TEXT	Variable length string	max 65535 characters
MEDIUM TEXT	Variable length string	max 16,777,215 char
BLOB	Binary large Object	max 65535 bytes
MEDIUMBLOB	Binary large Object	max 16,777,215 bytes
LONGBLOB	Binary large Object	max 4gb
ENUM	Allows only the values in the given list	

Naming conventions for objects/columns etc

1. Name must start with an alphabet.
2. Can contain numbers and special characters (_, $, #).
3. Must not contain space.
4. Must be unique.

ii) ALTER command: The ALTER table command is used to modify the structure of the table. Modifying the structure includes adding / dropping columns/constraints, changing data types and sizes, and renaming columns.

Adding a column to the table

To add a column to an existing table, use the ALTER TABLE...ADD statement.

Syntax:

ALTER TABLE table_name ADD column_name data type [FIRST |AFTER column_name];

Add Multiple columns

ALTER TABLE table_name ADD (column1_name data type [FIRST |AFTER column_name], ADD column2_name data type… [FIRST |AFTER column_name])

Note : FIRST | AFTER column_name: It is optional. It tells MySQL where in the table to create the column. If this parameter is not specified, the new column will be added to the end of the table.

Example :

```
> ALTER TABLE students ADD address VARCHAR(15);

> ALTER TABLE students ADD  regno INT  VARCHAR(15) FIRST;

> ALTER TABLE students ADD (city VARCHAR(10),

  add pincode MEDIUMINT);
```

Modify the column of a table

Use the ALTER TABLE...MODIFY statement to modify an existing column definition. You can modify column data type, default value, column constraint, and increase or decrease the length of the column.

Syntax:

```
> ALTER TABLE table_name MODIFY column column_name data type(size);
```

Example :

```
> ALTER TABLE students MODIFY column city VARCHAR(20);
```

Drop a column from the table

The columns which are no more needed can be dropped from the table using ALTER TABLE... DROP statement.

Syntax:

```
> ALTER TABLE table_name DROP COLUMN column_name;

> ALTER TABLE table_name DROP column1,  drop column2…);
```

Example :

```
> ALTER TABLE students DROP COLUMN pincode;
```

Note: You cannot drop all columns from a table

Renaming a column

The ALTER TABLE ….. RENAME COLUMN…. statement is used to change the name of the column or change the name of the table.

Syntax:

```
> ALTER TABLE table_name RENAME COLUMN old_name TO new_name
```

Example:

```
> ALTER TABLE students RENAME COLUMN date_of_birth TO dob;
```

Renaming a table

Syntax:

```
> ALTER TABLE table_name RENAME RENAME TO new_name
```

Example:

```
> ALTER TABLE students RENAME TO dob;
```

iii) TRUNCATE command: TRUNCATE is used to delete all the rows in the table, but maintains the structure of the table. It also releases the space allocated. It cannot be rolled back.

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE students;
```

iv) RENAME command: The RENAME command will change the name of the existing table

Syntax:

```
RENAME TABLE old_name TO new_name;
```

```
Renaming Multiple tables

RENAME TABLE old_name1 TO new_name1,

              TABLE old_name2 TO new_name2;
```

Data Manipulation language
(DML)

DML statements are the elements of SQL language which are used to manipulate the values in the database. It is used for adding ,deleting and modifying data

DML consists of the following set of commands.

- INSERT
- UPDATE
- DELETE

Insert statement:
This statement is used for adding / inserting rows to a table in the database.

Syntax:

```
INSERT INTO table_name (column1_name,column2_name….)

VALUES (value1,value2…..);
```

Example:

```
INSERT INTO students_profile(regno,name,age,mailid)
VALUES(1001,'Marshal',18,'marshal@gmail.com' )
```

Note:

- Column name list is optional if values are given for all columns.
- Character and date values must be enclosed within single quotes.
- Date must be in 'YYYY-MM-DD' format.
- Values are case sensitive.

Inserting multiple values:

Syntax:

```
INSERT INTO table_name (c1,c2,...) VALUES (v11,v12,...),
(v21,v22,...), ... (vnn,vn2,...);
```

Example:

```
INSERT INTO tasks(title, priority) VALUES ('My first task', 1),
                                          ('It is the second task',2),
                                          ('This is the third task of the week',3);
```

Inserting default values:

Syntax:

```
INSERT INTO table_name (c1,c2,...) VALUES (v11,v12,.DEFAULT..);
```

Example:

```
INSERT INTO tasks(title,priority) VALUES('My default tsk',DEFAULT);
```

NULL Value: NULL is the term used to represent a missing value or unknown value. It is not equal to 0 nor to Space.

Inserting NULL values

Option 1: Exclude the column names for which NULL values are to be inserted.

Syntax

```
INSERT INTO table_name (column1,column2…)
VALUES(value1,value2…..);
```

Example

```
INSERT INTO students_profile(regno,name,age)
VALUES(1001,'Marshal',18);
```

Option 2: Specify NULL explicitly

```
INSERT INTO students_profile VALUES(1001,'Marshal',18,null);
```

UPDATE statement: The UPDATE statement is used to modify or change the values in a table. Either all the rows can be updated or a subset may be chosen using a condition in the WHERE clause.

Syntax:

```
UPDATE table_name SET column1=value1, column2=value2 [WHERE
column=condition];
```

Example:

```
UPDATE students_profile SET NAME='ADAMS',
WHERE regno=2;
```

Updating Multiple Fields:

If you are going to update multiple fields, you should separate each field assignment with a comma.

Example:

```
UPDATE students  SET User_Name = 'Walkman',
        First_Name = 'Johnny'  WHERE Student_Id = '3'
```

DELETE statement: The DELETE statement removes one or more records from a table. A subset may be defined for deletion using a condition, otherwise all records are removed.

Syntax

```
DELETE FROM table_name [WHERE column=condition];
```

Example

```
DELETE FROM students_profile
WHERE regno=2;
```

To delete all the records

Syntax

```
DELETE FROM table_name;
```

Example

DELETE FROM student_profile;

Transaction Control Language (TCL)

What is a Transaction?

Database Transaction is a single unit of logic or operation that is carried out by a user or application. The operations can include retrieval, (Read), insertion, deletion and modification (Write) A transaction must be either completed or aborted. Transaction that changes the contents of the database must alter the database from one consistent state to another. A consistent database state is one in which all data integrity constraints are satisfied.A successful transaction can change the database from one CONSISTENT STATE to another. DBMS transactions must be atomic, consistent, isolated and durable

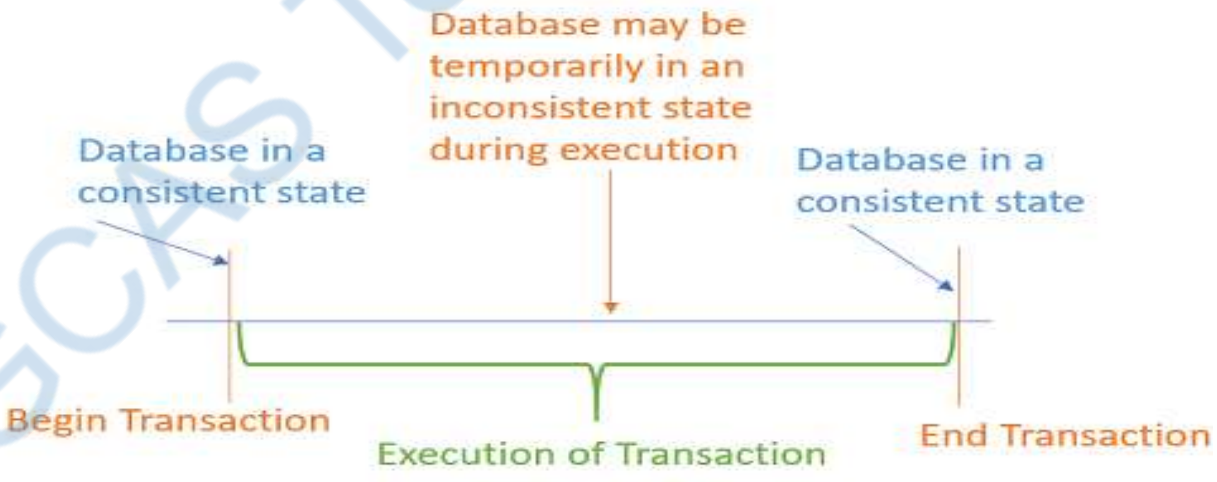


Figure 2.1

Example: Consider the following example of transaction operations to be performed to withdraw cash from an ATM vestibule.

Steps for ATM Transaction

- Transaction Start.
- Insert your ATM card.
- Select a language for your transaction.
- Select the Savings Account option.
- Enter the amount you want to withdraw.
- Enter your secret pin.
- Wait for some time for processing.
- Collect your Cash.
- Transaction Completed.

A transaction can include the following basic database access operation.

- Read/Access data (R): Accessing the database item from disk (where the database stored data) to memory variable.
- Write/Change data (W): Write the data item from the memory variable to the disk.
- Commit: Commit is a transaction control language that is used to permanently save the changes done in a transaction

Desirable properties of transaction

For a transaction to be performed in DBMS, it must possess several properties often called **ACID properties**.

ACID Properties are used for maintaining the integrity of the database during transaction processing. ACID in DBMS stands for Atomicity, Consistency, Isolation, and Durability.

- **A – Atomicity**
- **C – Consistency**
- **I – Isolation**
- **D – Durability**

Atomicity: A transaction is a single unit of operation. You either execute it entirely or do not execute it at all. There cannot be partial execution.

Consistency: Once the transaction is executed, it should move from one consistent state to another.

Isolation: Transactions should be executed in isolation from other transactions (no Locks). During concurrent transaction execution, intermediate transaction results from simultaneously executed transactions should not be made available to each other.

Durability:· After successful completion of a transaction, the changes in the database should persist. Even in the case of system failures.

Transaction Control language.

Transaction Control Language(TCL) commands are used to manage transactions in databases. These are used to manage the changes made by DML statements. It also allows statements to be grouped together into logical transactions.

TCL Commands are as follows

a) **COMMIT:** COMMIT command is used to permanently save any transaction into the database. It ends the current transaction.

syntax

```
COMMIT;
```

b) ROLLBACK: ROLLBACK command restores database to original since the last COMMIT. It is also used with the savepoint command to jump to a savepoint in a transaction.

syntax

```
ROLLBACK;
```

c) **SAVEPOINT :** SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever necessary

syntax:

```
Statement 1;
SAVEPOINT savepoint_name1;
Statement 2;
SAVEPOINT savepoint_name2;
ROLLBACK to SAVEPOINT savepoint_name;
```

Example:

```
INSERT INTO students_profile VALUES (10,'Jeni',18,'jeni@outlook.com';
SAVEPOINT a;
INSERT INTO students_profile VALUES
(11,'Mithun',19,'mithun@outlook.com';
SAVEPOINT b;
UPDATE students_profile SET age=20 WHERE regno=10;
SAVEPOINT c;
DELETE FROM  students WHERE name='Mithun';
ROLLBACK to SAVEPOINT b;
```

Exercises

Answer the following

1. Which of the following is/are the DDL statements?
    a. Create
    b. Drop.
    c. Alter
    d. All of the above

2. Alter table command is used to perform which of the following operations?
    a. Removing one or more records
    b. Modify the values of the records.
    c. Add one or more columns.
    d. Remove the table.

3. Which of the following commands is used to remove a database
    a. Drop database.
    b. Delete database.
    c. Remove database.
    d. None of the above.

4. Examine the structure of the EMPLOYEES table

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
HIREDATE		DATE
DEPARTMENT_ID		NUMBER(4)

Which two statements will insert a row into the EMPLOYEES table? (Choose two.)

A. INSERT INTO employees VALUES (101, 'John', 'Smith', 12000, SYSDATE);

B. INSERT INTO employees VALUES (101, 'John', 'Smith', 10, 12000, SYSDATE);

C. INSERT INTO employees (employee_id, salary, first_name,

hiredate, last_name) VALUES (101, 12100, 'John', SYSDATE, 'Smith');

D. INSERT INTO employees (employee_id, first_name, last_name, salary, hiresate)
VALUES ((SELECT 101, 'John', 'Smith'. 12000, SYSDATE FROM dual));

E. INSERT INTO employees SELECT 101, 'John', 'Smith', 12000, (SELECT SYSDATE FROM dual), 10 FROM dual;

F. INSERT INTO employees VALUES (101, 'John', ' ', 12000, SYSDATE, 10);

5. Which three statements are true about the ALTER TABLE....DROP COLUMN.... command? Choose all that apply.

A. A column can be dropped only if it does not contain any data.
B. A column can be dropped only if another column exists in the table.
C. A dropped column can be rolled back.
D. The column in a composite PRIMARY KEY with the CASCADE option can be dropped.
E. A parent key column in the table cannot be dropped.

6. Examine this command:

TRUNCATE TABLE test;

Table truncated.

Which two are true? (Choose two.)

A. The structure of the TEST table is removed.

B. All the indexes on the TEST table are dropped.

C. All the constraints on the TEST table are dropped.

D. Removed rows can not be recovered using the ROLLBACK command.

E. All the rows in the TEST table are removed.

7. Which three actions can you perform on an existing table containing data? (Choose all that apply.)

A. Increase the width of a numeric column.

B. Add a new column as the table's first column.

C. Define a default value that is automatically inserted into a column containing nulls.

D. Change a DATE column containing data to a NUMBER data type.

E. Change the default value of a column.

F. Add a new NOT NULL column with a DEFAULT value.

8. _____ is used to modify the existing values

a. Modify
b. Alter
c. Change
d. Update

9. Which of the following is not true about the DELETE command.

e. Can remove specific records
f. Can be rolled back
g. Will release the space
h. None of the above.

10. Examine this sequence of statements issued in a new session:

```
INSERT INTO books VALUES
            ('ADV112', 'Adventures of Tom Sawyer', NULL, NULL);
    SAVEPOINT a;
    DELETE FROM books;
    ROLLBACK TO SAVEPOINT a;
    ROLLBACK;
```

11. Which two statements are true? (Choose two.)

a. The second ROLLBACK command replays the delete.
b. The first ROLLBACK command restores the 101 rows that were deleted and commits the inserted row.
c. The first ROLLBACK command restores the 101 rows that were deleted, leaving the inserted row still to be committed.
d. The second ROLLBACK command undoes the insert.
e. The second ROLLBACK command does nothing.

Task 1:

1. Create the following tables for a hotel management system with the attributes given. Choose appropriate data types .
    a. Hotel (hotel_d, hotel_name, location)
    b. Rooms (roomid, hotelid, room_type)
    c. Guest (guestid, guestname, city)
    d. Reservations (roomid, guestid , date_checkin, date_checkout)
2. Store a minimum of 5 records to each of the above tables.
3. Display the records from all the above tables.
4. The mobile number of each guest must be stored. Modify the guest table to store the same.
5. Increase the size of location to 50.

6. Change the name of the column date_ckeckin to checkinDate.
7. Add a column price to the rooms table.
8. Store the price of each room type. Ex: Single room price is 1500. Double room price is 3000 and family suit price is 10000.
9. Remove the details of reservations done on 25-09-2023.
10. Show the structure of all tables.

Task 2:

1. Create the following table for animals with the attributes given. Choose appropriate data types .
   Animals(id, name, species,Life_span).
2. Insert the following records into the above table.

ID	Name	Species	Life_span
1	Cat	Animal	20
2	Dog	Animal	25
3	Monkey	Animal	26
4	Elephant	Animal	65
5	Canary	Bird	20
6	Swift	Bird	5
7	Dove	Bird	12
8	Rabbit	Animal	10
9	Cow	Animal	25

3. Change the length of life_expectancy column in such a way that it can hold a maximum of 3 digits.
4. Add a new column Habitat with a suitable data type.

5. Populate the Habitat column for each animal. The Values can be ‘Domestic’ or ‘Wild’ .
6. Add a new Domestic animal Horse with a life expectancy of 25 years.
7. Increase the life expectancy for all domestic animals by 5 .
8. Delete all the wild animals with life expectancy less than 8 years.
9. Display the details of all animals.
10. Remove all the data from the animals table.

Constraints

Constraints provide integrity to the data that is being used by an application by applying some rules or conditions. In other words, Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table. Constraints can be added to the table while creating the table or after it is created. Constraints can be enabled and disabled whenever needed.

Levels of Constraints

A constraint can be one of the following levels.

- Column-level constraint: These refer to a single column in the table and do not specify a column name (except check constraints). They refer to the column that they follow. These constraints are declared while creating the table.

- Table-level constraint: Table-level constraints refer to one or more columns in the table. Table-level constraints specify the names of the columns to which they apply. These constraints cahn be declared wile creating the table or after creating the table.

Types of Constraints

There are six types of constraints

a. PRIMARY KEY: It uniquely identifies each value of a column or a combination of columns. It also ensures that the column has a value. Table with a PRIMARY KEY is called a master table or parent table. A table can have only one PRIMARY KEY.
b. FOREIGN KEY: It establishes a link between two tables. It refers to the PRIMARY KEY values. A table with foreign key is called a Child table. One or

more foreign keys can be declared for a table and more than one FOREIGN KEY can refer to a single primary key.

c. UNIQUE: enforces the Uniqueness of the values (i.e. avoids duplicate values) on the column(s) on which it is defined. Also these key's can Uniquely identify each row in the database table. It allows null values.
d. NOT NULL: enforces that a column does not hold NULL value which ensures a field to always contain a value. This can be declared only at column level.
e. CHECK: allows you to specify a condition on each row in a table.
f. DEFAULT: Used to set a default value for a column. Default values are added when no other value is specified.

Note:

- PRIMARY KEY and FOREIGN KEY constraints are also called as key constraints.
- When a PRIMARY KEY or a UNIQUE KEY is created , automatically a unique index will be created. The uniqueness of the values are enforced using this index.

Declaring column level constraints.

Syntax:

