CHAPTER 3

SQL

3.0 INTRODUCTION

- SQL is a standard language for accessing and manipulating databases. SQL stands for Structured Query Language. SQL lets you to access and manipulate databases.
- SQL is the standard language for Relational Database System. SQL is a database computer language designed for the retrieval and management of data in a relational database.
- All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.
- In this chapter we study SQL with its basic concepts.

3.1 INTRODUCTION TO QUERY LANGUAGE

- A query language for relational model is based on relational algebra. It provides a notion for representing queries.
- The commercial database system requires a more user-friendly query language. SQL i.e. Structured Query Language (SQL), it uses a combination of relational algebra and relational calculus constructs.
- SQL contains other capabilities besides querying a database.
- These capabilities includes features for defining the structure of the data, features for modifying data in the databases and features for specifying security constraints.
- There are numerous versions of SQL. The original version was developed at IBM's San Jose Research Laboratory by Dr. E. F. Codd in early 1970. This was based on the relational database model.
- After that IBM's research division and Donald Chamberlin developed a prototype language called sequel i.e. Structured English Query Language.
- For some legal reason, IBM eventually changed the name sequel/2 to SQL (Structured Query Language). In 1986, ANSI and ISO had declared SQL as a standard language. Therefore sometimes, it is referred as Standard Query Language. From 1986, SQL has become universally adopted language.

Features of SQL:

1. SQL is very flexible.
2. SQL based applications are portable i.e., they can be run on any system.
3. SQL is a high level language.
4. SQL is a free-format syntax which gives users the ability to structure SQL statements on any of the screen.
5. SQL can be embedded also in front end application code like of PHP, JAVA and so on.

Elements of SQL:

- SQL contains following elements:
  1. Queries retrieve data based on particular or specific creation.
  2. Statements may control program flow, transactions etc.
  3. Expressions can produce either scalar values or tables which consisting of rows and columns of data.
  4. Clauses are in some cases optional, constituent components of queries and statements.

- The user can write a SQL statement and submit it to the DBMS. DBMS will retrieve the appropriate data from disk and return it to the user, (Refer Fig. 3.1).

Architecture of SQL:

- Fig. 3.2 shows SQL architecture which shows detailed process of SQL.
- When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

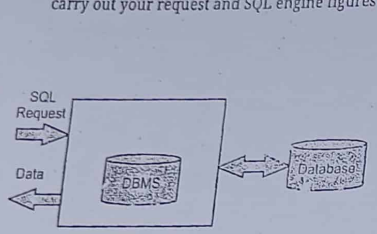

Fig. 3.1: Working of SQL

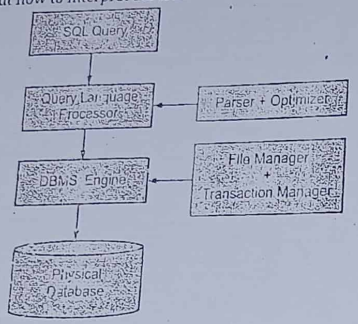

Fig. 3.2: Architecture of SQL

Advantages of SQL:

1. **Simplicity:** Many problems can be expressed in SQL more easily, simply and concisely than in lower level languages. Simplicity, in turn means increase productivity.
2. **Completeness:** The language is relationally complete. User can write very large class of queries.
3. **Data Independence:** SQL DML statements include no reference to explicit access paths such as indexes or physical sequence.
4. **Ease of Extension:** It can be easily extended by using built-in functions.
5. **Support for Higher Level Languages:** SQL can conveniently be used with higher languages.

3.2 BASIC STRUCTURE (April 16)

- SQL is used as a standard relational database language because it allows user to access data in RDBMS (Relational Database Management System) such as Oracle, Sybase, Informix, PostgreSQL etc.
- SQL also allows the user to define the data in a database and manipulates that data.
- The non-relational database systems also supports SQL interface. SQL is a special purpose, non-procedural language that supports the definition, manipulation and control of data in RDBMS.
- It is called as special-purpose because only the database can be handled.
- The SQL has several parts like Data Definition Language (DDL), Data Manipulation Language (DML) View definition, Embedded SQL, Integrity, Authorization, Transaction control.
- Even though SQL has various parts, we will consider only DDL which allows us to create and maintain data manipulation objects such as tables, views, sequence.
- View means duplicate copy of table used for security purpose. The sequence is for creating numeric values.
- A query is a database command that retrieves records. Using queries, we can pull data from one or more fields from one or more tables.
- The statement which is used to retrieve the information is called as a query. A query is used to extract data from the database in a readable format according to the user's request.

- For instance, if you have an employee table, you might issue a SQL statement that returns the employee who is paid the most. This request to the database for usable employee information is a typical query that can be performed in a relational database.
- The basic structure of an SQL expression consists of three clauses as given below:
  1. The SELECT clause corresponds to projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.
  2. The FROM clause corresponds to the Cartesian product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.
  3. The WHERE clause corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the FROM clause.
- A typical SQL query has the following form or structure:

```
SELECT A1, A2, ....., An
FROM r1, r2, ....., rn
WHERE p;
```

where, $A_1 \ldots A_n$ - represents an attribute,
$r_1 \ldots r_n$ - represents relation, and
p - is a predicate.

- SQL forms the cartesian product of the relations named in the from clause, performs a relational algebra selection using the where clause predicate, and then projects the result onto the attributes of the select clause.
- **Example:**

```
SELECT *
FROM  Book
WHERE price > 100.00
```

- Query processing includes translation of high-level queries into low-level expressions that can be used at the physical level of the file system, query optimization and actual execution of the query to get the result.
- It is a three-step process that consists of parsing and translation, optimization and execution of the query submitted by the user as shown in Fig. 3.3.

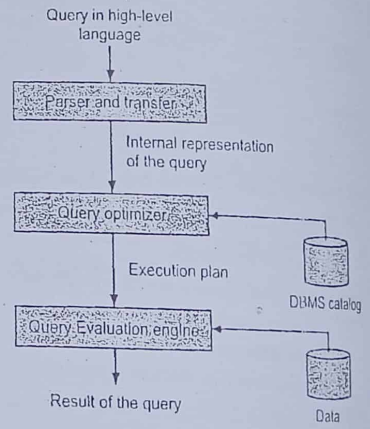

Fig. 3.3: Query Processing Steps

Creating and Deleting a Database:

To Create Database:

- The CREATE DATABASE statement is used to create a new database.
- After creating a database, we can create several other database objects (tables, views, procedures etc.) into it.

Syntax:

```
CREATE DATABASE Database_Name;
```

- Always database name should be unique within the RDBMS.
- **Example:** If you want to create new database <user_DB>, then CREATE DATABASE statement would be as follows:

```
CREATE DATABASE user_DB;
```

To Drop/Delete Database:

- The DROP DATABASE statement is used to drop or delete a database.
- Dropping of the database will drop all database objects (tables, views, procedures etc.) inside it.
  Syntax: `DROP DATABASE Database_Name;`
  Example: `DROP DATABASE user_DB;`
- PostgreSQL provides two ways of creating a new database using CREATE DATABASE, an SQL command and Using createdb a command-line executable.
- The syntax for createdb is: `createdb [option...] [dbname [description]]`.
- In PostgreSQL we can delete a database using DROP DATABASE, an SQL command and using dropdb a command-line executable.
- PostgreSQL command line executable dropdb is a command-line wrapper around the SQL command DROP DATABASE. The syntax for dropdb is: `dropdb [option...] dbname`.

Data Types in PostgreSQL:

- PostgreSQL supports a wide set of data types. Some of them are:

1. **Numeric:**
   PostgreSQL provides two distinct types of numbers namely, integers and floating-point numbers.
   Integer (INT) is a 4-byte integer that has a range from - 2,147,483,648 to 2,147,483,647.
   float(n) is a floating-point number whose precision, at least, n, up to a maximum of 8 bytes.
   numeric or numeric(p,s) is a real number with p digits with s number after the decimal point. The numeric(p,s) is the exact number.
2. **Character:**
   CHAR(n) is the fixed-length character with space padded. If you insert a string that is shorter than the length of the column, PostgreSQL pads spaces. If you insert a string that is longer than the length of the column, PostgreSQL will issue an error.
   VARCHAR(n) is the variable-length character string. With VARCHAR(n), you can store up to n characters.
3. **Date and Time:**
   PostgreSQL supports a full set of SQL date and time types. The date data type is 4 bytes and shows date (no time of day).
4. **Boolean:**
   PostgreSQL provides the standard SQL type Boolean. The Boolean data type (1 byte) can have the state true or false.

3.3 DATA DEFINITION LANGUAGE (DDL) COMMANDS (April 16, 18)

- DDL commands used for definition and creation objects in database like Table.
- DDL commands are mainly used for design and definition the structure of a database.
- SQL DDL commands are CREATE, ALTER, DROP, DESC, RENAME, and TRUNCATE as explained in following topics.

1. **CREATE TABLE Command:**

- The SQL CREATE TABLE statement is used to create a new table.
- Creating a basic table involves naming the table and defining its columns and each column's data type.

Syntax:

```
CREATE TABLE table_name(
  column1 datatype,
  column2 datatype,
  column3 datatype,
  .....
  columnN datatype,
PRIMARY KEY( one or more columns )
);
```


- Following example, creates a CUSTOMERS table with ID as primary key and NOT NULL are the constraints showing that these fields can not be NULL while creating records in this table:

Syntax:

```
CREATE TABLE CUSTOMERS(
   ID          INT             NOT NULL,
   NAME        VARCHAR (20)    NOT NULL,
   AGE         INT             NOT NULL,
   ADDRESS     CHAR (25),
   SALARY      DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

- We can verify if our table has been created successfully by looking at the message displayed by the SQL server, otherwise we can use DESC command as follows:

```
DESC CUSTOMERS;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI		
NAME	varchar(20)	NO			
AGE	int(11)	NO			
ADDRESS	char(25)	YES		NULL	
SALARY	decimal(18,2)	YES		NULL	

- CREATE TABLE command consist of following constraints:
  (i) Unique Constraint ensures that no two rows have the same value in the specified columns.
  (ii) Primary Key Constraints declares a column as the primary key of the table.
  (iii) Check Constraint limits values that can be inserted into a column of a table.
  (iv) Default Constraint assigns a default value can be specified for a column using the default clause.

2. **DROP TABLE Command:**

- The DROP TABLE statement is used to remove a table definition and all data, indexes, constraints, and permission specifications for that table.

Syntax:

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE CUSTOMERS;
```

- Now, if you would try DESC command, then you would get error as follows:

```
DESC CUSTOMERS;
ERROR 1146 (42S02): Table 'TEST.CUSTOMERS' doesn't exist
```

3. **DESC Command:**

- The table structure can be described by using DESC command

Syntax:

```
DESC table_name
```

Example: Let us first verify CUSTOMERS table and then we would delete it from the database:

```
DESC CUSTOMERS;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI		
NAME	varchar(20)	NO			
AGE	int(11)	NO			
ADDRESS	char(25)	YES		NULL	
SALARY	decimal(18,2)	YES		NULL	

- 'Delete' is used to delete tuples from the table whereas 'drop' is used to delete the whole table.

4. **TRUNCATE Command:**

- TRUNCATE TABLE command is used to delete complete data from an existing table.

```
TRUNCATE TABLE table_name;
```

Example: Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

- Following is the example to truncate:

```
TRUNCATE TABLE CUSTOMERS;
```

- Now, CUSTOMERS table is truncated and following would be the output from SELECT statement:

```
SELECT * FROM CUSTOMERS;
Empty set (0.00 sec)
```

5. **ALTER TABLE Command:** (Oct.-17, April-18)

- The ALTER TABLE command is used to modify the definition (structure) of a table by modifying the definition of its columns.
- The ALTER command is used to perform the following functions:
  (i) Add, drop, modify table columns,
  (ii) Add and drop constraints, and
  (iii) Enable and Disable constraints.

Syntax:

- The basic syntax for ALTER TABLE to a add a new column in a existing table is,

```
ALTER TABLE table_name ADD column_name datatype;
```

- The basic syntax of ALTER TABLE to DROP COLUMN in an existing table is,

```
ALTER TABLE table_name DROP COLUMN column_name;
```

- **Example:** Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

- Following is the example to ADD a new column in an existing table:

```
ALTER TABLE CUSTOMERS ADD SEX char(1);
```

- Now, CUSTOMERS table is changed and following would be output from SELECT statement:

ID	NAME	AGE	ADDRESS	SALARY	SEX
1	Ramesh	32	Ahmedabad	2000.00	NULL
2	Ramesh	25	Delhi	1500.00	NULL
3	Kaushik	23	Kota	2000.00	NULL
4	Kaushik	25	Mumbai	6500.00	NULL
5	Hardik	27	Bhopal	8500.00	NULL
6	Komal	22	MP	4500.00	NULL

- Following is the example to DROP sex column from existing table:

```
ALTER TABLE CUSTOMERS DROP SEX;
```

- Now, CUSTOMERS table is changed and following would be output from SELECT statement:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Ramesh	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Kaushik	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

- To add a constraint, the table constraint syntax is used. For example:

```
ALTER TABLE products ADD CHECK (name <> '');
ALTER TABLE products ADD CONSTRAINT some_name UNIQUE (product_no);
ALTER TABLE products ADD FOREIGN KEY (product_group_id) REFERENCES product_groups;
```

- To change a column to a different data type, use a command

```
ALTER TABLE products ALTER COLUMN price TYPE numeric(10,2);
To rename a column:
```

- ALTER TABLE products RENAME COLUMN product_no TO product_number;
- To rename a table use following query:

```
ALTER TABLE products RENAME TO items;
```

6. **RENAME Command:**

- RENAME command is used to rename a table.

Syntax:

```
RENAME TABLE old-table-name to new-table-name
```

- **Example:**

```
CREATE TABLE employees
(id NUMBER(6),
name VARCHAR(20)
);
INSERT INTO employees( ID, Name ) values( 1, 'Amar');
INSERT INTO employees( ID, Name ) values( 2, 'Deepa');
INSERT INTO employees( ID, Name ) values( 3, 'Kiran');
SELECT * FROM employees;
```

Output:

ID	Name
1	Amar
2	Deepa
3	Kiran

- For renaming the table use following query:

```
RENAME TABLE employees TO employees_new;
SELECT * FROM employees_new;
```

Output:

ID	Name
1	Amar
2	Deepa
3	Kiran

3.4 DATA MANIPULATION LANGUAGE (DML) COMMANDS (April 16, Oct. 16)

- DML commands are used for manipulating data in database.
- DML commands are used for storing, retrieving, modifying, and deleting data. Various DML commands are explained below:

1. **INSERT INTO Command:**

- The INSERT statement is used to add new rows (records) of data to a table in the database.

Syntax:

```
INSERT INTO TABLE_NAME
[(column1, column2, column3,...columnN)]
VALUES (value1, value2, value3,...valueN);
```

Here, column1, column2,...columnN are the names of the columns in the table into which you want to insert data.