```
Column_name data type(size) [CONSTRAINT constraint_name ]constraint type;
```

Note: Keywords given within [ ] are optional

Example:

```
CREATE TABLE student_profile(
regno INT  PRIMARY KEY,
name VARCHAR(10) NOT NULL,
age INT,
mailid VARCHAR(30) UNIQUE);
```

Note: If the programmer does not explicitly name constraints, then the server will automatically generate a name.

Declaring column level with user defined names constraints.

Syntax:

```
Column_name data type(size) [CONSTRAINT  constraint_name] constraint type
```

Example:

```
CREATE TABLE student_profile(
regno INT  CONSTRAINT regno_pk PRIMARY KEY,
name VARCHAR(10) NOT NULL,
age INT,
mailid VARCHAR(30) CONSTRAINT  mail_un UNIQUE);
```

Declaring a foreign key constraint.

syntax:

```
CONSTRAINT constraintName    FOREIGN KEY (referringColumnName)
   REFERENCES referredTable  (referredColumn)
   ON UPDATE reference-option
  ON DELETE reference-option
```

Example:

```
CREATE TABLE sem1(
regno INT ,
mark1 NUMBER CHECK  (mark1 between 0 and 100),
mark2 NUMBER CHECK  (mark2 between 0 and 100),
FOREIGN KEY(regno) REFERENCES student_profile(regno) on delete set null
);
```

Reference options.

5 reference options are available.

- CASCADE : If a row from the parent table is deleted or updated, the values of the matching rows in the child table automatically deleted or updated.
- SET NULL: if a row from the parent table is deleted or updated, the values of the foreign key column (or columns) in the child table are set to NULL.
- RESTRICT: if a row from the parent table has a matching row in the child table, MySQL rejects deleting or updating rows in the parent table.
- NO ACTION: is the same as RESTRICT.

Declaring Table level constraint.

Example:

```
CREATE TABLE student_profile(
regno NUMBER,
name VARCHAR(10) NOT NULL,
age NUMBER CONSTRAINT chk_name CHECK(age>18),
mailid VARCHAR(30),
CONSTRAINT rg_pk PRIMARY KEY(regno),
UNIQUE(mailid);
```

Declaring constraints (Except not null) after creating the table

The ALTER TABLE command is used to add a constraint after creating the table. Since all constraints are enabled after creation by default, the values in the column must be compatible.

Syntax:

```
ALTER TABLE table_name ADD [CONSTRAINT constraint_name]
constraint_type(column_name)
```

Example:

```
ALTER TABLE customers ADD CONSTRAINT id_pk PRIMARY KEY(cust_id);
```

Adding a NOT NULL constraint

Syntax:

```
ALTER TABLE table_name MODIFY column_name NOT NULL;
```

Example:

```
ALTER TABLE customers MODIFY cust_name NOT NULL;
```

Default constraint

It is used to assign a default value for a column and it will be added when no other value is specified.

Syntax:

```
column_name data_type DEFAULT value;
ALTER TABLE table_name ALTER column_name DEFAULT value;
```

It can be set while creating the table or can be altered after creating the table. It must be a literal constant i.e. a string,number or date. It cannot be a function or expression.

Dropping a constraint

The DROP CONSTRAINT option in ALTER TABLE command is used to drop a constraint from the table.

DROP a UNIQUE Constraint

Syntax

```
ALTER TABLE table_name DROP INDEX constraint_name;
```

DROP a PRIMARY KEY Constraint

Syntax

```
ALTER TABLE table_name DROP PRIMARY KEY;
```

Drop a FOREIGN KEY Constraint

Syntax

```
ALTER TABLE table_name DROP FOREIGN KEY constraint_name;
```

DROP a CHECK Constraint

Syntax

```
ALTER TABLE table_name DROP CHECK constraint_name;
```

Check the name of the Constraints declared for a table

```
SELECT TABLE_NAME, CONSTRAINT_TYPE, CONSTRAINT_NAME
FROM information_schema.table_constraints
WHERE table_name='student';
```

Auto Increment

Auto Increment is a function that operates on numeric data types. It automatically generates sequential numeric values every time that a record is inserted into a table for a field defined as auto increment. It is used to generate primary key values. By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

Example

```
CREATE TABLE `categories` (
  `category_id` int(11) AUTO_INCREMENT,
  `category_name` varchar(150) DEFAULT NULL,
  `remarks` varchar(500) DEFAULT NULL,
  PRIMARY KEY (`category_id`)
);
```

To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement

Syntax

```
ALTER TABLE table_name AUTO_INCREMENT=value;
```

Example

ALTER TABLE categories AUTO_INCREMENT=100;

Exercises

Answer the following

1. What is the difference between column level and table level constraints
    a. Constraints are applied to a single row using Column Level Constraints whereas Multiple rows can be constrained using a Table Level Constraint.
    b. Constraints are applied to multiple rows using Column Level Constraints whereas a single row can be constrained using a Table Level Constraint.
    c. Constraints are applied to a single column using Column Level Constraints whereas Multiple columns can be constrained using a Table Level Constraint.
    d. Constraints are applied to multiple columns using Column Level Constraints whereas only a single column can be constrained using a Table Level Constraint.

2. What is true about not null constraints.

    a. In columns that are subject to the NOT NULL constraint, duplicate values are not allowed.
    b. When a table's column is declared as NOT NULL, no record in the table can have an empty value for that column.
    c. By applying the NOT NULL constraint, we will always ensure that the column contains a unique value and won't allow nulls.
    d. The value will first be checked for certain conditions before inserting it into the column when a NOT NULL constraint applies to a column in the table.

3. Examine the following SQL statement

SQL> ALTER TABLE books_transactions ADD CONSTRAINT fk_book_id FOREIGN KEY (book_id) REFERENCES books (book_id) ON DELETE CASCADE;

What does ON DELETE CASCADE imply?

a. When the BOOKS table is dropped, the BOOK_TRANSACTIONS table is dropped.
b. When the BOOKS table is dropped, all the rows in the BOOK_TRANSACTIONS table are deleted but the table structure is retained.
c. When a row in the BOOKS table is deleted, the rows in the BOOK_TRANSACTIONS table whose BOOK_ID matches that of the deleted row in the BOOKS table are also deleted.
d. When a value in the BOOKS.BOOK_ID column is deleted, the corresponding value is updated in the BOOKS_TRANSACTIONS.BOOK_ID column.

4. What is a Foreign key constraint?

a. Will not allow duplicate values but allow null value
b. Used identify the records uniquely
c. Will allow null values but not duplicate values
d. Refers the values in the primary key column.

5. What is TRUE about DEFAULT constraints?

a. The value will first be checked for certain conditions before inserting it into the column when a DEFAULT constraint applies to a column in the table.
b. In the event of a DEFAULT constraint being applied to a table's column without a user specifying the value to be inserted when that constraint was applied, the default value that was specified when the constraint was applied will be put into that column.
c. An index can be created on the table using the DEFAULT constraint.
d. None of the above

6. View the exhibit and examine the ORDERS table.

ORDERS

NAME	Null?	Type

ORDER ID	NOT NULL	INT(4)
ORDATE		DATE
CUSTOMER ID		INT(3)
ORDER TOTAL		FLOAT(7,2)

The ORDERS table contains data and all orders have been assigned a customer ID. Which statement would add a NOT NULL constraint to the CUSTOMER_ID column?

a. ALTER TABLE orders MODIFY CONSTRAINT orders_cust_id_nn NOT NULL (customer_id);
b. ALTER TABLE orders ADD CONSTRAINT orders_cust_id_nn NOT NULL (customer_id};
c. ALTER TABLE orders MODIFY customer_id CONSTRAINT orders_cust_nn NOT NULL ;
d. ALTER TABLE orders ADD customer_id NUMBER(6)CONSTRAINT orders_cust_id_nn NOT NULL;

7. _____ is the preferred method to maintain data integrity.

a. Triggers
b. Stored procedures
c. Application Logic
d. Constraints

8. Which type of constraint in mysql allows you to define custom data validation rules?

a. Primary key
b. Check
c. Not Null
d. Foreign key
e. Unique

9. Which type of constraint automatically assigns a predefined value to a column if no value is provided during insertion

a. Primary key

b. Check
c. Default
d. Foreign key
e. Unique

10. Which of the following constraints does not enforce Uniqueness (choose two
a. Unique
b. Primary key
c. Foreign key
d. Not null

Writing basic select statements

The ***SELECT*** statement which represents the Data Query language in SQL is used to fetch data from one or more tables. It can consist of the list of columns to be fetched, or some additional clauses which specify what and how the data should be returned. It is used in conjunction with the FROM clause to extract data from the table.

Capabilities of select statement.

Projection: You can use the projection capability in SQL to choose the columns in a table that you want returned by your query. You can choose as few or as many columns of the table as you require.

Selection: You can use the selection capability in SQL to choose the rows in a table that you want returned by a query. You can use various criteria to selectively restrict the rows that you see.

Join: You can use the join capability in SQL to bring together data that is stored in different tables by creating a link through a column that both the tables share. You will learn more about joins in a later lesson.

Projection: Project specific column from the table

syntax:

```
SELECT  * | DISTINCT  | column1,column2,Expression, function ……..
FROM table_name;
```

Projecting all columns

Use * to project all columns from the table

Syntax:

```
SELECT * FROM table_name;
```

Example:
Find the details of all employees from the EMP table.

```
SELECT * FROM emp;
```

Selecting specific columns

List the names of the columns after select

Syntax :

```
SELECT column1,column2,..... FROM table_name;
```

Example :

Display the employee number,employee name , salary and department number of all employees.

```
SELECT empno, ename, sal, deptno FROM emp;
```

Note that the select statements are not case sensitive.

Eliminating the duplicate records

Use DISTINCT keyword

Syntax:

```
SELECT DISTINCT column_name,column_name….  FROM table_name;
```

Display the jobs available in the emp table.

```
SELECT DISTINCT job FROM emp;
```

Note : Only one distinct can be given for a select statement.

Sorting the output:

The order of rows that are returned by the query is undefined. To sort the output use ORDER BY clause. The ORDER BY clause must be the last clause in the select statement. The default mode of sorting is ascending order. For descending order use DESC.

Syntax:

```
SELECT column_list FROM table_name
WHERE condition
ORDER BY column_name/s,| columnAlias | column_position ;
```

Examples:

```
SELECT * FROM emp ORDER BY sal;
```

```
SELECT * FROM emp WHERE deptno=20 ORDER BY hiredate DESC;
```

```
SELECT * FROM emp ORDER BY deptno ,sal DESC;
```

```
SELECT * FROM emp ORDER BY 8 DESC;
```

Pagination:

Pagination is a technique used to divide large result sets into smaller pieces or pages.

Pagination is implemented using *LIMIT* and *OFFSET* clauses.

LIMIT clause is used to specify the maximum number of rows to return.

OFFSET clause is used to skip a certain number of rows before starting to return to rows.These two clauses must be the last clause after order by clause.

syntax:

```
SELECT column_list from table_name
Where condition
Order by column_name
Limit n  offset n;
```

Example:

Display the details of first 2 salesman from emp table

```
SELECT * FROM emp
WHERE job =  'SALESMAN'
LIMIT 2,
```

Example:

Display the details of second least paid manager

```
SELECT * FROM emp
WHERE job =  'MANAGER'
ORDER BY  sal
LIMIT 1 offset 1;
```

Column Alias

It is a temporary name given to a column or an expression in a select statement and used to improve readability particularly when performing calculations.

Use as keyword which is optional to specify the column alias. Enclose within quotes if the alias contains space

syntax:

```
SELECT column_name  as alias , expression  as "alias name" from table name
```

Example :

```
SELECT ename as Name , sal*12 as "Annual salary" from emp;
```

Selection :

The selection capability is used to filter the output based on conditions

The WHERE clause allows you to filter the output. It directly follows the FROM clause and if the condition is true, then the row is processed.

Syntax:

SELECT column_list FROM table_name WHERE condition ;

Examples:

SELECT * FROM emp WHERE ename='SCOTT';
SELECT ename,job,sal FROM emp WHERE sal>=1500;

Operators:

An operator manipulates individual data items and returns a result. The data items are called operands or arguments. These arguments can be user defined values or column names. Operators can be classified as

a) Arithmetic operators.
b) comparison or relational operators.
c) Logical operators.

A) Arithmetic operators.

Just as the name implies these operators are used to perform arithmetic computations.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division

Examples:

Calculate the salary of all employees after a raise of 100.

```
SELECT ename,sal,sal+100 FROM emp;
```

Display the annual salary of all employees after an increase of 1000

```
SELECT ename,sal,(sal+1000)*12 FROM emp;
```

Note

The order of precedence of the arithmetic operators

Parenthesis ()
Multiplication *
Division /
Addition + ,
subtraction -

B) Comparison operators

These are used in conditions that compare one expression with another.

Operator	Description	Example
=	Equality test	***SELECT * FROM emp WHERE ename='SCOTT';***
!= or <>	Inequality test	***SELECT ename,sal FROM emp WHERE job!='MANAGER'';***
>	Greater than	***SELECT * FROM emp WHERE sal>1600;***
>=	Greater than or equal to	***SELECT * FROM emp WHERE SAL>=2000;***

<	Less than	***SELECT * FROM emp WHERE sal<1600;***
<=	Less than or equal to	***SELECT * FROM emp WHERE SAL<=2000;***
[NOT] IN	Matching records in the given set of values	***SELECT * FROM emp WHERE deptno IN (10,20,30);***
[NOT]BETWEEN … AND	Select the rows within the specified range	***SELECT * FROM emp WHERE sal BETWEEN 1000 AND 2000;***
[NOT] LIKE	Select the rows if a character string matches the specified pattern	***SELECT ename FROM emp WHERE ename LIKE 'A%';***
IS [NOT] NULL	Tests for null.	***SELECT * FROM emp WHERE comm IS NULL;***

C) Logical Operators:

These operators compare two or more conditions at a time to determine whether a row can be processed or not. In other words these operators allow us to combine two or more conditions.

Operator	Description	Example
AND	select the row if both the conditions are true	***SELECT ename,job,sal FROM emp WHERE job='CLERK' AND sal <1200;***
OR	select the row if either one conditions is true	***SELECT * FROM emp WHERE deptno=10 OR deptno=20;***
NOT	select the row if conditions is false	***SELECT * FROM emp WHERE deptno NOT IN (10,20)***

Choose the correct choice to answer the following :

1. Examine the description of the PROMOTIONS table:

Name	Null?	Type
PROMO_ID	NOT NULL	NUMBER(6)
PROMO_NAME	NOT NULL	VARCHAR2(30)
PROMO_CATEGORY	NOT NULL	VARCHAR2(30)
PROMO_COST	NOT NULL	NUMBER(10,2)

You want to display the unique promotion costs in each promotion category.
Which two queries can be used? (Choose two.)

A. SELECT DISTINCT promo_category,promo_cost FROM promotions ORDER BY promo_category;

B. SELECT DISTINCT promo_cost , DISTINCT promo_category FROM promotions;

C. SELECT DISTINCT promo_category, promo_cost FROM promotions ORDER BY 1;

D. SELECT promo_category DISTINCT promo_cost, FROM promotions ORDER BY 2;

E. SELECT promo_cost, promo_category FROM promotions ORDER BY 1;

2. Which of the following statements are true about SELECT statements. Choose all that apply.
   A. Select statements are case sensitive.
   B. Select statements can be written in multiple lines
   C. Keywords cannot be abbreviated.
   D. No order for the placement of clauses (FROM ,WHERE , ORDER BY)
   E. Select statements can be used to read records from multiple tables.

3. Examine the description of the MEMBERS table:

Name	Null?	Type
MEMBER_ID	NOT NULL	VARCRAR2(6)
FIRST_NAME		VARCHAR2(50)
LAST_NAME	NOT NULL	VARCHAR2(50)
ADDRESS		VARCHAR2(50)
CITY		VARCHAR2(25)

Examine the partial query:

SELECT city, last_name AS lname FROM members;

You want to display all cities that contain the string AN. The cities must be returned in ascending order, with the last names further sorted in descending order.

Which two clauses must you add to the query?

A. ORDER BY 1, 2

B. ORDER BY 1, lname DESC

C. WHERE city LIKE '%AN%' ORDER BY last_name DESC , city ASC

D. WHERE city = '%AN%' ORDER BY last_name , city ASC

E. WHERE city LIKE '%A%N%' ORDER BY last_name DESC , city ASC

F. WHERE city IN ('%AN%') ORDER BY last_name DESC, city ASC

4. Which two are true about the precedence of operators and conditions? (Choose two.)

A. * has a higher order of precedence than + (addition).

B. ( ) has a lesser order of precedence than any other operator.

C. NOT has a higher order of precedence than AND and OR in a condition.

D. AND and OR have the same order of precedence in a condition.

E. The order of precedence is the same as the order given in the expression.

5. Which statement is true regarding the default behavior of the ORDER BY clause

A. In a character sort values are case-sensitive.

B. NULL values are not considered at all by the sort operation.

C. Only those columns listed in the SELECT list can be used in the ORDER BY clause.

D. Numeric values are displayed from the maximum to the minimum value if they have any decimal positions

6. Examine the structure of the BOOKS_TRANSACTIONS table:

Name	Type
TRANSACTION_ID	VARCHAR2 (6)
BORROWED_DATE	DATE
DUE_DATE	DATE
BOOK_ID	INT
MEMBER_ID	VARCHAR2 (6)

You want to display the member IDs, due date, and late fee as $2 for all transactions. Which SQL statement must you execute?

A. SELECT member_id AS MEMBER_ID, due_date AS DUE_DATE, $2 AS LATE_FEE FROM BOOKS_TRANSACTIONS;

B. SELECT member_id 'MEMBER ID', due_date 'DUE DATE', '$2 AS LATE FEE' FROM BOOKS_TRANSACTIONS;

C. SELECT member_id AS "MEMBER ID", due_date AS "DUE DATE", '$2' AS "LATE FEE" FROM BOOKS_TRANSACTIONS;

D. SELECT member_id AS "MEMBER ID", due_date AS "DUE DATE", $2 AS "LATE FEE" FROM BOOKS_TRANSACTIONS;

7. Examine the description of the EMPLOYEES table:

NAME	TYPE
EMPNO	INT
LAST_NAME	VARCHAR(50)
HIREDATE	DATE
SALARY	FLOATt(8,2)
DEPTNO	INT

For each employee in department 90 you want to display:
1. their last name
2. the number of complete weeks they have been employed
The output must be sorted by the number of weeks, starting with the longest serving employee first.
Which statement will accomplish this?

A.
```
SELECT last_name, ROUND((SYSDATE - hire_date) / 7) AS tenure
  FROM employees
 WHERE department_id = 90
 ORDER BY tenure DESC;
```
B.
```
SELECT last_name, TRUNC((SYSDATE - hire_date) / 7) AS tenure
  FROM employees
 WHERE delpartment_id = 90
 ORDER BY tenure DESC;
```
C.