- You can populate data into a table through select statement over another table provided another table has a set of fields, which are required to populate first table. Here, is the syntax:

```
INSERT INTO first_table_name [(column1, column2, ... columnN)]
SELECT column1, column2, ...columnN
FROM second_table_name
[WHERE condition];
```

- Following statements would create six records in CUSTOMERS table:

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

- You can create a record in CUSTOMERS table using second syntax as follows:

```
INSERT INTO CUSTOMERS
VALUES (7, 'Amar', 24, 'Indore', 9500.00 );
```

- All the above statements would produce the following records in CUSTOMERS table:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Amar	24	Indore	9500.00

2. **UPDATE Command:** (Oct. 17)

- *The UPDATE statement is used to modify the existing rows in a table.*

- We can use WHERE clause with UPDATE query to update selected rows otherwise all the rows would be affected.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

Example: Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

- Following is an example, which would update ADDRESS for a customer whose ID is 6:

```
UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;
```

- Now, CUSTOMERS table would have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	Pune	4500.00

- If we want to modify all ADDRESS and SALARY column values in CUSTOMERS table, we do not need to use WHERE clause and UPDATE query would be as follows:

```
UPDATE CUSTOMERS
SET ADDRESS = 'Pune', SALARY = 1200.00;
```

- Now, CUSTOMERS table would have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Pune	1200.00
2	Khilan	25	Pune	1200.00
3	Kaushik	23	Pune	1200.00
4	Chaitali	25	Pune	1200.00
5	Hardik	27	Pune	1200.00
6	Komal	22	Pune	1200.00

3. **DELETE Command:**

- The DELETE Statement is used to delete rows (records) from a table.
- We can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

Syntax:

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

Example: Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

- Following is an example, which would DELETE a customer, whose ID is 6:

```
DELETE FROM CUSTOMERS
WHERE ID = 6;
```

- Now, CUSTOMERS table would have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00

- If we want to DELETE all the records from CUSTOMERS table, we do not need to use WHERE clause and DELETE query would be as follows:

```
DELETE FROM CUSTOMERS;
```

- Now, CUSTOMERS table would not have any record.

3.5 FORMS OF A BASIC SQL QUERY (EXPRESSION AND STRINGS IN SQL) (April 16)

- A relational database consists of a collection of relation, each of which is assigned a unique name.
- SQL allows the use of null values to indicate that value is unknown or does not exist. For retrieving record from table, we use basic structure for data retrieval.
- This SQL expression consists of following three clauses or components:

```
SELECT <list of attributes>
FROM <table name>
WHERE <condition>;
```

- SQL forms the cartesian product of the relation named in the FROM clause, performs a relational algebra selection using the WHERE clause predicate and projects the result onto the attribute of the SELECT clause. The semicolon (;) is the terminator of SQL expression.

3.5.1 Simple SQL Queries

- Lets consider the relational database of Banking System given below. Underlined attributes are the primary keys.

  Branch (branch_name, branch_city, assets)
  Customer (cust_name, cust_street, cust_city)
  Loan (loan_no, branch_name, amount)
  Account (acc_no, branch_name, balance)
  Borrower (cust_name, loan_no)
  Depositor (cust_name, acc_no)

- In above example, the relations (tables) branch and loan are related with one to many relationship. Branch and account are also related with one-to-many relationship. Customer and loan are related with many-to-many relationship. Similarly, customer and account are related with many-to-many relationship. Therefore, borrower and depositor are the two tables created to show many-to-many relationship.
- The E-R diagram is created from the given schema which is shown in Fig. 3.4.

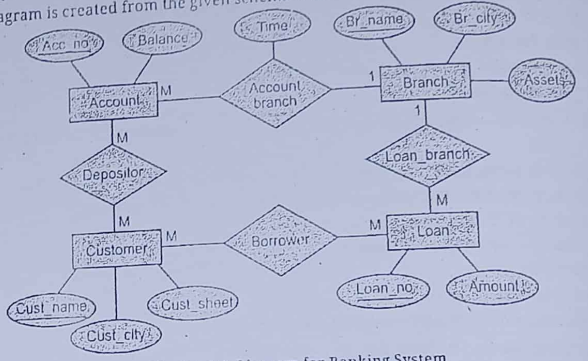

Fig. 3.4: E-R Diagram for Banking System

- When we want to write simple queries the relationships should be created. The relationship between different entities is shown in Fig. 3.5 which helps us to understand the above E-R diagram better.

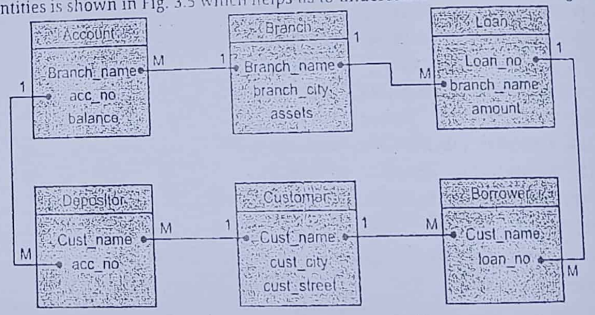

Fig. 3.5: Relationship Between Entities

- To understand the queries, let's maintain all the above tables (entities) with some data in the form of records, (Refer Fig. 3.6).

Relation = Loan

loan_no	branch_name	amount
1	J M Road	200000
2	J M Road	500000
3	F C Road	680000
4	M G Road	520000
5	G P Road	250000
6	F C Road	100000
7	F C Road	100000

(a)

Relation = Borrower

cust_name	loan-no
Umesh	1
Mugdha	3
Paresh	5
Rupali	2
Umesh	6
Rupali	7

(b)

Relation = Branch

branch_name	branch_city	assets
J M Road	Pune	100000
M G Road	Pune	500000
F C Road	Pune	200000
G P Road	Nagpur	120000

(c)

Relation = Account

acc_no	branch_name	balance
112	G P Road	10000
443	F C Road	5000
553	G P Road	2000
880	F C Road	15000
998	M G Road	20000

(d)

Relation = Customer

cust_name	cust_city	cust_street
Sayali	Nagpur	G P Road
Umesh	Pune	J M Road
Mugdha	Pune	F C Road
Nitin	Pune	F C Road
Rima	Nagpur	G P Road
Paresh	Pune	M G Road
Rupali	Pune	M G Road

(e)

Relation = Depositor

cust_name	acc_no
Sayali	443
Umesh	998
Rima	443
Sayali	880
Paresh	112

(f)

Fig. 3.6: Tuples in all Tables

3.5.1.1 SELECT Clause

- The SELECT clause in SQL is used to retrieve or fetch data from a database.

  Syntax:

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

- Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

Example: Find name of all branches in the loan relation:

```
SELECT branch_name
FROM loan;
```

- Here, is a terminator which shows the end of a query. The result of this query is shown in Fig. 3.7 (a).

branch_name
J M Road
J M Road
F C Road
M G Road
G P Road
F C Road
F C Road

Fig. 3.7 (a): Result of Select Query

- It displays the duplicate values also. If we want to force the elimination of duplicate values, insert a keyword DISTINCT before a field name. By default SQL takes a keyword ALL and that's why it displays all the values.
- The DISTINCT keyword is used to return only distinct (different) values.

  Syntax:

```
SELECT DISTINCT column_names
FROM table_name;
```

- The example with distinct is as follows:

```
SELECT DISTINCT branch_name
FROM loan;
```

- The result of this query is as shown in Fig. 3.7 (b).

branch_name
J M Road
F C Road
M G Road
G P Road

Fig. 3.7 (b): Result of Select with Distinct

- SQL allows us to use the keyword ALL to specify explicitly that duplicates are not removed. The query is as follows:

```
SELECT ALL branch_name
FROM loan;
```

- The result of this query is similar to the result depicted in Fig. 3.7 (a).

  Find all the attributes in loan relation,

```
SELECT loan_no, branch_name, amount
FROM loan;
```

- It displays the result depicted in Fig. 3.6 (a).
- When all the attributes from a table to be selected then in place of writing all the attribute names, a symbol '*' (asterisk) can be placed.
- So the above query can be written in the following format.

```
SELECT *
FROM loan;
```

- It will display the same result like Fig. 3.6 (a).
- The SELECT clause can also contain an arithmetic expression which includes the following operators in it, such as,

Addition	+
Subtraction	-
Mod (to find remainder)	%
Division	/
Multiplication	*
Raise to and Constants.	^

- Lets write some more queries using these operators.

  Find all the attributes in loan relation where interest on the amount is calculated as 3% of the amount.

  Query:

```
SELECT loan_no, branch_name, amount, amount * 0.03
FROM loan;
```

- As we know, the table has only three attributes, but we want to have result of amount * 0.03.
- The result of this query is shown in Fig. 3.8.

loan_no	branch_name	amount	Expr1
1	J M Road	200000	6000
2	J M Road	500000	15000
3	F C Road	680000	20400
4	M G Road	520000	15600
5	G P Road	250000	7500
6	F C Road	100000	3000
7	F C Road	100000	3000

Fig. 3.8: Result of Select with 3% of Amount

WHERE Clause

- It corresponds to select operation. This clause is used to define a predicate.
- While specifying predicate, SQL uses logical connectives AND, OR and NOT. SQL allows us to use the comparison operator to compare strings and arithmetic expressions as well as data types.
- The available comparison operators are <, < =, >, > =, = , <>.

Syntax:

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

- Let's write queries for WHERE clause.

  Find all the loan numbers for loan made at F C Road branch with loan amount greater than 300000.

  Query:

```
SELECT loan_no
FROM loan;
WHERE branch_name = 'FC Road' AND amount > 300000;
```

- It will select all loan numbers first from loan table and then apply predicate for branch_name and amount.
- The result of the query is as shown in Fig. 3.9.

loan_no
3

Fig. 3.9: Result of where Clause

Find the loan number of those loans with loan amount between Rs. 200000 to 550000.

Query:

```
SELECT loan_no
FROM loan
WHERE amount ≥ 200000 and amount ≤ 550000;
```

- The result is as shown in Fig. 3.10.

loan_no
1
2
4
5

Fig. 3.10: Result of Range Checking

BETWEEN Operator:

- The range can be specified by relational operator. If we do not want to use these operators, the range can be specified by a keyword BETWEEN.
- This operator allows selecting range. So the above query can be rewritten as follows:

```
SELECT loan_no
FROM loan
WHERE amount Between 200000 and 550000;
```

- The result is as shown in Fig. 3.10.

FROM Clause

- FROM clause itself defines a Cartesian product of a relation. This clause is also used to find natural join of more than one table.

- The SQL From clause is the source of a rowset to be operated upon in a Data Manipulation Language (DML) statement.

Syntax:

```
SELECT *
FROM table_name
WHERE predicate
```

- Find all customers who have a loan from the bank, find their names and loan_no.

  Query:

```
SELECT cust_name, loan.loan_no, borrower.loan_no
FROM loan, borrower;
```

- The result will have Cartesian product of borrower and loan as shown in Fig. 3.11.

cust_name	loan.loan_no	borrower.loan_no
Umesh	1	1
Mugdha	3	1
Paresh	5	1
Rupali	2	1
Umesh	6	1
Rupali	7	1
Umesh	1	2
Mugdha	3	2
Paresh	5	2
Rupali	2	2
Umesh	6	2
Rupali	7	2
Umesh	1	3
Mugdha	3	3
Paresh	5	3
Rupali	2	3
Umesh	6	3
Rupali	7	3
Umesh	1	4
Mugdha	3	4
Paresh	5	4
Rupali	2	4
Umesh	6	4
Rupali	7	4
Umesh	1	5
Mugdha	3	5
Paresh	5	5
Rupali	2	5
Umesh	6	5
Rupali	7	5
Umesh	1	6

Contd..

Mugdha	3	6
Paresh	5	6
Rupali	2	6
Umesh	6	6
Rupali	7	6
Umesh	1	7
Mugdha	3	7
Paresh	5	7
Rupali	2	7
Umesh	6	7
Rupali	7	7

Fig. 3.11: Result of borrower × loan

- For finding natural join, we have to provide the comparison using where clause. Now query 6 will be as follows:

```
SELECT cust_name, loan_no
FROM loan, borrower
WHERE loan.loan_no = borrower.loan_no;
```

- This will find cartesian product first and then natural join. The result of natural join is as shown in Fig. 3.12.

cust_name	loan_no
Umesh	1
Rupali	2
Mugdha	3
Paresh	5
Umesh	6
Rupali	7

Fig. 3.12: Result of natural join

Find names and loan numbers of all customer who have loan at FC Road branch.

Query:

```
SELECT cust_name, loan.loan_no
FROM Loan, Borrower
WHERE loan.loan_no = borrower.loan_no
AND branch_name = 'FC Road';
```

- The result is depicted in the Fig. 3.13.

cust_name	loan_no
Mugdha	3
Umesh	6
Rupali	7

Fig. 3.13: Result of natural join where branch-name = 'FC Road'

3.5.14 Rename Operation

- SQL provides a mechanism for renaming both the relations and attributes. For this keyword AS is used. It can be appear in both SELECT and FROM clause.

- Let's consider previous query, suppose we want to change name of loan-loan-no to loan-id, the query is rewritten as follows:

Query:

```
SELECT cust_name, loan.loan_no AS loan_id
FROM loan, borrower
WHERE loan.loan_no = borrower.loan_noand branch_name = "FC Road";
```

- The result of this query is similar to Fig. 3.13 except the change in the name of attribute. This is depicted in Fig. 3.14.

cust_name	loan_id
Mugdha	3
Umesh	6
Rupali	7

Fig. 3.14: Result of Natural Join with as Keyword

- Similarly, we can change the relation name.

Find names of all branches that have assets greater than atleast one branch located in pune.

Query:

```
SELECT distinct T.branch_name
FROM branch as T, branch as S
WHERE T.assets > S.assets
    AND S.branch_city = 'Pune';
```

- Here, S and T are called as Tuple variables which are used to compare two records (tuples) in the same relation.
- The tuple variables are defined in the FROM clause.
- In 9th query we have used two tuple variables to compare the assets and attribute.
- The result of this query is as depicted in Fig. 3.15.

branch_name
FC Road
GP Road
MG Road

Fig. 3.15: Result of T.assets > S.assets and S.branch_city = 'pune'

- If we do a single change in the previous query, the result is as shown in Fig. 3.16.

```
SELECT distinct T.branch_name
FROM branch T, branch
WHERE T.assets > S.assets
    AND T.branch_city = 'Pune';
```

branch_name
FC Road
MG Road

Fig. 3.16: Result of T.branch_city = 'Pune'

ORDER BY Clause: (Oct.-16)

- The ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

Syntax:

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

- You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

Example: Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

- Following is an example, which would sort the result in ascending order by NAME and SALARY:

```
SELECT * FROM CUSTOMERS
ORDER BY NAME, SALARY;
```

Output:

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
3	Kaushik	23	Kota	2000.00
2	Khilan	25	Delhi	1500.00
6	Komal	22	MP	4500.00
1	Ramesh	32	Ahmedabad	2000.00

- Following is an example, which would sort the result in descending order by NAME:

```
SELECT * FROM CUSTOMERS
ORDER BY NAME DESC;
```

Output:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
6	Komal	22	MP Delhi	4500.00
2	Khilan	25	Kota	1500.00
3	Kaushik	23	Bhopal	2000.00
5	Hardik	27	Mumbai	8500.00
4	Chaitali	25		6500.00

3.5.2 String Operations

- Database consists of some string values. The most commonly used operation on string is pattern matching. The operator used for pattern matching is LIKE.
- The patterns are case sensitive, i.e. uppercase characters do not match lowercase characters or vice versa.

- With the keyword LIKE, we can use two characters:
  1. Percent (%): It matches with any substring. It means that % stands for more number of characters.
  2. Underscore (_): It matches with any single character. It means _ stands for one character only.
- Consider following examples:
  1. "mad%": It matches with any string which begins with "mad". It may have characters after "mad".
     For example: madhav, madhuri, madam, madhura etc.
  2. "%mi%": It matches any string containing "mi" in it. The result may have characters before and after it.
     For example: amit, Sumit, Ashmit, Jasmit etc.
  3. "_ _ _": matches with any string which has exactly 3 characters in it.
     For example: jam, Ram, mat etc.
  4. "_ a _": It matches with any string which has exactly 3 characters and the middle character is 'a'.
     For example: Ram, bat, mat etc.
- Let's write queries for string operation.
- Select the row from branch table whose branch name starts with letter "M".

Query:

```
SELECT*
FROM branch
WHERE branch_name like 'M%';
```

- The result of this query is shown in Fig. 3.17.

branch_name	branch_city	assets
MG Road	Pune	500000

Fig. 3.17: Result of Like Query

- Select tuples from customer table whose name of customer has a character 'e' at third position.