```
SELECT last_name, ROUND((SYSDATE - hire_date) / 7) AS tenure
  FROM employees
 WHERE department_id = 90
 ORDER BY tenure;
```

D.

```
SELECT last_name, TRUNC((SYSDATE - hire_date) / 7) AS tenure
  FROM employees
 WHERE department_id = 90
 ORDER BY tenure;
```

8.Examine the description of the PRODUCT_DETAILS table:

Name	Null?	Type
PRODUCT_ID	NOT NULL	NUMBER(2)
PRODUCT_NAME	NOT NULL	VARCHAR2(25)
PRODUCT_PRICE		NUMBER(8,2)
EXPIRY_DATE		DATE

Which two statements are true? (Choose two.)

A. EXPIRY_DATE contains the SYSDATE by default if no date is assigned to it.

B. PRODUCT_PRICE can be used in an arithmetic expression even if it has no value stored in it.

C. PRODUCT_NAME cannot contain duplicate values.

D. EXPIRY_DATE cannot be used in arithmetic expressions.

E. PRODUCT_PRICE contains the value zero by default if no value is assigned to it.

F. PRODUCT_ID can be assigned the PRIMARY KEY constraint.

9. Examine the description of the EMPLOYEES table

NAME	TYPE
EMPID	INT(3)
FIRST_NAME	VARCHAR(200)
LAST_NAME	VARCHAR(200)
SALARY	FLOAT(6,2)
DEPTNO	INT(2)

Which two queries will result in an error? (Choose two.)

A.

```
SELECT first_name last_name
  FROM employees;
```

B.

```
SELECT first_name, last name
  FROM employees;
```

C.

```
SELECT last_name, 12 * salary AS annual_salary
  FROM employees
 WHERE annual_salary > 100000
 ORDER BY 12 * salary;
```

D.

```
SELECT last_name, 12 * salary AS annual_salary
  FROM employees
 WHERE 12 * salary > 100000
 ORDER BY 12 * salary;
```

E.

```
SELECT last_name, 12 * salary AS annual_salary
  FROM employees
 WHERE annual_salary > 100000
 ORDER BY annual_salary;
```

F.

```
SELECT last_name, 12 * salary AS annual_salary
  FROM employees
 WHERE 12 * salary > 100000
 ORDER BY annual_salary;
```

10. Which statement will execute successfully, returning distinct employees with non-null first names?

A. SELECT first_name, DISTINCT last_name FROM employees WHERE first_name <> NULL;

B. SELECT first_name, DISTINCT last_name FROM employees WHERE first_name IS NOT NULL;

C. SELECT DISTINCT * FROM employees WHERE first_name IS NOT NULL;

D. SELECT DISTINCT * FROM employees WHERE first_name <> NULL;

Exercises

Read the questions below and write suitable queries

1. The city table is described as follows

CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

a. Find the name , district and Population of all cities.
b. Write a query to find the district and population of all American cities with a population more than 100000. The country code for America is USA.
c. Query all the columns for a city in the city table with ID 1661.
d. Write a query to find the details of all Japanese cities . The country code for Japan is JPN.
e. Write a query to find the name and population of Brooklyn district in New York city.

2. The customer table has the following structure and the values are given below.

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3007	Brad Davis	New York	200	5001
3005	Graham Zusi	California	200	5002
3008	Julian Green	London	300	5002
3004	Fabian Johnson	Paris	300	5006
3009	Geoff Cameron	Berlin	100	5007
3003	Jozy Altidor	Moscow	200	5007
3001	Brad Guzan	London		5005

a. Write a query to find the names of the customers who have ‘o’ as second from the last character. Ex: Marg**o**n.
b. Find the Customer_id, cust_name,city ,grade and salesman_id of the customers who do not live in New York and have a grade value that exceeds 100.
c. Find the customers who live in California and London and have a rating from 200 to 300.
d. Find the customers who have not received any grading
e. Find the details of the customers who were attended by salesman 5002.
f. Display the names of the cities in alphabetical order and then the names of the customers in reverse order within each city.

g. Display the unique cities from the customer table.
h. Display the details of the customers who live in London and attended by salesmen either 5002 or 5005.

Answers to MCQs

1. A, C
2. B,C,E
3. C
4. A,C
5. A
6. C
7. C
8. B,F
9. C, E
10. C

MySQL Functions

MySQL built-in Functions

A function is a programming unit, which is used to compute values. Functions accept arguments and return values. The arguments can be a user defined constant or column name. The Main features of functions are

- ➔ Perform calculations on data.
- ➔ Modify individual data items.
- ➔ Manipulate output for group of rows
- ➔ Formats dates and numbers for display.
- ➔ Convert column data types.

Types of functions

SQL Functions are broadly categorized into two. 1) Single row functions and 2) Multiple row functions or group functions or aggregate functions.

SINGLE ROW FUNCTIONS

These functions act on each row that is returned by the query and gives one result per row.

- ★ Can manipulate data items
- ★ Accepts arguments and returns values.
- ★ Can modify data types
- ★ Can be nested.
- ★ Can be used in the select list, where clause etc.

Types of single row functions

There are 5 types of single row functions

a) **CHARACTER FUNCTIONS:** These functions accept character values as input and return either character or number values as output. Character functions can be classified into

- ➢ Case manipulation functions
- ➢ Character manipulation functions

i) **Case Manipulation functions:**

These functions convert the case for the given character string. Accept only one argument.

Types of case Manipulation Functions:

- ➢ UPPER/UCASE: convert the given character string into upper case.
- ➢ LOWER/LCASE: convert the given character string into lower case.

syntax:

```
SELECT UPPER(stringvalue),LOWER(stringvalue) FROM emp;
```

Example:

```
SELECT ename,UPPER(ename),LOWER(ename) FROM emp;
```

ii) Character manipulation functions: These functions are used to extract, change or format the given character string.

Types of character manipulation functions:

- ➢ **CONCAT**: Joins values together. Accepts n no. of arguments..

Syntax

```
CONCAT(value1,value2.value3….)
```

Example:

```
SELECT CONCAT(ename,job) FROM emp;
```

- ➢ **CONCAT_WS**: adds two or more expressions together with a separator.

Syntax

```
CONCAT_WS('seperator'(value1,value2)
```

Example:

```
SELECT CONCAT_WS('-',ename,job) FROM emp;
```

➢ **INSERT**: inserting a string within a string, removing a number of characters from the original string.

Syntax

```
INSERT('string','position',length,'string2')
```

Example:

```
SELECT INSERT('123456789012',8,4,'XXXX');
```

➢ **SUBSTR**: Returns 'n' no of characters from the given string starting from the 'm' th position. Accepts a maximum of 3 arguments.

Syntax:

```
SUBSTR('string_value',m,n)
```

Example:

```
SELECT SUBSTR('hello',1,3);
```

➢ **LENGTH:** Returns the length of the given string. Accepts only one argument.

Syntax:

```
SELECT LENGTH('String')  FROM table_name;
```

Example:

```
SELECT LENGTH(ename) FROM emp;
```

➢ **INSTR:** Searches for a character or a substring in the given string and returns its position in the string

Syntax:

```
SELECT INSTR('string','substring') FROM table_name;
```

Example:

```
SELECT INSTR(ename,'T') FROM  emp;
```

➢ **LPAD:** Returns the string value left padded with the given pad_value. The length of the whole string will be of 'n' characters.Accepts three arguments.

syntax

```
LPAD('string_value',pad_value',n);
```

Example :

```
SELECT LPAD(sal,'*',10) FROM emp
```

➢ **RPAD:** Returns the string value right padded with the given pad_value. The length of the whole string will be of 'n' characters. Accepts three arguments.

Syntax

```
RPAD('string_value',pad_value',n);
```

Example :

```
SELECT RPAD(sal,'*',10) FROM emp
```

➢ **REPLACE:** It is used to replace all presence of a substring within a string, with a new substring.

Syntax:

```
REPLACE ( string, findstr, replacewith )
```

Example :

```
SELECT REPLACE (ename,'A','#') FROM emp;
```

➢ **REVERSE :** Returns the given string in reverse order.

syntax:

```
REVERSE  ( string)
```

Example:

```
SELECT REVERSE  ( ename) FROM emp;
```

➢ **TRIM :** Removes the leading or trailing spaces or specific characters from the given string.

Syntax:

```
TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)
```

Example :

```
SELECT TRIM(LEADING FROM "      www.kgcas.com          " )
SELECT LTRIM( "      www.kgcas.com    " )
SELECT TRIM(TRAILING FROM "      www.kgcas.com    " )
SELECT RTRIM( "      www.kgcas.com    " )
SELECT TRIM(BOTH FROM "      www.kgcas.com    " )
SELECT TRIM( "      www.kgcas.com    " )
SELECT TRIM(LEADING 'www'  FROM "      www.kgcas.com    " )
```

➢ **STRCOMP :** Compares two strings and returns the following

If two strings are equal then it returns 0.
If Str1 >str2 then returns 1
If Str1 < Str2 then returns -1
If any of the string is NULL then it returns NULL

Syntax:

```
STRCOMP(string1, string2)
```

Example :

```
SELECT STRCOMP('hello','world );
SELECT STRCOMP('WELCOME','ALL');
SELECT STRCOMP('Good',null);
```

➢ **REPEAT :** Repeat the given string n no. of times. Accepts two arguments.

Syntax:

```
REPEAT (string,n)
```

Example :

```
SELECT REPEAT('hello',3 );
```

b) NUMBER FUNCTIONS:

Number functions accept numeric values as input and return numeric values as output. The most commonly used number functions are

➢ **ROUND** : Round off the given number to the nearest integer.Accepts two arguments.

Syntax:

```
ROUND (x,y)
```

*** x is the numeric value and round off to 'y' decimal places*

Example:

```
SELECT ROUND(54.926,2) FROM dual;
SELECT ROUND(54.926,-1) ;
```

➢ **TRUNCATE**: Truncate the value 'x' up to 'y' decimal places. Accepts two arguments.

Syntax:

```
TRUNCATE (x,y)
```

** *x is the numeric value and truncate to 'y' decimal places*

Example:

```
SELECT TRUNCATE(54.926,2);
SELECT TRUNCATE(54.926,-1);
```

➢ **MOD** :Returns the remainder of 'x' and 'y'. Accepts two arguments.

Syntax:

```
MOD(x,y)
```

Example :

```
SELECT (10,3);
```

➢ **CEIL** :Returns the smallest integer value that is bigger than or equal to the given number. Accepts only one argument

Syntax:

```
CEIL(N)
```

Example :

```
CEIL (249.78);
```

➢ **POW / POWER** : returns the value of a number raised to the power of another number.

Syntax:

```
POW/POWER(X,Y)
```

➢ **PI** : Returns the value of PI. Accepts no arguments

Syntax:

PI()

➢ **SQRT**: Returns the Square root of the given number.Accepts only one argument.

Syntax:

SQRT(n)

➢ **GREATEST / LEAST**: Returns the greatest/smallest value for the given list of values. Accepts n no of arguments.

Syntax:

GREATEST(value1, value2,value3….)

Exampe

SELECT GREATEST(100, 150,125);

C) Date functions:

Mysql Stores the date in 'YYYY-MM-DD' format. Optimally time values can also be stored with fractional seconds. These functions operate with date values and return either date or time or number values as output.

Types of date functions

➢ **CURRENT_DATE() /CURDATE()**: Returns the current date in 'YYYY-MM-DD' format. Accepts no arguments

Syntax:

CURRENT_DATE()/ CURDATE( )

Example:

```
SELECT CURRENT_DATE(), CURDATE( );
```

- **SYSDATE( ) / *NOW()***: Returns the date with time of execution in 'YYYY-MM-DD HH:MI:SS' format. Accepts no arguments

Syntax:

```
SYSDATE();
```

Note: The key difference between sysdate() and now() is that *SYSDATE()* returns the actual time when the function is evaluated but *NOW()* returns the time when the query starts executing.

- **CURRENT_TIME()/CURTIME()**: Returns the current time in HH:MI-SS' format. Accepts no arguments

Syntax:

```
CURRENT_TIME/ CURTIME( )
```

- **CURRENT_TIMESTAMP/LOCALTIME()/LOCALTIMESTAMP()**: Returns the current time in HH:MI-SS' format. Accepts no arguments

Syntax:

```
LOCALTIME/ LOCALTIMESTAMP( )/NOW()/CURRENT_TIMESTAMP()
```

Calculating Date and Time

- **DATE_ADD/ADDDATE**: Add n no of date interval to the given date.
- **DATE_SUB** : Subtract n no of date interval to the given date.

Syntax:

```
ADDDATE/DATE_ADD(datevalue, INTERVAL n unit )
```

Examples:

```
SELECT ADDDATE/DATE_ADD('2023-10-09', INTERVAL  10 day )
SELECT ADDDATE('2023-10-09', 6);
SELECT DATE_ADD('NOW(), INTERVAL 15 MINUTE);
SELECT ADD_DATE('SYSDATE(), -6 MONTH);
SELECT NOW(), DATE_ADD(NOW(), INTERVAL '1 5'  DAY_HOUR );
```

The type of interval can be one of the following

MICROSECOND
SECOND
MINUTE
HOUR
DAY
WEEK
SECOND_MICROSECOND
MONTH
QUARTER
YEAR
MINUTE_MICROSECOND
MINUTE_SECOND
HOUR_MICROSECOND
HOUR_SECOND
HOUR_MINUTE
DAY_MICROSECOND
DAY_SECOND
DAY_MINUTE
DAY_HOUR
YEAR_MONTH

- **DATEDIFF**: No. of days between two given dates.Accepts two arguments.

Syntax:

```
DATEDIFF(startdate, enddate )
```

Example:

```
SELECT DATEDIFF(CURDATE(), hiredate ) FROM emp;
```

➢ **EXTRACT**: Extract the specified part from the given date.

Accepts only one argument.

Syntax:

```
EXTRACT( part FROM date  )
```

Example:

```
SELECT EXTRACT(DAY  FROM CURDATE()) FROM  dual;
```

➢ **DAYOFMONTH**: Returns the day of the month for a given date (a number from 1 to 31)

Accepts only one argument.

Syntax:

```
DAYOFMONTH( datevalue  )
```

Example:

```
SELECT DAYOFMONTH(CURDATE()) FROM  dual;
```

Other functions include *DAYOFYEAR, DAYOFWEEK, MONTH, YEAR,WEEK.*

➢ **DAYNAME**: Returns the name of the name of the day for a specified date.

Accepts only one argument.

Syntax:

```
DAYNAME(datevalue )
```

Example:

```
SELECT DAYNAME(CURDATE() ) FROM dual;
```

➢ ***STR_TO_DATE():*** Convert a date string into DATE OR DATETIME value.
Accepts two arguments

Syntax:

```
STR_TO_DATE('stringvalue','format_model )
```

Example:

```
SELECT STR_TO_DATE('1947-08-15','%Y-%m-%d' ) FROM dual;
```

➢ ***DATE_FORMAT():*** Convert a date value into a string value in the required format.
Accepts two arguments

Syntax:

```
DATE_FORMAT('datevalue','format_model )
```

Example:

```
SELECT STR_TO_DATE(CURDATE(),'%Y-%m-%d' ) FROM dual;
```

Format models

%a	Weekday name abbreviated (sat)
%b	Month name abbreviated(jan)
%c	Month as numeric value(0 - 12)
%D	Day of the month in ordinal number

%d	Day of the month as numeric value
%H	Hours (0- 23)
%h	Hours(0-12)
%j	Day of the year
%M	Full month name
%m	Month as numeric value
%p	AM or PM

%S	Seconds(0-59)
%U	Week of the year
%W	Weekday name in full
%w	Day of the week
%Y	Year i 4 digits
%y	Year in 2 digits

Table 5.1

➢ ***LAST_DAY () :*** Return the last day of the month for a given date.
Accepts one argument

Syntax:

LAST_DAY(datevalue)

Example:

```
SELECT LAST_DAY(CURDATE()) FROM dual;
```

d) CONVERSION FUNCTIONS:

Conversion functions are used to convert a value from one data type to another. These functions can convert a number or string value to character or date values

Types conversion functions

- **CAST()**: Converts the data type of the given value.

Syntax:

```
CAST(expression AS target_data_type );
```

Example:

```
CAST('123' AS SIGNED )
```

convert the string '123' into an integer 123.

Example:

```
CAST('2023-05-19' AS DATE )
```

convert the string '2023-05-19' into a date value .

Example:

```
CAST( 5000 AS CHAR )
```

convert the string '2023-05-19' into a date value .

- **CONVERT()**: Similar to *CAST()* function but commonly used with character set conversion.

Syntax:

```
CONVERT(expression ,target_data_type );
```

Example:

```
CONVERT('3421' ,SIGNED );
```

Example:

```
CONVERT(865 ,CHAR );
```

Example:

```
CONVERT('hello' USING UTF8 );
```

e) ADVANCED FUNCTIONS

Types advanced functions

- **CURRENT_USER() / USER()**: Returns the username and hostname for the MySQL account that was used by the server to authenticate the current client.

  Accepts no arguments

Syntax:

```
CURRENT_USER( );
```

Example:

```
SELECT CURRENT_USER( );
SELECT USER();
```

- **DATABASE()**: Returns the name of the current database.

  Accepts no arguments

Syntax:

```
DATABASE( );
```

Example:

```
SELECT  DATABASE( );
```

➢ **VERSION()**: Returns the current version of the database.

Accepts no arguments

Syntax:

```
VERSION( );
```

Example:

```
SELECT  VERSION( );
```

➢ **IF()**: Returns one value if a condition evaluates to true or another value if it evaluates to false.

Accepts three arguments

Syntax:

```
IF ( condition, value if true, value if false );
```

Example:

```
SELECT  IF (10<20,'yes','no' );
```

➢ **IFNULL()**: checks whether the given expression is NULL or has a value.
Accepts two arguments.
Returns the second argument if the first argument is NULL else returns the first argument itself

Syntax:

```
IFNULL( argument1,argument2 );
```

Example:

```
SELECT  IFNULL(comm,0 ) FROM emp;
```

➢ **COALESCE()**:Returns the first NOT NULL argument in the given list.

Accepts n no. of arguments.

Syntax:

```
COALESCE( argument1,argument2 …….);
```

Example:

```
SELECT  COALESCE(comm,NULL,SAL ) FROM emp;
```

➢ **CASE** Provide the functionality of IF-THEN-ELSE statement and allows to evaluate conditions and return a value when the first condition is met..

Syntax:

```
CASE expression
        WHEN condition1 THEN result1
        WHEN condition2 THEN result2
      …….
    ELSE  result
END     AS “alias”
```

Example:

```
SELECT ename,job,sal,
        CASE job
             WHEN ‘President’ THEN sal*0.3
             WHEN ‘Manager    THEN sal*0.2
             ELSE  “No tax”
        END     AS “tax”
FROM emp;
```

Exercises:

1. Which of the following statements is true regarding the COUNT function?
    A. COUNT (*) counts duplicate values and NULL values in columns of any data type.
    B. COUNT function cannot work with DATE data types.
    C. COUNT (DISTINCT job_id) returns the number of rows excluding rows containing duplicates and NULL values in the job_id column.
    D. A SELECT statement using the COUNT function with a DISTINCT keyword cannot have a WHERE clause.

2. What will be the outcome of the query given below?
    SELECT 100+NULL+999 FROM dual;

    A. 100
    B. 999
    C. NULL
    D. 1099

3. Which of the following statements are true regarding the single row functions?

    A. They accept only a single argument.
    B. They can be nested only to two levels.
    C. Arguments can only be column values or constants.
    D. They can return a data type value different from the one that is referenced.

4. Out of the below clauses, where can the single-row functions be used?

    A. SELECT
    B. WHERE
    C. ORDER BY
    D . All of the above

5. Examine the structure of the EMPLOYEES table as given.

```
SQL> DESC employees
 Name          Null?           Type
----------------------- --------  ----------------
EMPLOYEE_ID       NOT NULL    INT(6)
ENAME                         VARCHAR(20)
EMAIL                NOT NULL    VARCHAR(25)
PHONE_NUMBER                     VARCHAR(20)
HIRE_DATE            NOT NULL    DATE
JOB_ID               NOT NULL    VARCHAR(10)
SALARY                           DECIMAL(8,2)
COMM                             DECIMAL(2,2)
MANAGER_ID                       INT6)
DEPARTMENT_ID                    INT(4)
```

What will be the outcome of the following query?

```
SELECT last_name, IFNULL(COMM, 'Nocomm')
FROM employees
WHERE last_name LIKE 'A%'
ORDER BY last_name;
```

A. It will throw an error on execution.
B. It will list the job IDs for all employees from EMPLOYEES table.
C. will list the job IDs of all employees and substitute COMM having NULL with a literal 'Nocomm'.
D. It will display the last names for all the employees and their COMM including the NULL values in the COMM.

6. What will be the outcome of the following query?

```
SELECT ROUND(144.23,-1) FROM dual;
```

A. 140
B. 144
C. 150
D. 100

7. You need to display the names of the employees who have the letter 's' in their first name and the letter 't' at the second position in their last name. Which query would give the required output?

A. SELECT first_name, last_name FROM employees
WHERE INSTR(first_name,'s') <> 0 AND SUBSTR(last_name,2,1) = 't';

B. SELECT first_name, last_name FROM employees
WHERE INSTR(first_name,'s') <> '' AND SUBSTR(last_name,2,1) = 't';

C. SELECT first_name, last_name FROM employees
WHERE INSTR(first_name,'e') IS NOT NULL AND SUBSTR(last_name,2,1) = 't';

D. SELECT first_name, last_name FROM employees
WHERE INSTR(first_name,'e') <> 0 AND
SUBSTR(last_name,LENGTH(first_name),1) = 't';

8. What will be the output for the following query?
SELECT UPPER(SUBSTR('computer Science',3,6)

A. MPU
B. MPUT
C. MPUTER
D. PUT

Write queries for the following questions

1. Find the name, job and hire date of the employees who had joined on Tuesday.
2. Find the day on which you are born.
3. Find the names of the employees who have an E as the fourth character in their names.
4. Find the details of the employees who have more than 42 years of experience.
5. Each employee who had joined the company should be confirmed after 240 days from the date of joining. Display the name, hire date and the confirmation date of each employee.
6. Display the experience of each employee in no of days, no of months and no of years expressed in whole numbers.
7. Create a report as per the format given below.
<ename> joined on <hiredate> earns <sal> but wants <3 times sal>
8. Write a query that displays the name and commission amount for each employee. If the employee does not earn a commission display " No Comm".

9. Find the name and hiredate of all employees who had joined in the first half of the year 1981.
10. Display the job code for each employee along with their names , job and salary of all employees.
    The job code will be as follows
    President 'O'
    Manager 'A'
    Analyst 'C'
    Others 'D'

Answers to MCQS

1. **A**
2. **C**
3. **D**
4. **D**
5. **C**
6. **A**
7. **A**
8. **C**

Group functions / Aggregate functions / Multiple row functions

Introduction

Group functions commonly called aggregate functions are used to perform calculations and aggregation on data. Group functions act on a set or group of rows and return one result per set.These sets may comprise the entire table or the table split into groups.

Most of the group functions accept only one argument and cannot be nested.

Group functions automatically ignore NULL values.

The most common group functions are

1. COUNT
2. MIN
3. MAX
4. AVG
5. SUM
6. GREATEST / LEAST

1. COUNT(): Returns the no of rows retrieved by a select statement. Accepts only one argument.

syntax:

```
COUNT(argument);
```

Example:

```
SELECT COUNT(ename) FROM emp;
```

COUNT(*) is different from the count(arg) function. It returns a count of the number of rows retrieved, whether they contain NULL or not.

2) MIN() : Returns the least value within the given set of data from a specified column in a table.. Accepts only one argument.

syntax:

```
MIN(argument);
```

Example:

```
SELECT MIN(SAL) FROM emp;
```

3) Max() : Returns the highest or maximum value within the given set of data from a specified column in a table.. Accepts only one argument.

syntax:

```
MAX(argument);
```

Example:

```
SELECT MAX(SAL) FROM emp;
```

4) AVG() : Returns the mean or average value within the given set of numeric values. Accepts only one argument.

syntax:

```
AVG(argument);
```

Example:

```
SELECT AVG(SAL) FROM emp;
```

2) SUM() : Calculate the total value of a specified numeric column in a table.. Accepts only one argument.

syntax:

```
SUM(argument);
```

Example:

```
SELECT SUM(SAL) FROM emp;
```

2) GREATEST() / LEAST() : Returns the GREATEST / LEAST() value within the given set of data.Accepts N no of arguments.

syntax:

```
GREATEST/ LEAST()(argument1, argument2….);
```

Example:

```
SELECT GREATEST(1000,2000,1500,2500) ;
```

Example:

```
SELECT LEAST(1000,2000,1500,2500);
```

Note: differenceThe between MAX() and GREATEST() is that MAX() accepts only one argument while GREATEST() accepts multiple arguments.

CREATING GROUPS OF DATA

At times you need to divide the table information into smaller groups. For example you need to find the total salary paid to each department. For accomplishing this task first the rows must be grouped according to the departments and then the total salary must be calculated. This can be done by including a GROUP BY clause in the select statement. The group by clause will group similar records together that contain data in the specified column(s) and will allow the aggregate function to be performed.

Syntax:

```
SELECT column_list,group_function(column2) FROM table_name
GROUP BY column_list;
```

Example1: Find the total salary paid to each dept.

```
SELECT  deptno,SUM(sal) FROM emP
GROUP BY deptno;
```

Example 2: Find the no.of employees working in each job category in each department.

```
SELECT deptno,job, COUNT(ename) FROM emp
GROUP BY deptno,job;
```

Note: The columns which are listed in the select list and not included in the group function must be included in the group function.

To restrict the output before grouping use WHERE clause (before GROUP BY clause)

Example : Find the total salary earned by salesmen

```
SELECT job,SUM(sal) FROM emp
WHERE job='SALESMAN'
GROUP BY job;
```

HAVING CLAUSE: The HAVING clause allows you to specify conditions on the rows for each group. In other words it is used to restrict the output after grouping. The HAVING clause should follow the GROUP BY clause. The operation will be as follows.

- ➔ The rows are grouped together.
- ➔ The group function is applied.
- ➔ Groups matching the HAVING clause are selected and displayed.

Syntax:

```
SELECT column_list, group_function(column_name) FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING group_condition
```

Example:

Display the dept.no and maximum salary paid for a department which pays more than 2000 as maximum salary.