Query:

```
SELECT*
FROM CUSTOMER
WHERE cust_name like "_ _ e%"
```

- The result is depicted in Fig. 3.18.

cust_name	cust_city	cust_street
Umesh	Pune	J M Road

Fig. 3.18: Result of Like Query

- Find names of all customers whose street address includes the substring 'FC'.

Query:

```
SELECT cust_name
FROM customer
WHERE cust-street like '%FC%';
```

- The result is shown in Fig. 3.19.

cust_name
Mugdha
Nitin

Fig. 3.19: Result of Like Query

- The SQL allows the specification of an escape character.
- The escape character is used immediately before a special pattern character to indicate that the special pattern character is to be treated as a normal character.
- We can define the escape character for a like comparison using the escape keyword.
- Let's consider backslash "\" as escape character in the following pattern.
- For example:
  1. `LIKE "ab\%cd%" Escape "\"`
     It matches all strings beginning with "ab%cd" i.e. ab%cdy, ab%cdd etc.
  2. `LIKE "ab\\cd%" Escape "\"`
     It matches all strings beginning with "ab\cd".

3.6 SET OPERATIONS

- SQL supports few set operations to be performed on table data. These are used to get meaningful results from data under different special conditions.
- Set operators combine the results of two or more component queries into one result. Queries containing set operators are called compound queries.
- Fig. 3.20 shows set operators used in set operations.

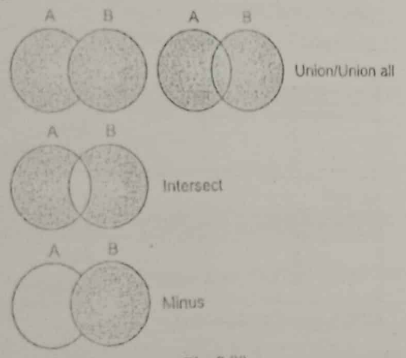

Fig. 3.20

1. UNION Operator: The UNION operator returns all rows that are selected by either query. Use the UNION operator to return all rows from multiple tables and eliminate any duplicate rows.
2. UNION ALL Operator: Use the UNION ALL operator to return all rows from multiple queries.
3. INTERSECT Operator: Use the INTERSECT operator to return all rows that are common to multiple queries.
4. MINUS Operator: Use the MINUS operator to return all distinct rows selected by the first query, but not present in the second query result set (the first SELECT statement MINUS the second SELECT statement).

1. UNION Operation:

- It is used to combine the results of two or more Select statements.
- However, it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables.

Syntax:

```
SELECT field1, field2, ..field_n
FORM tables
UNION
SELECT field1, field2, ..field_n
FORM tables;
```

Example: Take two tables as given below:

First

ID	Name
1	Abhi
2	Adam

Second

ID	Name
2	Adam
3	Chester

Union SQL query will be:

```
SELECT * FROM First
UNION
SELECT * FROM Second
```

ID	Name
1	Abhi
2	Adam
3	Chester

2. UNION ALL Operation:

- This operation is similar to Union. But it also shows the duplicate rows.

Syntax:

```
SELECT field1, field2, ..field_n FROM tables
UNION ALL
SELECT field1, field2, ..field_n FROM tables;
```

Example: Take following two tables:

First

ID	Name
1	Abhi
2	Adam

Second

ID	Name
2	Adam
3	Chester

Union All query will be like:

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

Output:

ID	Name
1	Abhi
2	Adam
2	Adam
3	Chester

3. INTERSECT Operation:

- Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements.
- In case of Intersect the number of columns and datatype must be same.

Syntax:

```
SELECT field1, field2, ..field_n FROM tables
INTERSECT
SELECT field1, field2, ..field_n FROM tables;
```

Example: Take following two tables:

First

ID	Name
1	Abhi
2	Adam

Second

ID	Name
2	Adam
3	Chester

Union All query will be like:

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

Output:

ID	Name
2	adam

4. **INTERSECT ALL Operation:**

- The INTERSECT ALL is equal to the INTERSECT command, except that INTERSECT ALL selects all repeated values.

Syntax:

```
SELECT field1, field2, ..field_n FROM tables
INTERSECT ALL
SELECT field1, field2, ..field_n FROM tables;
```

Example: Take two table given below:

EMP_Pune

E_NO	F_NAME
101	Smith
102	Jack
103	Smith
104	Ross
105	Smith

EMP_Satara

E_NO	F_NAME
101	Adam
102	Smith
103	Curry
104	Smith
105	William

INTERSECT ALL query look like:

```
SELECT f_name FROM EMP_Pune
INTERSECT ALL
SELECT f_name FROM EMP_Satara;
```


Output:

F_NAME
Smith
Smith

5. **MINUS Operation:**

- MINUS operation combines result of two SELECT statements and return only those result which belongs to first set of result.

Syntax:

```
SELECT field1, field2, ..., field_n
FROM  tables
[WHERE conditions]
MINUS
SELECT field1, field2, ...., field_n
FROM tables
[WHERE conditions];
```

Example: Take two tables as given below:

First

ID	Name
1	Abhi
2	Adam

Second

ID	Name
2	Adam
3	Chester

MINUS query will be like:

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

Output:

ID	Name
1	Abhi

3.7 AGGREGATE OPERATORS AND FUNCTIONS

3.7.1 Aggregate Functions

(Oct. 16, April 17)

- Aggregate means to sum. So actually aggregate function works on the whole database.
- These functions take a collection of values as inputs and return a single value. SQL offers 5 built-in aggregate functions:

1. **avg:** This function returns Average value.
2. **min:** This function returns minimum value.
3. **max:** This function returns maximum value.
4. **sum:** This function returns sum of numeric value.
5. **count:** This function returns the total number of values.

- These operations are called aggregate functions because they operate on aggregates of tuples.
- The result of aggregate function is a single value.
- The input to sum and avg must be a collection of numbers but the other operation can operate on collection of non-numeric datatypes such as string.
- Find average amount balance of FC Road branch.

Query:
```
SELECT avg (balance)
FROM account
WHERE branch_name = 'FC Road';
```

- Find total account balance in MG Road branch

Query:
```
SELECT sum(balance)
FROM account
WHERE branch_name = 'MG Road';
```

- All the balance value of FC Road are added and divided by the total number of values for FC Road. The result is shown in Fig. 3.21.

Expr 1000
10000

Fig. 3.21: Result of average

- Sometimes, we have to apply aggregate function not only to a single tuple but for set of tuples. This is specified using GROUP BY clause.

3.7.2 Aggregate Operators (GROUP BY and HAVING)

1. HAVING Clause:

- The HAVING clause enables you to specify conditions that filter which group results appear in the final results.
- The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax:

- The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used. The following is the syntax of the SELECT statement, including the HAVING clause:

```
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
HAVING [ conditions ]
ORDER BY column1, column2
```

- Example: Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

- Following is the example, which would display record for which similar age count would be more than or equal to 2:

```
SELECT ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
GROUP BY age
HAVING COUNT(age) >= 2;
```

Output:

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00

2. GROUP BY Clause:

- The GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups.
- The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

- The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

- Example: Consider the CUSTOMERS table is having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

- If you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

```
SELECT NAME, SUM(SALARY) FROM CUSTOMERS
GROUP BY NAME;
```

Output:

NAME	SALARY
Chaitali	6500.00
Hardik	8500.00
Kaushik	2000.00
Khilan	1500.00
Komal	4500.00
Ramesh	2000.00

- Now, let us have following table where CUSTOMERS table has the following records with duplicate names:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Ramesh	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Kaushik	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

- Now again, if you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

```
SELECT NAME, SUM(SALARY) FROM CUSTOMERS
GROUP BY NAME;
```

Output:

NAME	SALARY
Hardik	8500.00
kaushik	2000.00
Komal	4500.00
Ramesh	3500.00

3.8 DATE AND STRING FUNCTIONS

- In this section we study various date and string functions.

3.8.1 Date Functions,

- SQL provides a number of functions that return values related to the current date and time.

Sr. No.	Function	Return Type	Description
1.	AGE	INTERVAL	Calculate ages between current date (at midnight) and a timestamp and returns a "symbolic" result which uses years and months.
2.	CURRENT_DATE	DATE	Return the current date.
3.	CURRENT_TIME	TIMESTAMPTZ	Return the current time.
4.	CURRENT_TIMESTAMP	TIMESTAMPTZ	Return the current date and time with time zone at which the current transaction starts.
5.	DATE_PART	DOUBLE PRECISION	Get a field of a timestamp or an interval e.g., year, month, day, etc.
6.	DATE_TRUNC	TIMESTAMP	Return a timestamp truncated to a specified precision.
7.	EXTRACT	DOUBLE PRECISION	Same as DATE_PART() function.
8.	NOW	TIMESTAMPTZ	Return the date and time with time zone at which the current transaction start.

Contd...

9.	TIMEOFDAY	TEXT	Return the current date and time, like clock_timestamp, as a text string.
10.	TO_DATE	DATE	Convert a string to a date.
11.	TO_TIMESTAMP	TIMESTAMPTZ	Convert a string to a timestamp.

- SELECTNOW(); function shows function returns current date and time based on the database server's time zone setting.
- CURRENT_DATE:

```
SELECT CURRENT_DATE;
```

Result: 2019-08-23

- CURRENT_TIME.

```
SELECT CURRENT_TIME;
```

Result: 14:39:53.662522-05

- The TO_DATE function is used to converting strings into dates.

Syntax: TO_DATE(text, text) and the return type is date.

Examples:

27/08/2019	TO_DATE(date, 'DD/MM/YYYY')
27-08-2019	TO_DATE(date, 'DD-MM-YYYY')
08272019	TO_DATE(date', 'MMDDYY')
August 25, 2019	TO_DATE(date', 'Month DD, YYYY')

- The TO_TIMESTAMP() function converts string data into timestamps with timezone.

Syntax: to_timestamp(text, text).

```
AGE(timestamp, timestamp) function will Subtract arguments, producing
        a "symbolic" result  that uses years and months
AGE(timestamp '2001-04-10', timestamp '1957-06-13')
```

Result: 43 years 9 mons 27 days

- age(timestamp) function subtract from current_date (at midnight).
  For example: AGE(timestamp '1998-06-13')
- date_part(text, timestamp) function get subfield (equivalent to extract).
  For example: date_part('hour', timestamp '2019-02-16 20:38:40')
  Result: 20
- date_trunc(text,timestamp) function truncate to specified precision.
  For example: date_trunc('hour', timestamp '2019-02-16 20:38:40')
  Result: 20019-02-16 20:00:00

3.8.2 String Functions

- SQL provides functions and operators for examining and manipulating string values. Strings in this context include values of the types character, character varying, and text.
- The most commonly used PostgreSQL string functions that allow you to manipulate string data effectively.

Sr. No.	Function	Description	Example	Result
1.	ASCII	Return the ASCII code value of a character or Unicode code point of a UTF8 character.	ASCII('A')	65
2.	CHR	Convert an ASCII code to a character or a Unicode code point to a UTF8 character.	CHR(65)	'A'

Contd...

3.	CONCAT	Concatenate two or more strings into one.	CONCAT('A','B','C')	'ABC'
4.	CONCAT_WS	Concatenate strings with a separator.	CONCAT_WS(',', 'A', 'B', 'C')	'A,B,C'
5.	FORMAT	Format arguments based on a format string.	FORMAT('Hello %s', 'PostgreSQL')	'Hello PostgreSQL'
6.	INITCAP	Convert words in a string to title case.	INITCAP('hI tHERE')	Hi There
7.	LEFT	Return the first n character in a string.	LEFT('ABC',1)	'A'
8.	LENGTH	Return the number of characters in a string.	LENGTH('ABC')	3
9.	LOWER	Convert a string to lowercase.	LOWER('hI tHERE')	'hi there'
10.	LPAD	Pad on the left a a string with a character to a certain length.	LPAD('123', 5, '00')	'00123'
11.	LTRIM	Remove the longest string that contains specified characters from the left of the input string.	LTRIM('00123')	'123'
12.	MD5	Return MD5 hash of a string in hexadecimal.	MD5('ABC')	
13.	POSITION	Return the location of a substring in a string.	POSTION('B' in 'A B C')	2
14.	REGEXP_MATCHES	Match a POSIX regular expression against a string and returns the matching substrings.	SELECT REGEXP_MATCHES ('ABC', '^(A)(..)$', 'g');	{A,BC}
15.	REGEXP_REPLACE	Replace substrings that match a POSIX regular expression by a new substring.	REGEXP_REPLACE('John Doe','(.*) (.*)','\2, \1');	'Doe, John'
16.	REPEAT	Repeat string the specified number of times.	REPEAT('*', 5)	'*****'
17.	REPLACE	Replace all occurrences in a string of substring from with substring to.	REPLACE('ABC', 'B', 'A')	'AAC'
18.	REVERSE	Return reversed string.	REVERSE('ABC')	'CBA'
19.	RIGHT	Return last n characters in the string. When n is negative, return all but first |n| characters.	RIGHT('ABC', 2)	'BC'
20.	RPAD	Pad on the right of a string with a character to a certain length.	RPAD('ABC', 6,'xo')	'ABCxox'
21.	RTRIM	Remove the longest string that contains specified characters from the right of the input string.	RTRIM('abcxxzx', 'xyz')	'abc'

Contd...

22.	SPLIT_PART	Split a string on a specified delimiter and return nth substring.	SPLIT_PART('2017-12-31','-',2)	'12'
23.	SUBSTRING	Extract a substring from a string.	SUBSTRING('ABC',1,1)	'A'
24.	TRIM	Remove the longest string that contains specified characters from the left, right or both of the input string.	TRIM(' ABC ')	'ABC'
25.	UPPER	Convert a string to uppercase.	UPPER('hI tHERE')	'HI THERE'

- For example, the substring() function is used to extract a string containing a specific number of characters from a particular position of a given string.
  Syntax: substring(string [from <str_pos>] [for <ext_char>])
  For example: SELECT substring('POSTGRESSQL'from6for3);
  Result: RES
  For example: SELECT UPPER('PostgreSql');
  Result: POSTGRESQL

3.9 NULL VALUES

- NULLs are the special markers by using which SQL systems represent missing information. For example, if we say that the weight of a person is null. Then it means that, the person is exists-live otherwise he or she does not have it. But we do not known how much it is.
- In other words we do not know the actual weight of that person which we can sensibly placed in the weight column.
- Therefore, we mark it as NULL. Using NULL keyword we clear a point to the database that the null is not a value and not blank or zero at the same time.
- The representation of NULL is implementation dependent. But SQL standard specify that we should have a representation where null and not null values are distinguishable.
- Definition of NULL: A NULL value is an implementation dependent special value which is distinct from all null values and there is effectively one and only one value.
- In general, if we do not explicitly specify a column as NOT NULL, the column has nulls. If a column is allowed to contain nulls then RDBMS automatically place a null into that column.
- This is possibly done by RDBMS in the situation where a row is inserted into a table and no values are specified for that column.
- Similarly, if we write NOT NULL for a particular column name then we are not allowed the update a column again. If we use a option "NOT NULL WITH DEFAULT", then it means that the column can contain nulls.Still it is possible to omit the values for the column during INSERT or UPDATE.
- If we insert a row and no values are specified for that column then the RDBMS will insert one of the default given below:
  1. Zero if a numeric column.
  2. Blank if a fixed-length string column.
  3. Empty if a varying length column.
- The INSERT command is used to insert a value given for every attribute. Lets write a query using null values.
- Insert a tuple into Custome tabler without Salary.
  Query:

```
INSERT INTO CUSTOMER
VALUES (6,'Komal' 22,'MP' ,NULL);
```

- It will insert the value in Customer table with NULL as salary.