```
SELECT deptno,MAX(sal) FROM emp
GROUP BY deptno
HAVING MAX(sal)>2000;
```

STRUCTURE OF A SELECT STATEMENT

SELECT

*** | column_list | expression | functions | expressions**

FROM (clause)

Name/s of the tables from which the records to be retrieved.

WHERE (clause) : used to filter the output before grouping

Column_names | single row functions | expressions and conditions

GROUP BY(clause): Used to group identical records

Column_names listed in SELECT list

HAVING (clause): Used to filter the output after grouping

Group functions and conditions

ORDER BY (clause): Used to sort the output

Column_names | position | column_alais ASC |DESC

LIMIT (clause) : used to read specific number of records

LIMIT(n) n is the no. of columns to be retrieved.

OFFSET (clause) : Used to skip the given number of records

OFFSET(n) n is the no. of columns to be skipped.

Exercises

Choose the correct answer

1. Which of the following is NOT a group funcion
    a. MAX()
    b. MIN()
    c. IFNULL
    d. AVG()

2. Which of the following statements are true about COUNT() function?
    a. The COUNT function counts the number of rows
    b. The COUNT(*) function counts the number of rows with duplicates and NULL values
    c. The COUNT(DISTINCT) function counts the number of distinct rows
    d. COUNT(*) is equivalent to COUNT(ALL)

3. Which of the following SELECT query returns the department number with maximum salary compensated to an employee
    a. SELECT department_id , max(salary) FROM employees ;
    b. SELECT department_id , max(salary) FROM employees GROUP BY department_id ;
    c. SELECT max(salary) FROM employees GROUP BY department_id
    d. SELECT max(salary) FROM employees ;

4. Which clause is used to filter the query output based on aggregated results using a group by function?
    a. WHERE
    b. LIMIT
    c. GROUP WHERE
    d. HAVING

5. Which of the following statements are true about group functions.
    a. The MIN function can be used only with numeric data.
    b. The MAX function can be used only with date values.

c. The AVG function can be used only with numeric data.
d. The SUM function can be part of a nested function.

6. Which of the following is a valid SELECT statement.

a. SELECT AVG(retail-cost) FROM books GROUP BY category;
b. SELECT category, AVG(retail-cost) FROM books;
c. SELECT category, AVG(retail-cost) FROM books WHERE AVG(retail-cost) > 8.56 GROUP BY category;
d. SELECT category, AVG(retail-cost) Profit FROM books GROUP BY category HAVING profit > 8.56;

7. Which of the following SELECT statements list only the book with the largest profit?

a. SELECT title, MAX(retail-cost) FROM books GROUP BY title;
b. SELECT title, MAX(retail-cost) FROM books GROUP BY title HAVING MAX(retail-cost);
c. SELECT title, MAX(retail-cost) FROM books;
d. None of the above.

8. Determine the correct order of execution of the following clauses in a SELECT statement.

1.SELECT
2.FROM
3.WHERE
4.GROUP BY
5.HAVING
6.ORDER BY

a. 2-3-4-5-1-6
b. 1-2-3-4-5-6
c. 6-5-4-3-2-1
d. 5-4-2-3-1-6

9 select the positions inA select query where a group function can appear.

a. SELECT list
b. WHERE clause
c. ORDER BY clause
d. GROUP BY clause.

10. WHERE and HAVING clause can be used interchangeably in a SELECT query

a. Yes
b. No
c. Depend upon RDBMS
d. Not Sure

Write queries to answer the following

1. Display the total number of employees working in the company.
2. Find the total salary and total commission paid.
3. Display the hire date of the senior most employee.
4. Find the average salary paid to salesmen who are working in Deptno 30.
5. display the deptno and total salary paid to each department.
6. Display the number of jobs available in the emp table.
7. Find the deptno and no of employees working in the dept with more than 3 employees.
8. Display various jobs , deptnos and average salaries of each job in deptno 20 which pays an average salary more than an average salary of 1500. Order your result in alphabetical order of job.

Answers

1. c
2. b
3. b
4. c
5. c
6. a
7. a
8. a
9. a, c
10. b

JOINS

Introduction

Tables in MySQL are often related to each other where the relationship between the tables can be established by the columns which share the same data set. To derive data from these tables use JOIN clause. The JOIN clause provides a structure to connect data from multiple tables by specifying the tables to be joined and under what conditions they are joined. The common columns must have the same data type and the data should match. The names of these columns can be different.

MySQL provides three basic types of Joins

a. Cross Join
b. Inner Join
c. Outer Join

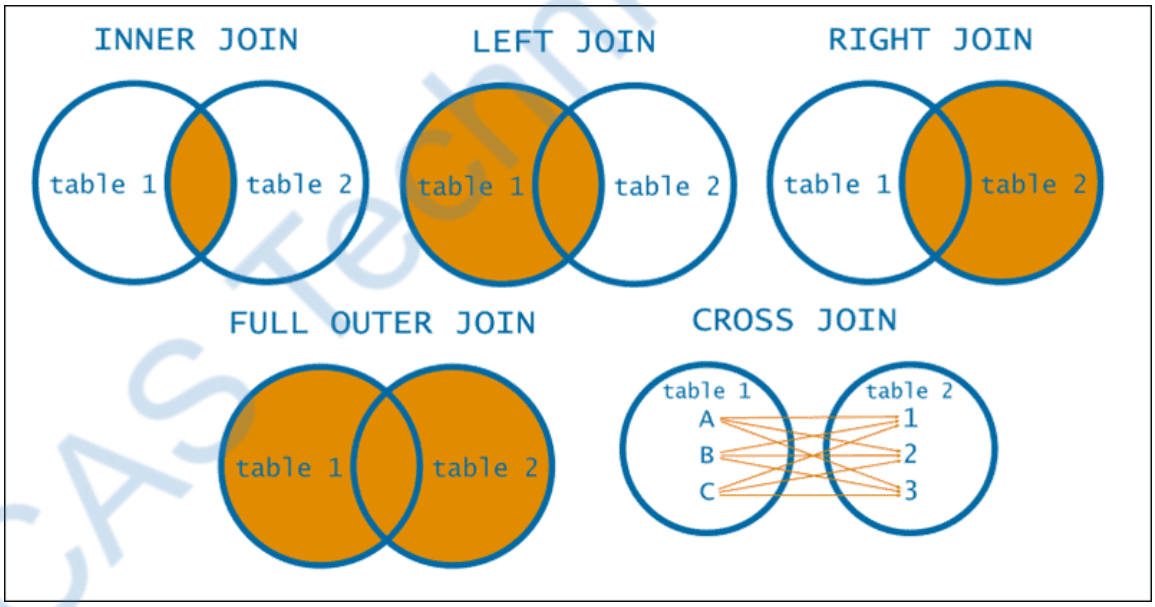


Figure 9:1

a. CROSS JOIN

It is a type of join which produces a **CARTESIAN PRODUCT** of the tables where each row of one table will be joined or matched with every row of the other table. The result-set of a CROSS join is the product of the number

of rows of the joined tables. The CROSS join will be formed when the JOIN condition is omitted or invalid.

If you have a products table with 2 rows and a categories table with 3 rows, a CROSS JOIN will result in a table with 2 * 3 = 6 rows.

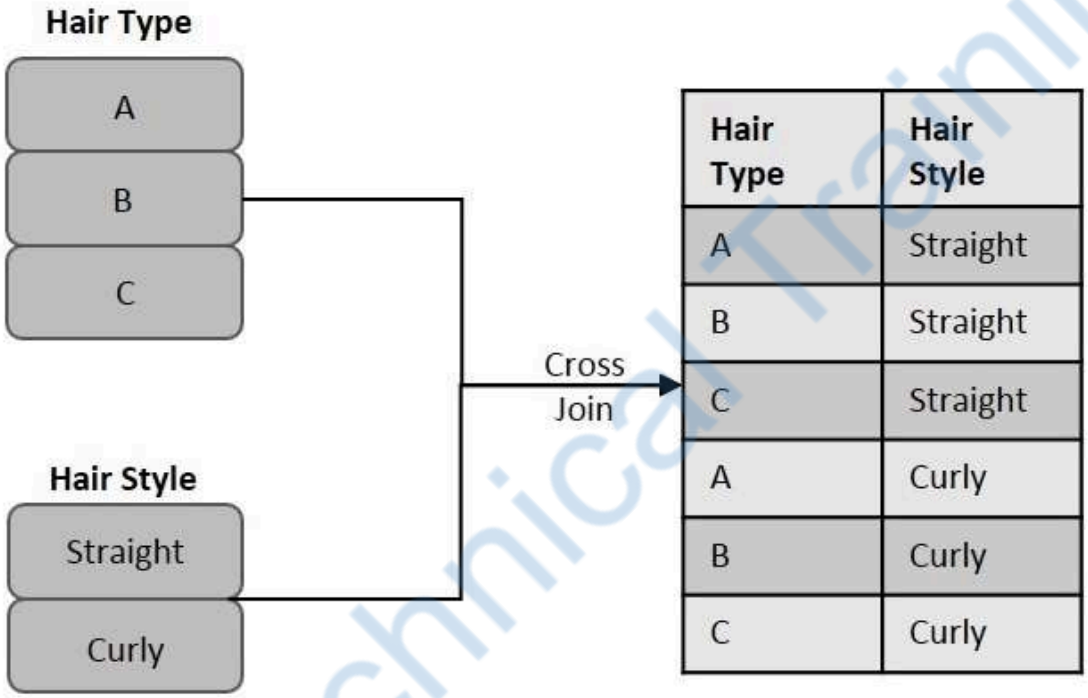


Figure 9.2

Syntax

SELECT column_list FROM table1 CROSS JOIN table2;

Or

SELECT column_lsit FROM table1 , table2;

Example

SELECT ename,dname,loc FROM emp CROSS JOIN dept;

Or

SELECT ename,dname,loc FROM emp , dept;

b. INNER JOIN

The INNER JOIN in MySQL retrieves rows from two or more tables based on a common column. It returns rows that match the specified match condition. The condition is specified in the ON clause.

Employee Table

Empno	Ename	Deptno
1	smith	10
2	mark	10
3	allen	20
4	kayl	30

Inner Join

ename	location
smith	Berlin
mark	Berlin
allen	Munich

Deptno	Dep Name	Location
10	Accounts	Berlin
20	HR	Munich
40	IT	Frankfurt
50	Sales	Freiburg

Department Table

Figure 9:3

It is similar to the intersection of the sets in Mathematics. i.e. when you take the intersection of two or more sets only the common element (in all the sets) are taken together.

Syntax

```
SELECT column_list FROM table1 INNER JOIN table2;
ON (table1.columnName=table2.columnName;
```

Example

```
SELECT ename,dname,loc FROM emp INNER JOIN dept
ON (emp.deptno=dept.deptno);
```

Ambiguous columns

When a certain column is called which is available in both the tables and SQL has no idea from which table the data for that column should be read. The server raises an ambiguous error. To avoid this error the column names must be qualified with the table name or table alias. It is a best practice to qualify all the columns (whether ambiguous or not) with table name or alias name.

Syntax

```
SELECT table1.columnName,table2.columnName…. FROM table1 INNER
JOIN table2;
ON (table1.columnName=table2.columnName;
```

Example

```
SELECT emp.ename,dept.dname,dept.loc FROM emp INNER JOIN dept
ON (emp.deptno=dept.deptno);
```

example

```
SELECT e.ename,d.dname,d.loc FROM emp e  INNER JOIN dept d
ON (e.deptno=d.deptno);
```

Subsets of *INNER JOIN*

i) ***EQUI JOIN***

It is a subset of *INNER JOIN* that combines rows from two or more tables based on a condition which uses the equality (=) operator. The join condition is normally specified in the where clause.

Syntax

```
SELECT column_list FROM table1 ,table2;
WHERE table1.columnName=table2.columnName;
```

Example

```
SELECT ename,dname,loc FROM emp  JOIN dept
ON (emp.deptno=dept.deptno);
```

II) ***NATURAL JOIN***

Natural join combines rows from two or more tables based on the common columns between the tables. It automatically matches the columns with the same name and data types. It does not need an explicit *JOIN* condition.

Syntax

```
SELECT column_list FROM table1 NATURAL  JOIN table2;
```

Example

```
SELECT ename,dname,loc FROM emp NATURAL JOIN dept;
```

III) ***SELF JOIN***

It is a type of join where a table is joined with itself which is used to compare or join data within the same table. A single table is virtually considered as two separate tables by using aliases and relationships are established using rows. To use a self join, a table must have a unique identifier column, a parent column, and a child column.

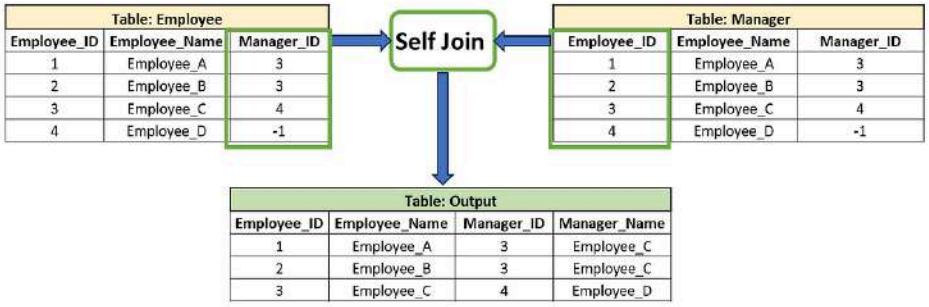


Figure 9:4

Syntax

SELECT tablealias1.columnName,tablealias2.columnName…. FROM tablealias1 INNER JOIN table alias;
ON (tablealias1.columnName=tablealias2.columnName;

Example

SELECT e.ename as worker m.ename as manager
FROM emp e INNER JOIN emp m
ON (e.mgr=m.empno);

EQUI JOIN vs. INNER JOIN vs NATURAL JOIN

INNER JOIN	EQUI JOIN	NATURAL JOIN
Combines rows from two tables based on the specified condition	Tables are joined using the equality operator (=).	Automatically matches columns with the same name in both tables.
Explicit condition specified using *ON* clause	used with *ON* or *WHERE* clauses.	Automatically joins columns with the same name in both tables.
Retains all columns from both tables, even if they have the same name.	Retains all columns from both tables, even if they have the same name.	Removes duplicate columns that have the same name. Only one copy of the column is kept.

USING CLAUSE

The USING clause is another SQL feature used for joining tables, particularly when dealing with identical-named columns in both tables. Use the USING clause to specify the columns for the equi join where several columns have the same names but not the same data types. Use the USING clause to match only one column when more than one column matches.

Syntax

```
SELECT table1.column, table2.column
FROM table1 JOIN table2
ON  (join_column);
```

Example

```
SELECT e.ename,d.dname,deptno
 FROM emp e  JOIN dept d
USING  (deptno);
```

ON clause vs. USING clause

ON clause	USING clause
More flexible	Simpler
Can specify any condition (equality, inequality, range, etc.) using complex logic.	Automatically assumes equality (=) between columns with the same name.
Can join columns with different names.	Columns in both tables must have the same name.

c. OUTER JOIN

Outer join in MySQL is used to retrieve records from two or more tables including the records that do not have matching values. It retrieves all the records from one table usually called as OUTER table and only the matching values

from the other table. When no matches are found *NULL* will be filled in the columns of the respective tables.

Types of OUTER joins

There are two main types of outer joins in MySQL

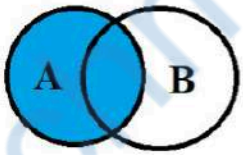
i) LEFT OUTER JOIN or LEFT JOIN

A left join allows to retrieve all the rows from the left side table and only the matching rows from the right side table. For the records in the left side table that do not match the condition *NULL* values are displayed.

Student ID	Name
1001	A
1002	B
1003	C
1004	D

+

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science



Student ID	Name	Department
1001	A	NULL
1002	B	NULL
1003	C	NULL
1004	D	Mathematics

Figure 9:5

Syntax

```
SELECT table1.column, table2.column
FROM table1  LEFT OUTER JOIN table2
ON  (tablealias1.columnName=tablealias2.columnName);
```

Example

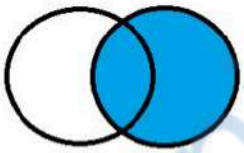
```
SELECT e.ename,d.dname,d.deptno
 FROM emp e  LEFT OUTER JOIN dept d
ON   (e.deptno = d.deptno);
```

i) RIGHT OUTER JOIN or RIGHT JOIN

A right join allows to retrieve all the rows from the right side table and only the matching rows from the left side table. For the records in the right side table that do not match the condition *NULL* values are displayed.

Student ID	Name
1001	A
1002	B
1003	C
1004	D

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science



Student ID	Name	Department
1004	D	Mathematics
1005	NULL	Mathematics
1006	NULL	History
1007	NULL	Physics
1008	NULL	Computer Science

Figure 9.6

Syntax

```
SELECT table1.column, table2.column
FROM table1  RIGHT OUTER JOIN table2
ON  (tablealias1.columnName=tablealias2.columnName);
```

Example

```
SELECT e.ename,d.dname,d.deptno
 FROM emp e  RIGHT OUTER JOIN dept d
ON   (e.deptno = d.deptno);
```

Note : The RIGHT OUTER JOIN and LEFT OUTER JOIN can be combined with the UNION operator in MySQL to get the results similar to FULL OUTER JOIN which is available in Oracle and SQL sever.

Syntax

```
SELECT table1.column, table2.column
FROM table1  RIGHT OUTER JOIN table2
ON  (tablealias1.columnName=tablealias2.columnName)
UNION
SELECT table1.column, table2.column
FROM table1  LEFT OUTER JOIN table2
ON  (tablealias1.columnName=tablealias2.columnName)
```

Example

```
SELECT e.ename,d.dname,deptno
 FROM emp e  RIGHT OUTER JOIN dept d
ON   (e.deptno = d.deptno)
UNION
SELECT e.ename,d.dname,deptno
 FROM emp e  LEFT OUTER JOIN dept d
ON   (e.deptno = d.deptno);
```

Exercises

Choose the correct answer.

1. What is an INNER JOIN in MySQL.

    a. Returns all rows from both tables, with matching rows from both sides
    b. Returns all rows from the first table and matching rows from the second table
    c. Returns only matching rows from both tables
    d. Returns all rows from the second table

2. Which of the following join types returns all rows from the left side table and matching rows from the right side table.

    a. FULL JOIN
    b. OUTER JOIN
    c. RIGHT OUTER JOIN
    d. LEFT OUTER JOIN

3. Table T1 has 10 rows and 5 attributes. Table T2 has 0 rows and 7 attributes. When a CROSS JOIN is achieved between R1 and R2, how many tuples would the resultant set have?

    a. 28
    b. 10
    c. 0
    d. 35

4. Which join is to be used between two tables A and B when the resultant table needs rows from A and B that matches the condition and rows from A that does not match the condition?

a. Outer join
b. Cross join
c. Natural join
d. Equi join

5. Which Join is equivalent to cartesian product
a. Natural join
b. Cross join
c. Self join
d. Outer join

6. Which clause is used to specify a NATURAL JOIN
a. ON clause
b. WHERE clause
c. USING clause
d. No need to specify a clause.

Write queries to find the output for the following questions

1) Write a query to display the name, job, salary, department name and the department number of all the employees.
2) Display the name, hiredate, salary, department name and the department number of all the employees who work in New York.
3) Your next task is to find the manager names of each employee. Write a query to display the employee id, employee name, manager id and the manager name of all the employees.
4) Modify the above query to display the name, job, salary, department name and the department number of all the employees. The query should include the employee 'KING' also who has no manager.
5) Display the name, job, salary and the salary grade of each employee.
6) Find the department name available in chicago along with the number of

employees in that department.

7) display the name, job salary , department name and salary grade of all employees.

ANSWERS

1. C
2. D
3. C
4. A
5. B
6. D

MySQL subqueries

Using Subqueries to solve problems

A MySQL subquery is a query nested within another query such as *SELECT, INSERT, UPDATE, DELETE*. A subquery can be nested within another sub query. It allows to perform operations in multiple steps making complex queries easily understandable and manageable. Sub-queries are mainly used to find the unknown values. The subquery can return values from another table.

Guidelines for using subqueries

Enclose the subquery within parenthesis.

Place subqueries on the right hand side of the comparison operator.

Use appropriate operators to filter the output based on the type of subquery.

Parts of a sub query

A subquery typically consists of two parts. The first is the main query or outer query, which gives the actual output. The inner query or subquery is the query nested within the main main query, which is used to find the unknown values.

Working of a subquery

The subquery will be executed independently and will return a result. Depending upon the type of subquery, the inner query may return a single value (row) or multiple rows. This result is then passed to the main query to produce the final output.

Where subqueries are used?

- ➔ *WHERE* clause
- ➔ *FROM* clause
- ➔ *SELECT* clause
- ➔ *HAVING* clause

Types of subqueries

a. Single row subquery
b. Multiple row subquery
c. Multiple column subquery
d. Correlated subquery

Single row subquery

A single row subquery is a subquery, which will return only one row and one column. Use single row operators (=, >, >=, <, <= and !=) with this type of subquery.

Syntax:

```
SELECT select_list FROM table_name

WHERE condition (SELECT column_name FROM table_name
WHERE condition);
```

Examples:

Find the employees who earn more than Allen.