3.9.1 Comparison using NULL Values

- Consider the following table, CUSTOMERS having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	

- Now, following is the usage of IS NOT NULL operator:

```
SELECT ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
WHERE SALARY IS NOT NULL;
```

Output:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00

- Now, following is the usage of IS NULL operator:

```
SELECT ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
WHERE SALARY IS NULL;
```

Output:

ID	NAME	AGE	ADDRESS	SALARY
6	Komal	22	MP	

3.9.2 Logical Connectives AND, OR and NOT

- Generally, to connect or compare logical conditions we use AND, OR and NOT keywords.
- But once we have null values, we must define these logical operators using a three-valued logic, where the expression evaluated as true, false or unknown.
- For example:

```
SELECT cust_name
FROM customer, account
WHERE cust_name = 'Sayali'
AND acc_no < 1000;
```

- In this case both conditions should be true then only the result of AND is true. Similarly, for OR any one condition has to true then only result is true else it is false.

3.9.3 Impact on SQL Constructs

- When we use Boolean expression in SQL then the impact of null values must be recognized. To check this impact on nested queries we use keyword EXISTS or UNIQUE.
- The arithmetic operators +, –, *, / returns null if any one argument is null. Similarly, null cause the problems with aggregate function such as COUNT, SUM, AVG, MIM, MAX etc.
- Even comparison of two null values give a unknown result.

3.9.4 Disallowing NULL Values

- We can disallow null values by specifying NOT NULL while defining the column in a table.
- A field which have a null value should not be considered as a primary key.
- For example:

```
rollno char (20) NOT NULL;
```

3.10 NESTED SUB QUERIES

- A query within a query is called nested query. Sometimes, it is referred as nested sub-query. A sub-query is a SELECT-FROM-WHERE expression that is nested within another query.
- Sub-query can be defined as "a query within a query". A sub-query can use values from the outer query, in which case it is known as a correlated sub-query.
- A sub-query or inner query or nested query is a query within another SQL query and embedded within the WHERE clause.
- A sub-query is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- Sub-queries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

Sub-queries with the SELECT Statement:

- Sub-queries are most frequently used with the SELECT statement. The basic syntax is as follows:

```
SELECT column_name [, column_name ]
FROM  table1 [, table2 ]
WHERE column_name OPERATOR
(SELECT column_name [, column_name ]
FROM table1 [, table2 ]
[WHERE])
```

- Example: Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

- Now, let us check following sub-query with SELECT statement:

```
SELECT *
FROM CUSTOMERS
WHERE ID IN (SELECT ID
FROM CUSTOMERS
WHERE SALARY > 4000) ;
```

Output:

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

Sub-queries with the INSERT Statement:

- Sub-queries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the sub-query can be modified with any of the character, date or number functions.
- The basic syntax is as follows:

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
SELECT [ *|column1 [, column2 ]
FROM table1 [, table2 ]
[ WHERE VALUE OPERATOR ]
```

Example:

- Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy complete CUSTOMERS table into CUSTOMERS_BKP, following is the syntax:

```
INSERT INTO CUSTOMERS_BKP
SELECT * FROM CUSTOMERS
WHERE ID IN (SELECT ID
FROM CUSTOMERS) ;
```

Sub-queries with the UPDATE Statement:

- The sub-query can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.
- The basic syntax is as follows:

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[ WHERE) ]
```

Example:

- Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table.
- Following example updates SALARY by 0.25 times in CUSTOMERS table for all the customers whose AGE is greater than or equal to 27:

```
UPDATE CUSTOMERS
SET SALARY = SALARY * 0.25
WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
WHERE AGE >= 27 );
```

- This would impact two rows and finally CUSTOMERS table would have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	125.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	2125.00
6	Komal	22	MP	4500.00

Sub-queries with the DELETE Statement:

- The sub-query can be used in conjunction with the DELETE statement like with any other statements mentioned above.
- The basic syntax is as follows:

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[ WHERE) ]
```

Example:

- Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table.
- Following example deletes records from CUSTOMERS table for all the customers whose AGE is greater than or equal to 27:

```
DELETE FROM CUSTOMERS
WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
WHERE AGE > 27 );
```

- This would impact two rows and finally CUSTOMERS table would have the following records:

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
6	Komal	22	MP	4500.00

3.11 SQL MECHANISMS FOR JOINING RELATIONS

- PostgreSQL JOINS are used to retrieve data from multiple tables. A PostgreSQL JOIN is performed whenever two or more tables are joined in a SQL statement.
- There are different types of PostgreSQL joins:
  1. INNER JOIN (or sometimes called simple join).
  2. LEFT OUTER JOIN (or sometimes called LEFT JOIN).
  3. RIGHT OUTER JOIN (or sometimes called RIGHT JOIN).
  4. FULL OUTER JOIN (or sometimes called FULL JOIN).

3.11.1 INNER JOIN (Simple Join)

- The INNER JOIN selects all rows from both participating tables as long as there is a match between the columns. An SQL INNER JOIN is same as JOIN clause, combining rows from two or more tables.

  Syntax:

```
SELECT * FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
            OR
SELECT * FROM table1
JOIN table2
ON table1.column_name = table2.column_name;
```

- JOIN returns all rows from tables where the key record of one table is equal to the key records of another table.

- The INNER JOIN selects all rows from both participating tables as long as there is a match between the columns. The SQL INNER JOIN is same as JOIN clause, combining rows from two or more tables.
- An inner join of table1 and table2 gives the result of table1 intersect table2, i.e. the inner part of a Venn diagram intersection.

Fig. 3.22: INNER JOIN

3.11.2 OUTER JOIN (April 15, 17; Oct. 17)

- The important variant of join operation which relies on null values, called outer joins.
- If we want outer join there we can write a query as follows:

```
SELECT cust_name, loan-loan_no
FROM LOAN NATURAL LEFT OUTER JOIN BORROWER;
```

- In LEFT OUTER JOIN, loan rows are appeared into a result where the loan rows are not matched with borrowers rows.
- But not vice versa. In RIGHT OUTER JOIN, borrower rows are appeared into a result without matching with loan table.
- In FULL OUTER JOIN, both the rows appeared in the result without a match.

3.11.2.1 FULL OUTER JOIN (Oct. 17)

- This is a type of OUTER JOIN is called a FULL OUTER JOIN. This type of join returns all rows from the LEFT-hand table and RIGHT-hand table with nulls in place where the join condition is not met.

Syntax:

```
SELECT columns
FROM table1
FULL OUTER JOIN table2
ON table1.column = table2.column;
```

- In this FULL OUTER JOIN diagram, the SQL returns the shaded area, (Refer Fig. 3.23).

Fig. 3.23: FULL OUTER JOIN

3.11.2.2 LEFT OUTER JOIN

- Another type of outer join is called a LEFT OUTER JOIN. This type of join returns all rows from the LEFT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal (join condition is met).

Syntax:

```
SELECT columns
FROM table1
LEFT OUTER JOIN table2
ON table1.column = table2.column;
```

- In this LEFT OUTER JOIN diagram, the SQL returns the shaded area (Refer Fig. 3.24).

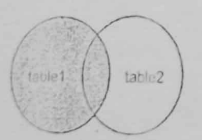

Fig. 3.24: LEFT OUTER JOIN

- The LEFT OUTER JOIN would return the all records from table1 and only those records from table2 that intersect with table1.

3.11.2.3 RIGHT OUTER JOIN

- Another type of outer join is called the RIGHT OUTER JOIN. This type of join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal (join condition is met).

Syntax:

```
SELECT columns
FROM table1
RIGHT OUTER JOIN table2
ON table1.column = table2.column;
```

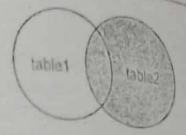

Fig. 3.25: RIGHT OUTER JOIN

- In this diagram, the RIGHT OUTER JOIN returns the shaded area (Refer Fig. 3.25).
- The RIGHT OUTER JOIN would return the all records from table2 and only those records from table1 that intersect with table2.

3.12 VIEWS

- A view is named query that provides another way to present data in the database tables. A view is defined based on one or more tables, which are known as base tables.
- When you create a view, we basically create a query and assign it a name, therefore a view is useful for wrapping a commonly used complex query. Note that a normal view does not store any data except the materialized view.
- We can manage views such as creating, modifying, and removing views from the database.

3.12.1 Creating Views

- To create a view, we use CREATE VIEW statement. The simplest syntax of the CREATE VIEW statement is as follows:

```
CREATE VIEW view_name AS query;
```

- For example: `CREATE VIEW Cust AS Select * from Customer where Cust_city='Pune';`
- Now cust view is,

cust_name	cust_city	cust_street
Umesh	Pune	J M Road
Mugdha	Pune	F C Road
Nitin	Pune	F C Road
Paresh	Pune	M G Road
Rupali	Pune	M G Road

- Whenever, you need to get a complete customer data, you just query it from the view by executing the following simple SELECT statement.