```
SELECT *  FROM emp

WHERE sal > (SELECT sal FROM emp WHERE ename='Allen');
```

Find the employees who work with the same designation as Jones and joined before Scott

```
SELECT *  FROM emp

WHERE job > (SELECT job FROM emp WHERE ename='Jones'

AND hiredate > (SELECT hiredate FROM emp

                                    WHERE ename='Scott');
```

Find the Highest paid employee

```
SELECT *  FROM emp

WHERE sal > (SELECT MAX(sal) FROM emp);
```

Note: Single row subqueries can return rows from different tables.

Find the employees working in new york

```
SELECT *  FROM emp

WHERE deptno = (SELECT deptno FROM dept where loc='New York');
```

If the inner query returns *NULL* value then the outer query will not return any value.

Multiple row subqueries

Multiple row subquery is a query which returns more than one row. Multiple row operators are used to handle such subqueries.

Multiple row operators

IN **:** compare the values with the set of values returned by the subquery.

NOT IN: Compare the values with other than the set of values returned by the subquery.

>ANY: Compare the values more than the lowest value returned by the subquery

<ANY: Compare the values less than the highest value returned by the subquery

>ALL: Compare the values more than the highest value returned by the subquery

<ALL: Compare the values less than the lowest value returned by the subquery

Syntax:

```
SELECT select_list FROM table_name

WHERE expression multiple row operator

(SELECT column_name FROM table_name WHERE condition);
```

Examples:

Find the employees who are working in Chicago and Dallas

```
SELECT *  FROM emp

WHERE deptno  IN

(SELECT deptno FROM dept WHERE  loc IN ('Chicago','Dallas');
```

Find the employees who are earn less than the highest salary earned by Analysts

```
SELECT *  FROM emp

WHERE SAL  <ANY

(SELECT SAL FROM EMP WHERE  JOB= 'Analyst';
```

Multiple column subqueries

Multiple column subquery is a query which returns more than one column and compared with multiple columns in the outer query.

Syntax:

```
SELECT select_list FROM table_name

WHERE column1_name,column2_name IN

 (SELECT column1_name,column2_name FROM table_name
WHERE condition);
```

Examples:

Find the employees who earn the highest salary in their respective departments and job category.

```
SELECT *  FROM emp

WHERE job,sal  IN

(SELECT job,MAX(sal)  FROM emp group by job);
```

Find the employees who work with the same designation as CLARK and report to the same Manager.

```
SELECT *  FROM emp

WHERE job,mgr  IN

(SELECT job,mgr FROM emp);
```

Correlated subqueries

Correlated subquery is a subquery which depends on the outer query and executed every time when a row in the outer query row is processed. It references the columns from the outer query. Correlated subqueries are less efficient that the normal queries.

Syntax:

```
SELECT column_names

FROM outer_table AS outer

WHERE columnName IN (SELECT column_name

                     FROM inner_table AS inner

                     WHERE outer.columnName = inner.columnName);
```

Find the employees who earn more than the average salary in their department.

```
SELECT empno, ename, sal

FROM emp  e

WHERE sal > (SELECT AVG(sal)

             FROM emp

             WHERE deptno = e.deptno);
```

Exercises

Choose the correct answer

1. Which of the following is true about sub-queries?
    a. They execute after the main query executes
    b. They execute in parallel to the main query
    c. The user can execute the main query and then, if wanted, execute the sub-query
    d. They execute before the main query executes.

2. In which of the following clauses can a sub-query be used?
    a. HAVING
    b. WHERE
    c. FROM
    d. All of the above

3. What will be the outcome of the following query

```
SELECT first_name, last_name, salary
FROM employees
WHERE salary ANY (SELECT salary FROM employees);
```

    a. It executes successfully giving the desired results
    b. It executes successfully but does not give the desired results
    c. It throws an ORA error
    d. It executes successfully and gives two values for each row obtained in the result set

4. Which of the following operators cannot be used in a sub-query?
    a. AND
    b. =
    c. >ANY
    d. IN

5. You need to find the salaries for all the employees who have a higher salary than the Vice President of a company 'ABC'.Which of the following queries will give you the required result?

a. SELECT first_name, last_name, salary
FROM employees
WHERE salary > (SELECT salary
FROM employees
WHERE job_id = 'VICE-PRESIDENT');

b. SELECT first_name, last_name, salary
FROM employees
WHERE salary = (SELECT salary
FROM employees
WHERE job_id = 'VICE-PRESIDENT');

c. SELECT first_name, last_name, salary
FROM employees
WHERE job_id = 'VICE-PRESIDENT');

d. None of the above

6. Which of the following queries will list out the details of all employees earning less than the average salary.

a. SELECT * FROM emp WHERE Salary < AVG(Salary)
b. SELECT * FROM emp WHERE Salary <
(SELECT Salary/COUNT(*) FROM emp)
c. SELECT * FROM emp WHERE SALARY <
(SELECT AVERAGE(Salary) FROM emp)
d. SELECT * FROM emp WHERE Salary <
(SELECT AVG(Salary) FROM emp)

7. In a correlated subquery, how does the inner query reference the outer query?

a. Using a JOIN condition
b. By using a column from the outer query in the subquery's condition
c. By using an alias for the outer query
d. By using a WHERE clause with a fixed value

8. Which of the following are not allowed in a subquery

a. Using *SELECT* to read records.
b. Using *GROUP BY* in a subquery.
c. Modifying the outer query with *UPDATE* in a subquery
d. Using ORDER BY in a subquery

Write queries to solve the problems

1. Write a query to find the details of the employees.

2. Find the details of the senior most employee.

3. Find the details of the employees who are reporting to smith.

4. Write a query to find the details of the employees who earn more than the least paid employee in deptno 20.

5. List the jobs having the minimum salary exceeds the average salary of all employees.

6. Find the department numbers , which is having more than 3 employees.

7. Find the deptno and total salary paid for all employees working in Chicago.

8. Find the details of the employees who work in a department with any employee joined in the year 1981.

9. Find the employees working in Dallas.

10. Write a query to find the name, deptno and salary of all employees whose salary is above the average salary in their respective department numbers. Arrange your output in descending order of deptno.

Answers

1. D
2. D
3. C
4. A
5. A
6. D
7. B
8. C

2. Database Development Life Cycle(DDLC).

Introduction

If you were to build a new house, you certainly wouldn't want to start with putting the walls up first without being prepared. You would want to talk to an architect to determine how big the house will be, how much it will cost, who the builders will be, how long it will take etc. And once this process is complete you could use your blueprint to begin construction on the house. The same goes for building a database.

A well-designed database is essential to guarantee information consistency, eliminate redundant data, efficiently execute queries, and improve the database's performance. A methodological approach towards a designed database will save you time and the database development phase.

Database Development Life Cycle

It is a process used to design, develop, implement, and maintain a database system. It involves a series of stages that help to ensure the successful creation of a database that meets the needs of the organization and its users.

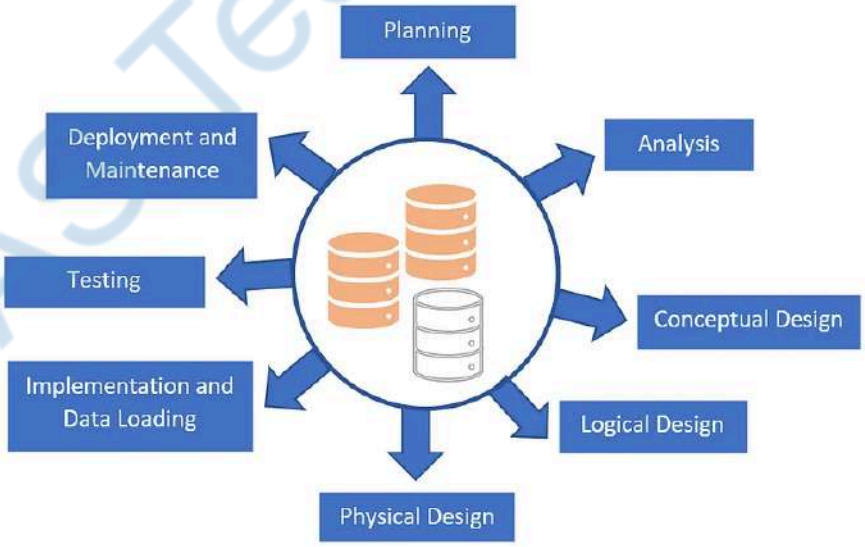


Figure 2.1

a. **Planning**

This is the first step in the design process which sets the foundation for the whole project. The need for a database, its goal , cost estimation and feasibility are analyzed and the objectives are documented clearly.

b. **Requirement analysis**

The next step is to collect the requirements for the proposed database. This includes determination of the types of users, security requirements, performance expectations, and data integrity rules.

c. **Conceptual design**

In this step, the requirement specifications gathered in the previous stage are translated into a conceptual model. The data is converted into a graphical representation which could be used to convert into logical design.

d. **Logical design**

In the logical design phase, the focus is on translating the conceptual model (ER diagram) into a logical model that can be implemented in a specific database management system (DBMS). It involves Converting the E.R into tables, constraints and relationships. Then normalize to ensure data is efficiently structured and there is no redundancy**.**

e. **Physical design**

This phase involves translating the logical design into a physical structure that will be implemented in the actual database system. The focus is on performance, storage, and optimization based on the requirements and the DBMS being used.

f. **Implementation**

This is the actual construction of the database. It involves creating the database structure (tables, indexes, relationships) and populating it with data.Set up the database in the chosen DBMS, implementing tables, relationships, and constraints.

g. **Testing**

The testing phase is critical to ensuring that the database performs as expected, meets the requirements, and is free from errors.

h. **Deployment and maintenance.**

Once the database has been tested and validated, it is ready for deployment in a production environment. After deployment, regular maintenance is required to keep the database running efficiently.

Database Designing techniques

The database design process is essential to creating good and useful information. These technologies help ensure that data is effectively supported by data storage, retrieval, and management, while maintaining data integrity and reducing duplication.

The two most common techniques used to design a database include:

1. ER Modelling
2. Normalization

Entity-Relationship (ER) Modeling: A graphical database design approach models entities, attributes, and defines relationships among them to signify real-life objects. An entity is any real-world item that is different or unique from the surroundings.

Building the Relational Model using ER Modeling

Creating a relational model (ER) is an important step in database design and involves defining and visualizing the database model. ER models help understand how data is related to each other and form the basis for building a good database system.

There are three fundamental components of an ER model:

- Entities
- Attributes
- Relationships

Entities

The types of information that are saved in the database are called 'entities'.

An entity is a real world object. Each entity is a specific, distinct kind of thing, about which you need to track information.These entities exist in four kinds: people, things, events, and locations. Everything you could want to put in a database fits into one of these categories. These Entities will be considered as tables in the database.

For example, if you want to create a database system for a motorcycle company, you might need to track information about motorcycles, customers, and sales. Now we have three kinds of things, each with its own set of characteristics.

Example:

Figure 2.2

Entity set :

An entity set is a group of entities of the same entity type.

Figure 2.3

In DBMS a strong entity set consists of a Primary key whereas a weak entity set does not contain a primary key.

Types of entities

a. Strong Entity type:

It is an entity that has its own existence and is independent. In the above example customers, products and suppliers are strong entity types.

b. Weak Entity type:

It is an entity that does not have its own existence and relies on a strong entity for its existence. For example the address of a customer

Attributes

An attribute is a property or characteristic of an entity. It describes various aspects of an entity.These Attributes are going to be transformed into fields or columns in the database. In other words, attributes represent the data we want to store in the database.

Example

Each attribute holds specific information about a motorcycle, customers and sale information making it easier to manage and retrieve the data.

Figure 2.4

Types of attributes

a. Simple attribute:

It is also known as atomic attributes which means they hold only single values (atomic values) that cannot be broken down further.They are commonly used for basic SQL data types such as integers, strings, and dates.

Example: customer ID

b. Composite attribute:

An attribute that can be split into components is a composite attribute. It is made up of multiple simple attributes, each representing a more detailed aspect of the overall attribute.

Example : Address of the customer. It can be further split into House Number, Street name, city , district etc..

c. Single-valued attribute.

The attribute, which takes up only a single value for each entity instance, is a single-valued attribute.

Example : The age of a customer.

d. Multi-valued attribute.

The attribute, which takes up more than a single value for each entity instance, is a multi-valued attribute.

Example : The phone number of a customer.

e. Derived attribute.

The attribute that can be derived from other attributes is called a Derived attribute. Derived attributes are not stored directly in the database but are computed or derived from other stored attributes.

Example : The total purchase done on a day by a customer.

f. Key attribute.

It is the attribute, which is used to identify the entity in an entity set. It ensures that each instance of an entity can be uniquely identified within the database.

Example: The ID of a customer.

Attribute Domains

Each attribute has a **domain**, which defines the set of possible values that the attribute can take. For example, the domain of the "Age" attribute could be all integers from 0 to 120, or the domain of the "Date of Birth" attribute could be all valid dates.

Identify the relationships

The next step is to determine the relationships between the entities and to determine the cardinality of each relationship. The relationship is the connection between the entities, just like in the real world: what does one entity do with the other, how do they relate to each other? For example, customers buy products, products are sold to customers, a sale comprises products, a sale happens in a shop. The cardinality shows how much of one side of the relationship belongs to how much of the other side of the relationship. Relationships allow the datasets to share and store data in separate tables

Cardinality of relationship

Cardinality of relationship refers to the number of instances (records) of one entity that can be associated with instances of another entity in a relationship. It defines the **quantity** of records that can or must exist in one table in relation to records in another table. It defines how many records from one table can be linked to records in another table.

Basically,there are three types of relationships.

1. **One to One Relationship**

   Such a relationship exists when each record of one table is related to Only one record of the other table.

Example:

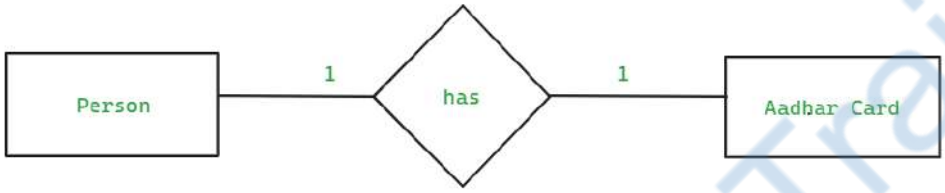


Figure 2.5

Consider two entities: "Person" and "Aadhar card". Each person can have only one Aadhar card and each Aadhar card is assigned to only one person.

One to Many Relationships

Such a relationship exists when each record of one table can be related to one or more than one record of the other table. This relationship is the most common relationship found.

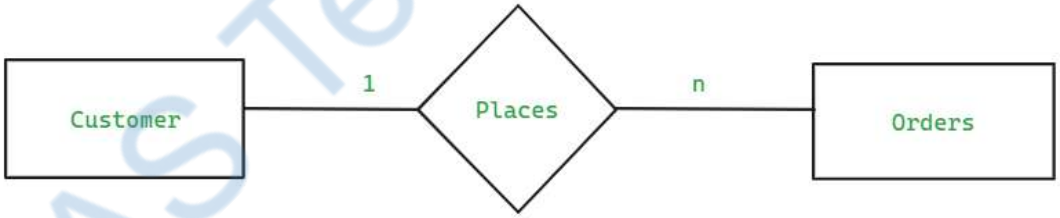


Figure 2.6

Many to Many Relationships

Such a relationship exists when each record of the first table can be related to one or more than one record of the second table and a single record of the second table can be related to one or more than one record of the first

table. A many-to-many relationship can be seen as a two one-to-many relationship which is linked by a 'linking table' or 'associate table'.

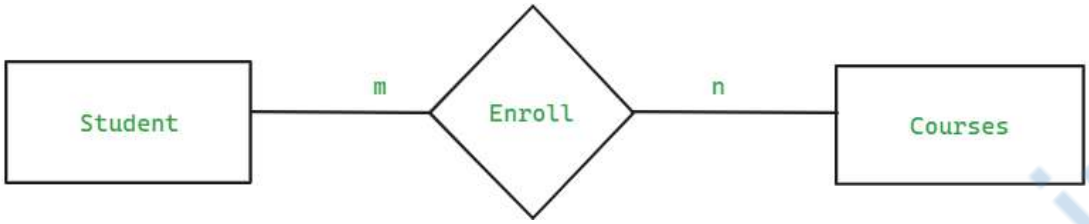


Figure 2.7

In a "Student" and "Course" relation, where each student can enroll in multiple courses and each course can have multiple students enrolled in it.

The E- R diagram

Entity relationship model is the next step in the design of a database.
An Entity Relationship Diagram (ER Diagram) pictorially explains the relationship between entities to be stored in a database. Fundamentally, the ER Diagram is a structural design of the database. It acts as a framework created with specialized symbols for the purpose of defining the relationship between the database entities. ER diagram is created based on three principal components: entities, attributes, and relationships.

Symbols Used in ER Diagrams

- Rectangles: This Entity Relationship Diagram symbol represents entity types
- Ellipses: This symbol represents attributes
- Diamonds: This symbol represents relationship types
- Lines: It links attributes to entity types and entity types with other relationship types
- Primary key: Here, it underlines the attributes
- Double Ellipses: Represents multi-valued attributes

Symbols used in ER Diagram

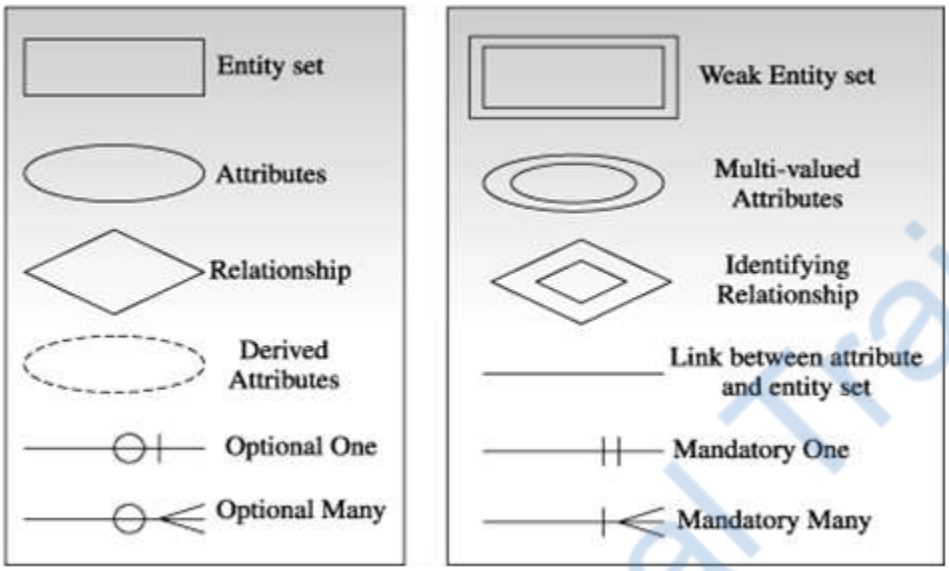


Figure 2.8

Drawing the ER Diagram.

Step 1: Identify the Entities

This is the first step in creating an ER diagram. Entities are represented by RECTANGLES.

Example

a. Student
b. Course
c. Department

The above entities can be used to store the information for the university database.

Step 2: Identify the Relationships.

Relationships show how entities are connected to each other. For example, a Student can be enrolled in a Course.

Draw each relationship as a diamond. Connect the entities to the relationship diamond using lines.

Step 3: Define the attributes.

Attributes describe the properties or details of entities or relationships. For example, a Student entity may have attributes like Student_ID, Name, and Email.

Draw attributes as ovals and connect them to their corresponding entity or relationship. Label the attributes clearly.

Tools to draw the ER diagram

Many online tools are available to draw an ER diagram and many notations are also available. The most commonly used notations are

a. Chen's notation.
b. Crow's foot notation.

Chen's notation.

In a Chen notation diagram, entities are represented by rectangles with the entity name written inside. Relationships are represented by a triangle. Two entities can be connected with a line between them and their respective cardinalities written where the line connects each entity.

Participation constraints

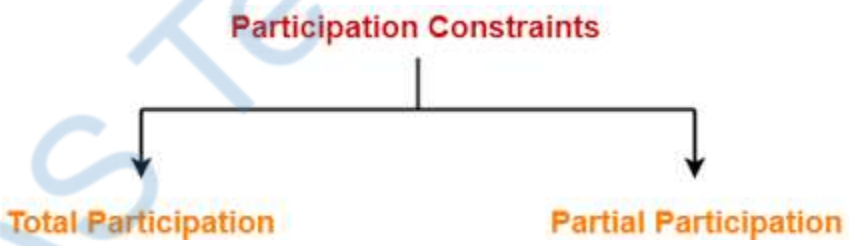


Figure 2.9

The participation constraint specifies the number of instances an entity can participate in a relationship set.

There are two main types of participation constraints:

1. **Total Participation**
2. **Partial Participation**

1. Total Participation

It specifies that each entity present in the entity set must mandatorily participate in at least one relationship instance of that relationship set,for this reason, it is also called as mandatory participation

It is represented using a double line between the entity set and relationship set

Example

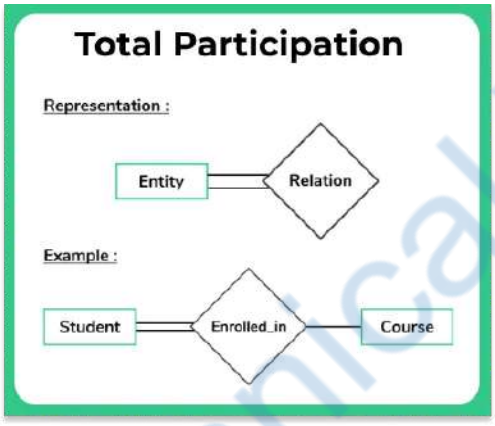


Figure 2.9

- It specifies that each student must be enrolled in at least one course where the “student” is the entity set and relationship “enrolled in” signifies total participation
- It means that every student must have enrolled at least in one course

2. Partial Participation

It specifies that each entity in the entity set may or may not participate in the relationship instance of the relationship set, and is also called as optional participation. It is represented using a single line between the entity set and relationship set in the ER diagram

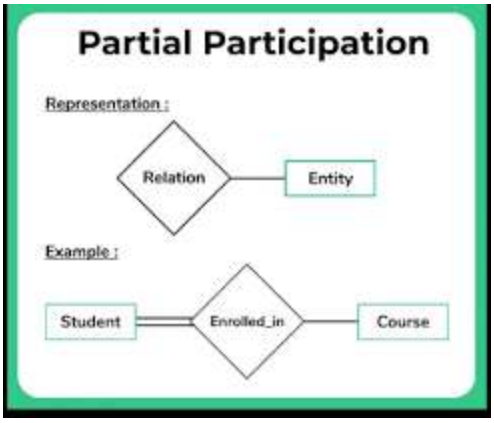


Figure 2.10

- It specifies that a course might exist in the system with or without a student enrolled in it.

Cardinality Constraints

Cardinality means “HOW MANY”.. It specifies the number of instances of one entity that can be associated with an instance of another entity in a relationship. In other words, cardinality defines how many instances of one entity can or must be associated with a single instance of another entity in a relationship.

Types of Cardinality Constraints

There are four main types of cardinality constraints that define the relationship between two entities:

1. One-to-One (1:1) : Single instance of an entity is associated with a single instance of another entity.

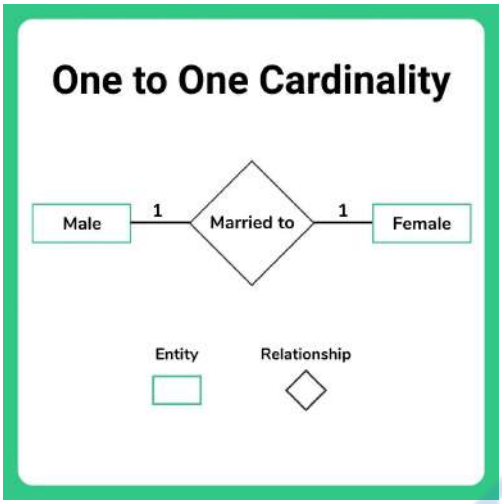


Figure 2.11

2. One-to-Many (1:N) : Single instance of an entity is associated with more than one instance of another entity

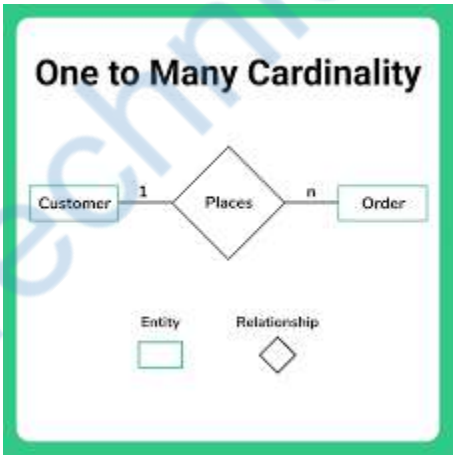


Figure 2.12

3. Many-to-One (N:1)

   Many instances of one entity are associated with one instance of another entity.

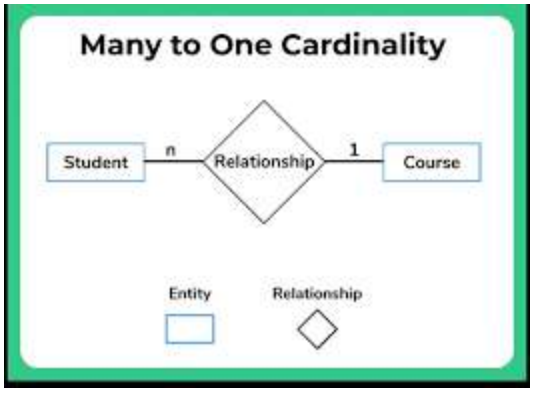


Figure 2.12

4. Many-to-Many (N:M)

Many instances of one entity are associated with many instances of another entity.

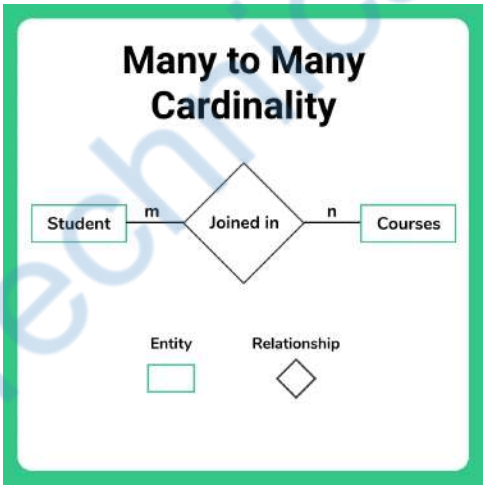


Figure 2.13

Use Case

Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.

Identify the Entities:

1. Patient
2. Doctor (Medical Doctor)
3. Test

Identify the Relationships:

Patient - Doctor Relationship:

A Patient is associated with one or more Doctors. This could be an "assigned to" relationship, indicating that a doctor is responsible for the patient's care.
A Doctor can have many patients, so this is a one-to-many relationship.

Patient - Test Relationship:

A Patient undergoes multiple Tests. Each test might be logged with details like the test name, result, and date.
This is a one-to-many relationship from Patient to Test/Examination, as each patient can have multiple tests, but each test is linked to one patient.

Define the Attributes for each Entity:

- **Patient:**

  Patient_ID (Primary Key)
  Name
  Date_of_Birth
  Address
  Phone_number

- **Doctor:**

  Doctor_ID (Primary Key)
  Name
  Specialization
  Phone_number

- **Test:**

  Test_ID (Primary Key)
  Test_Name
  Test_Date
  Result

Define the cardinality of relationship:

Patient - Doctor Relationship: A Patient is assigned to one or more Doctors, and each Doctor can have multiple Patients.

Cardinality: Many-to-Many (M:N)

Patient - Test/Examination Relationship: A Patient can have many Tests/Examinations, but each Test/Examination is linked to a single Patient.

Cardinality: One-to-Many (1:M)

Draw the ER Diagram:

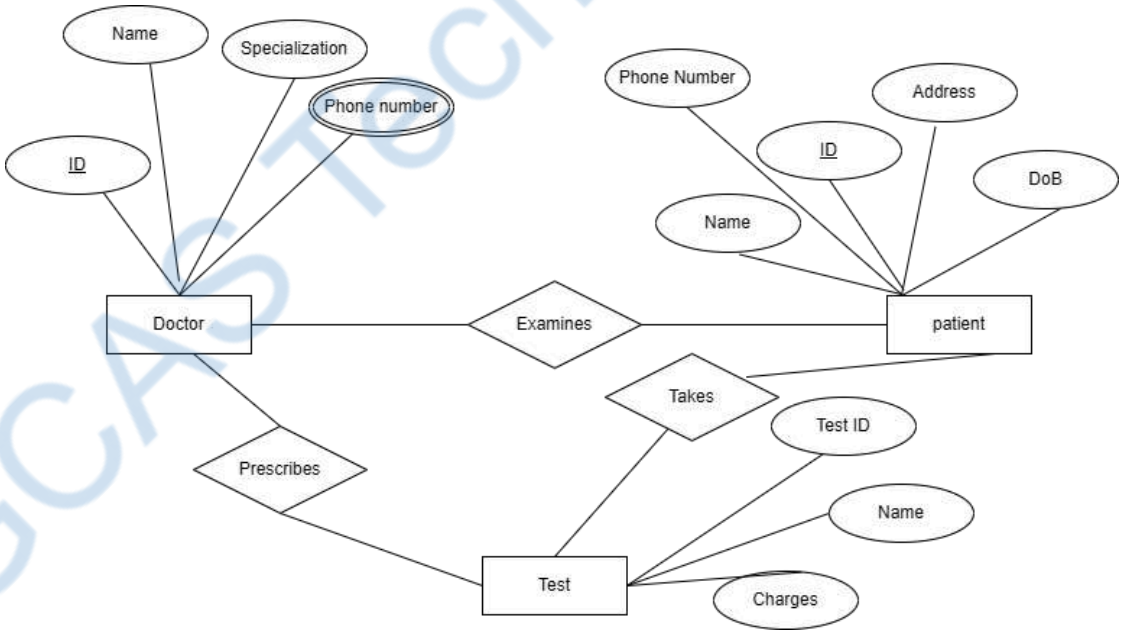


Figure 2.14

Crow's foot Notation

It is a popular method to draw ER diagrams to represent the relationships between the Entities. Here entities are represented by rectangles with its name on the top.

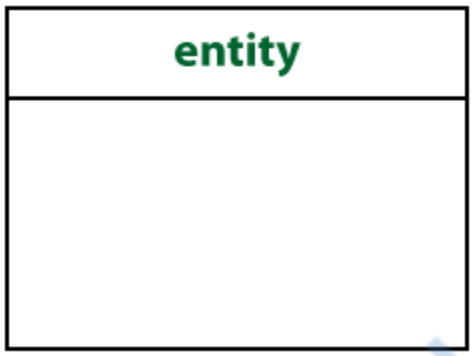


Figure 2.15

Attributes are listed below the entity

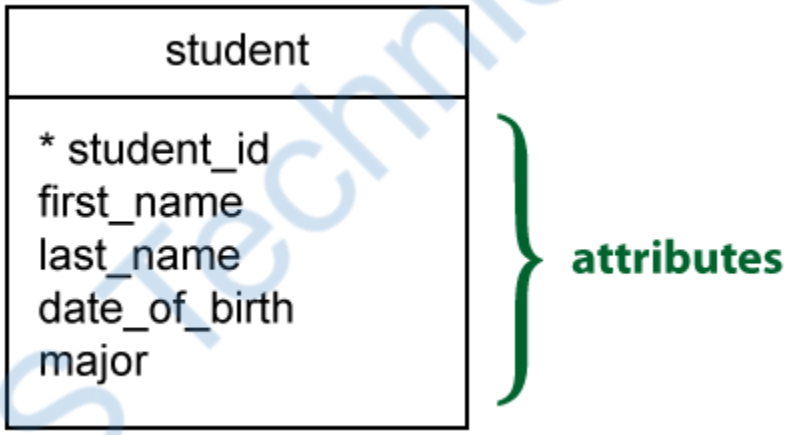


Figure 2.16

Relationships between the entities are represented by lines that connect the entities. The following symbols are used to represent the cardinality

Mandatory One

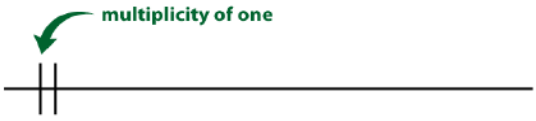


Figure 2.17

OPtional One

Figure 2.18

Mandatory Many

Figure 2.19

Optional Many

Figure 2.20

Example

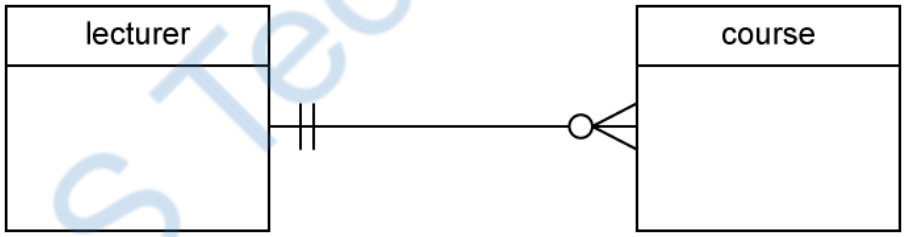


Figure 2.21

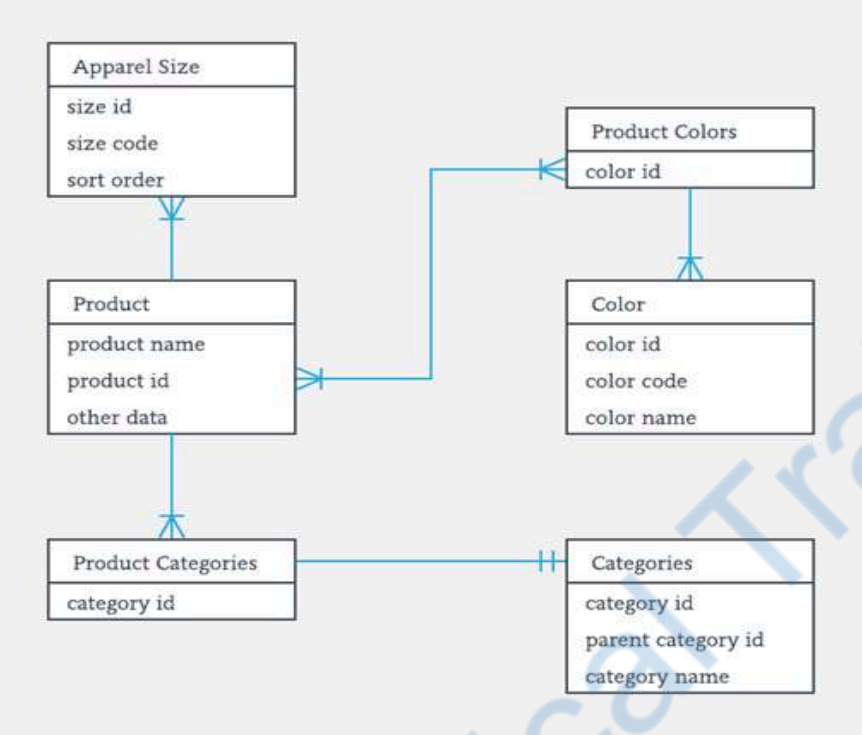


Figure 2.22

Normalization

Normalization, in this context, is the process of organizing data within a relational database to eliminate data anomalies, such as redundancy and enhance data integrity.Normalization divides larger tables into smaller tables and links them using relationships.

Anomaly

Any form of change to data in a table can lead to errors or inconsistencies. These changes can be adding a new row, updating the existing data or deleting one or more records. This may lead to unintended loss of data. This is called Anomaly. Three types of anomalies exist. They are

Insertion anomaly: An insertion anomaly is the inability to add data to the database due to the absence of other data

Example

Employee_ID	Name	Department	Student_Group
123	J. Longfellow	Accounting	Beta Alpha Psi
234	B. Rech	Marketing	Marketing Club
234	B. Rech	Marketing	Management Club

Table 2.1

In the Above table if Employee ID is a mandatory field, then a new department cannot be added until an employee is added.

Update anomaly: An update anomaly is a data inconsistency that results from data redundancy and a partial update

Example : In table 2:2 If the name of the employee 234 has to be changed, it must be updated twice otherwise there will be inconsistent data.

Employee_ID	Name	Department	Student_Group
123	J. Longfellow	Accounting	Beta Alpha Psi
234	B. Rechar	Marketing	Marketing Club
234	B. Rech	Marketing	Management Club

Table 2:2

Delete anomaly: A deletion anomaly is the unintended loss of data due to deletion of other data.

Example: In table 2:2, If the student group Beta Alpha Psi was disbanded and deleted from the table, then the details of Longfellow will be lost.

Update, deletion, and insertion anomalies are very undesirable in any database. Anomalies are avoided by the process of normalization. The purpose of normalization is to avoid complexities , eliminate redundant data and to organize data in a consistent way.

Normal Forms

Normal forms are a series of guidelines that help to ensure that the design of a database is efficient , organized and free from data anomalies. Even though there are up to 5 normal forms, the most commonly used normal forms are

1) First Normal Form.
2) Second Normal Form.
3) Third Normal Form.
4) Boyce & Codd Normal form.

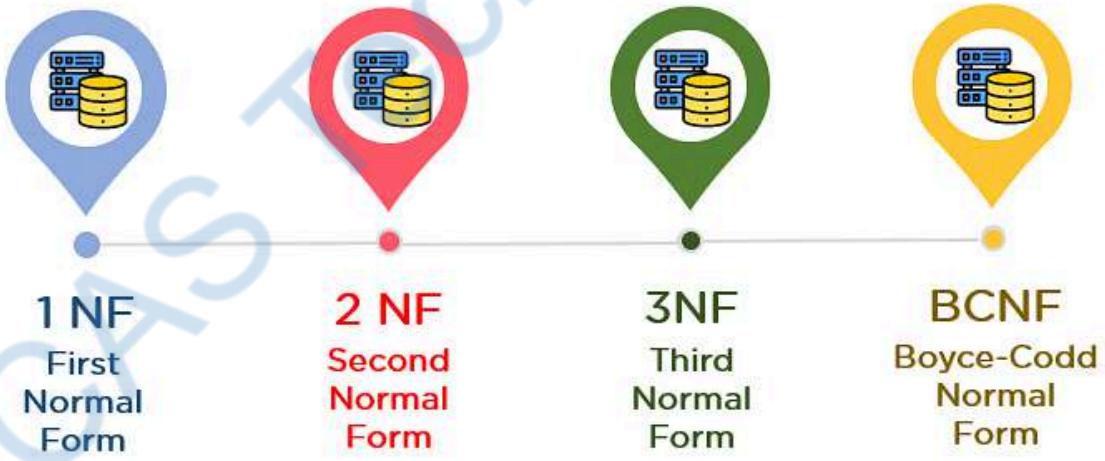


Figure 2.23

First Normal Form (1NF)

For a table to be in first normal form it must meet the following criteria:

- ➔ A single column must not hold more than one value (atomicity)
- ➔ All columns in a table should have unique names
- ➔ All columns must adhere to the domain.

Example:

Title	Author	Borrowed by
To Kill a Mockingbird	Harper Lee	John Doe, Jane Smith, James Brown
The Lord of the Rings	J. R. R. Tolkien	Emily Garcia, David Lee
Harry Potter and the Sorcerer's Stone	J.K. Rowling	Michael Chen

Table 2.3

The above table is not in 1NF since the values in Borrowed by field are not atomic.

To achieve first normal form without losing data, the table must be replaced with two relations as below.

Books Table

Book ID	Title	author
1	To Kill a Mockingbird	Harper Lee
2	The Lord of the Rings	J. R. R. Tolkien
3	Harry Potter and the Sorcerer's Stone	J.K. Rowling

Table 2.4

Borrowers Table

Borrower ID	Name	Book ID
B1	John Doe	1
B2	Jane Smith	1
B3	James Brown	1
B4	Emily Garcia	2
B5	David Lee	2
B6	Michael Chen	3

Table 2.5

Second Normal Form (2NF)

For a table to be in the Second Normal Form, it should follow the following:

- It should be in its first normal form.
- There should be no partial dependency.

Before proceeding further, it is necessary to understand two important concepts 1) Keys and 2) Dependencies

Keys

In a database management system (DBMS), a key is an attribute or set of attributes that uniquely identifies a row in a table. Keys are essential concepts that help in identifying, organizing, and establishing relationships between data in a relational database. They ensure that data integrity and consistency are maintained across the database.

Different types of keys

1. **Candidate key**

   The candidate keys refer to those attributes that identify rows uniquely in a table. In a table, we select the primary key from a candidate key. Thus, a candidate key has similar properties as that of the primary keys

that we have explained above. In a table, there can be multiple candidate keys.

2. **Primary key**

   A primary key is a unique identifier for a record in a database in a table. It will ensure that no two records have the same values for this field and cannot be NULL. A table can have only one primary key.

3. **Super Key**

   A super key refers to the set of all those keys that help us uniquely identify all the rows present in a table. It means that all of these columns present in a table that can identify the columns of that table uniquely act as the super keys

4. **Alternate Key**

   any table can consist of multiple choices for the primary key. But, it can only choose one. Thus, all those keys that did not become a primary key are known as alternate keys.

5. **Foreign key**

   A Foreign key refers to the values of the primary key column in another table. It is used to establish a relationship between the two tables. It allows duplicate and null values. A table can have many Foreign keys and a single Primary key can be referred to by many foreign keys.

6. **Unique key**

   A unique key refers to a column/a set of columns that identify every record uniquely in a table. All the values in this key would have to be unique. Remember that a unique key is different from a primary key. It is because it is only capable of having one null value. A primary key, on the other hand, cannot have a null value.

7. **Composite Key**

   The composite key refers to a set of multiple attributes that help us uniquely identify every tuple present in a table. The attributes present in a set may not be unique whenever we consider them separately. Thus, when we take them all together, it will ensure total uniqueness.

Dependency

Dependencies in DBMS is a relation between two or more attributes. A dependency is denoted by an arrow "→".

The following are the types of dependencies.

- Functional dependency
- Fully- Functional dependency.
- Partial dependency.
- Transitive dependency
- Multi valued Dependency

If X and Y are attributes then the functional dependency is

X--->Y

Represented as x tends to y. Here X determines y or y is functionally dependent on X. If the value of X is known then y can be identified uniquely. Here X is called as **determinant** and Y is called as **dependent.** When x ---> y is functionally dependent when it satisfies the following rule.

If t1.x=t2.x then t1.y must be = t2.y

X can be a set of attributes y also can be a set of attributes.

Example Data

Sid	Sname	City	Course
101	Suraj	Delhi	Java
102	Smith	Mysore	RDBMS
101	Suraj	Delhi	HTML
103	Jones	Chennai	MongoDB
102	Smith	Mysore	Java
104	Patel	Agra	HTML
101	Suraj	Delhi	RDBMS

Table 2.6

Partial dependency

A partial dependency is a functional dependency where the dependent attributes are partially determined by the determinant attributes.

Employee_ID	Task_No	Employee_Name	Task_Name
C01	34	Mona	App Development
C02	58	Genine	UX/UI Designing

Table 2.7

In the table given above, we have a partial dependency. Here is how:
Here, the prime key attributes are Employee_ID and Task_No, and also:
Employee_ID = A unique ID of the employee

Employee_Name = Name of the employee

Task_No = A unique ID of the task

Task_Name = The name of the task

As stated above, the non-prime attributes are Employee_Name and Task_Name. These must be dependent functionally on the part of the candidate key so as to be Partial Dependent.

The Employee_Name can be determined using the Employee_ID. It actually makes the relation Dependent Partially.

The Task_Name can be determined using the Task_No. It makes the relation Dependent Partially.

Thus, the <Employee_Task> relation would violate the Second Normal Form in Normalization and is considered to be a bad database design.

We decompose the tables to remove Partial Dependency along with the violation on the second normal form:

<Employee_Info>

Employee_ID	Task_No	Employee_Name
C01	34	Mona
C02	58	Genine

Table 2.8

<Task_Info>

Task_No	Task_Name
34	App Development
58	UX/UI Designing

Table 2.9

Thus, the relation happens to be in the second normal form in the case of Database Normalization.

Transitive dependency

A transitive dependency refers to some non-prime attribute other than the candidate key that depends on another non-prime attribute that is dependent entirely on the candidate key.

EMPNO	ENAME	EMP_ZIP	CITY	DISTRICT	STATE
7839	King	641035	Saravanampatti	CBE	TN
7782	Smith	222088	Gwalior	Ratan	MP
7852	Scott	60028	Anna Nagar	Chennai	TN

Table 2.10

The above table includes a transitive dependency. That is
EMPNO —> ENAME, EMP_ZIP
EMP_ZIP → CITY

Thus, the following has a transitive type of functional dependency.

EMPNO -> CITY

Multi valued Functional dependency

Multi valued dependency is a property of a database that occurs when two distinct attributes in a table are independent of each other but depend on a third attribute.
Example:
Let's consider a scenario where a car manufacturing company produces two distinct colours, red and blue, for each of its models every year.

CAR_MODEL	MANUF_MONTH	COLOUR
S2022	FEB	Blue
S2022	MAR	Red
S2023	APR	Blue
S2023	MAY	Red

Table 2.11

In the given scenario, the COLOUR and MANUF_MONTH columns are dependent on CAR_MODEL but are independent of each other. Therefore, they can be classified as multivalued, being dependent on CAR_MODEL. The dependencies can be represented as:

CAR_MODEL ⟶ ⟶ MANUF_MONTH
CAR_MODEL ⟶ ⟶ COLOUR
This can be interpreted as "CAR_MODEL determines MANUF_MONTH" and "CAR_MODEL determines COLOUR".

To eliminate this dependency, we divide the table into two as below:

CAR_MODEL	COLOUR
S2022	Blue
S2022	Red
S2023	Blue
S2023	Red

CAR_MODEL	MANUF_MONTH
S2022	FEB
S2022	MAR
S2023	APR
S2023	MAY

Table 2.11

Third Normal form 3NF

For a table to be in the Third Normal Form, it should follow the following:

- It should be in its second normal form.
- There should be no transitive dependency.

Example:

StudentID	CourseID	StudentName	CourseName	InstructorID	InstructorName
1	101	Alice	Math	201	Dr. Smith
1	102	Alice	English	202	Prof. Johnson
2	101	Bob	Math	201	Dr. Smith
2	103	Bob	History	203	Dr. Brown

Table 2.13

In this table:

- StudentName is dependent on StudentID.
- CourseName is dependent on CourseID.
- InstructorID and InstructorName are dependent on CourseID.

Why the Table is Not in 3NF

- InstructorName is dependent on InstructorID, which in turn is dependent on CourseID. This means InstructorName is transitively dependent on CourseID through InstructorID.

- This transitive dependency violates the rules of 3NF.

Transforming to 3NF

To convert this table into 3NF, we need to eliminate the transitive dependency by creating separate tables for Student, Course, Instructor, and Enrollment.

1. Student Table:

StudentID	StudentName
1	Alice
2	Bob

2. Course Table:

CourseID	CourseName	InstructorID
101	Math	201
102	English	202
103	History	203

3. Instructor Table:

InstructorID	InstructorName
201	Dr. Smith
202	Prof. Johnson
203	Dr. Brown

4. Enrollment Table (This table represents the many-to-many relationship between students and courses):

StudentID	CourseID
1	101
1	102
2	101
2	103

Table 2.14

Boyce Codd normal form (BCNF):

- It is considered as the advanced level of 3NF. It is stricter than 3NF.
- Called as3.5 NF
- The table should be in third norma; form.
- Right-Hand Side (RHS) attribute of the functional dependencies should depend on the super key of that particular table.

For a functional dependency X → Y, has to be part of the super key of the provided table.

Example:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

Table 2.15

In the above table, functional dependency can be described as follows

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPTNO_NO}

Here the candidate key is {EMP_ID, EMP_DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Now, this is in BCNF because left side of both the functional dependencies is a key.

Answer the following questions.

1) The entity relationship set is represented in E-R diagram as
   a) Double diamonds
   b) Undivided rectangles
   c) Dashed lines
   d) Diamond

2) Weak entity set is represented as
   a) Underline
   b) Double line
   c) Double diamond
   d) Double rectangle

3) What term is used to refer to a specific record in your music database; for instance; information stored about a specific album?
   a) Relation
   b) Instance
   c) Table
   d) Column

4) By normalizing relations or sets of relations, one minimizes ___.
   A. Data
   B. Fields
   C. Redundancy
   D. Database

5) In addition to removing undesirable characteristics, normalization also eliminates ___ anomalies.

   A. Insert
   B. Update
   C. Delete
   D. All of the above

6) When a relation contains an atomic value, it is a ___ relation.

A. 1NF
B. 2NF
C. 3NF
D. BCNF

7) 2NF relations are those that are in 1NF with all the attribute types dependent on the ___ key.

A. Primary
B. Foreign
C. Composite
D. Alternate

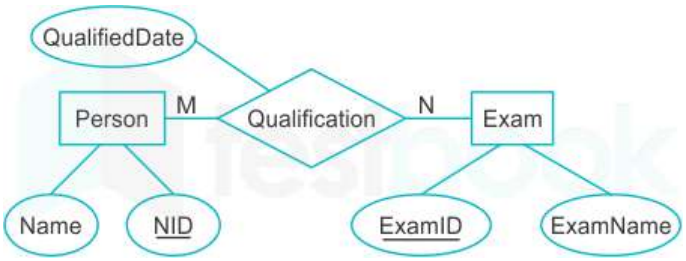
8) Which of the following shapes is used to represent an attribute.

A) Rectangle
B) Diamond
C) Oval
D) Double rectangle

9) Which symbol is used to denote a derived attribute?

A) Double ellipse
B) Dotted ellipse
C) Rectangle.
D) Ellipse with attribute name underlined

10) Consider the following entity relationship diagram.



Which of the following relations will not hold if the above ERD is converted into a relational model.

A) Person (NID, Name)
B) Qualification (NID, ExamID,QualifiedDate)
C) Exam(ExamID, EnameName,NID)
D) Exam(ExamID, EnameName)

11) The ER Model data is represented as
A) Entity Only
B) Attribute only
C) Entities , relationships and attributes
D) None of the above

12) In a relational model, relationships between relations or tables are created using
A) Composite keys
B) Determinants
B) candidate keys
D) Foreign keys