```
SELECT  cust_name
FROM  Cust
Where cust_street ='M G Road';
```

Result: Paresh, Rupali

3.12.2 Changing Views Query

- To change the defining query of a view, you use the CREATE VIEW statement with OR REPLACE addition as follows:

```
CREATE OR REPLACE    view_name  AS query;
```

- For example: `CREATE VIEW Cust AS SELECT cust_name, cust_city FROM Customer ;`

3.12.3 Modifying Views

- To change the definition of a view, we can use the ALTER VIEW statement.
- For example: You can change the name of the view from Cust to CustData.

```
ALTER VIEW Cust RENAME TO CustData;
```

3.12.4 Removing Views

- SQL allows us to delete an existing View. We can delete or drop a View using the DROP statement.

Syntax: `DROP VIEW [IF EXISTS] view_name`

- To remove an existing view in PostgreSQL, you use DROP VIEW statement as follows:

```
DROP VIEW CustData;
    OR
DROP VIEW IFEXISTS CustData;
```

3.13 SOLVED CASE STUDIES ON SQL

Case Study 1: Consider the following entities and relationships.

Owner (Licence_no, name, address, phone)

Car (carno, model, colour)

Owner and Car are related with one-to-many relationships.

Create a RDB for the above and solve the following queries in Postgre SQL.

Solution: Owner and car are the two relations given. Lets draw an E-R diagram with the given relationships. The E-R diagram is shown in Fig. 3.26.

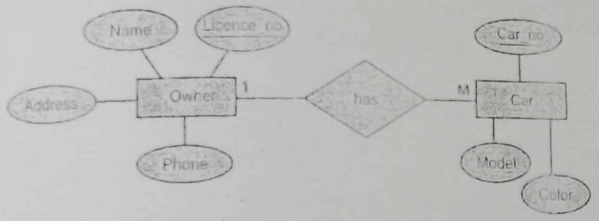

Fig. 3.26: E-R Diagram

After drawing E-R diagram, we have to convert it into the form of table with the given relationship. The relationship is one-to-many, the primary key of relation one (called parent) is taken and added as a foreign key into the many (called child) relation. Lets draw on RDB which is shown in Fig. 3.27.

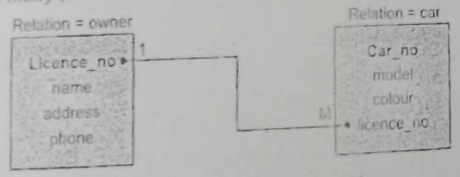

Fig. 3.27: One-to-Many Relationship

After this we should add some tuples into the table which will answer the written queries.

Let's write the queries.

Query 1: Find the name of owner of 'Zen' and 'Indica' cars.

```
SELECT name, address, model
FROM car AS c, owner AS o
WHERE (model='zen' Or model='indica') And c.car_no=o.car_no;
```

Query 2: Insert a record in a car relation.

```
INSERT INTO car
VALUES (154, 'Maruti', 'BLACK');
```


Query 3: List all the models of owner 'Mr. Shah' having colour 'blue'.

```
SELECT[model,[colour
FROM car AS c, owner As o
WHERE name='mr shah' and colour='blue' and c.car_no=o.car_no;
```

Query 1: To list the information of all cars in 'Pune'.

```
SELECT name, address,model, colour
FROM car AS c, owner As o
WHERE c.carno=o.carno and address='pune'
GROUP BY name, address,model, colour;
```

Case Study 2: Consider the following entities and relationships: Oct. 17

Machine (m_no, m_name, m_type, m_cost)

Part (p_no, p_name, p_desc)

Machine and Part are related with one-to-many relationships.

Create a RDB for the above and solve the following queries in PostgreSQL.

Solution: The one-to-many relationship is as shown in Fig. 3.28.

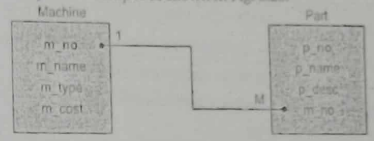

Fig. 3.28: One-to-Many

Queries are:

Query 1: Increase the cost of machine by 5%.

```
UPDATE machine SET m_cost = m_cost + (m_cost * 0.05);
```

Query 2: Delete all machine s having particulars "wheel".

```
DELETE m_type
FROMMachine
WHERE m_type="wheel";
```

Query 3: List all the machines whose cost > 1,00,000.

```
SELECTm_name, m_cost
FROM machine
WHERE m_cost>100000;
```

Case Study 3: Consider the following entities and relationships:

Customer (cust-no, Name, addr)

Quotation (quot_no, desc, Amt_quoted)

Customers and quotations are related with one-to-many relationships.

Create a RDB for the above and solve the following queries in postgreSQL.

Solution: The RDB is as shown in Fig. 3.29.

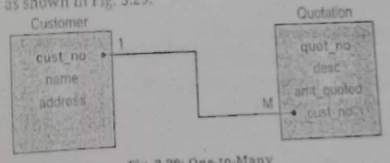

Fig. 3.29: One-to-Many

Queries are:

Query 1: List all the customer who are demanding for the quotation of "washing machine".

```
SELECT name,addr
FROM customer, quotation
WHERE desc="washing machine" AND customer.cust_no=quotation.cust_no;
```

Query 2: List all the customer who are demanding for the quotation of "fridge".

```
SELECT name, addr
FROM customer, quotation
WHERE desc="Fridge" AND customer.cust_no=quotation.cust_no;
```

Query 3: Delete all the customer with details with address 'pune'.

```
DELETE *
FROM customer
WHERE addr="pune";
```

Query 4: To print customer wise list of quotation but only those quotations whose amt > 5000.

```
SELECT name, customer.addr, desc, amt_quoted
FROM customer, quotation
WHERE amt_quoted>5000 AND customer.cust_no=quotation.cust_no;
```

Case Study 4: Consider the following entities and relationships: Oct. 17

Company (C_product, C_name, region, state)

Branches (b_product, city)

Company and Branches are related with one-to-many relationships.

Create a RDB for the above and solve the following queries in Postgre SQL.

Solution: Here we cannot consider cproduct and bproduct as primary keys because product name can be duplicate. So lets add one more field called c_id into relation company which will save as primary key. The one-to-many relationship is as shown in Fig. 3.30.

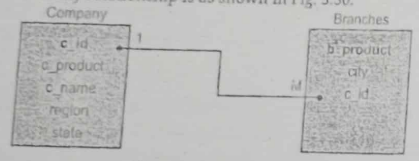

Fig. 3.30: One-to-Many

Let's write queries:

Query 1: List all the cities having branch product 'H_2SO_4' and 'SO_3'

```
SELECT city, b_product
FROM branches AS b, company AS c
WHERE b.c_id=c.c_id and (b_product='H2SO4' or b_product='SO3');
```

Query 2: List all the states whose branch product is 'oleum'.

```
SELECT state
FROM branches As b, company AS c
WHERE b.c_id=c.c_id and (b-product='oleum');
```

Query 3: Delete all the company details of city 'baroda'.

```
DELETE c_id
FROM branches
WHERE city='baroda';
```


Query 4: Print city wise branches in shorted order.

```
SELECT cname as company, city
FROM branches AS b, company AS c
WHERE b.c_id=c.c_id
ORDER BY cname;
```

Case Study 5: Consider the following entities and relationships:

Teacher (t_no, t_name, college_name, dept)

E-Test(e_no, test_name)

Teacher and E_test are related with many-to-many relationships.

Create a RDB for the above and solve the following queries in postgre SQL.

Solution: The relationship is many-to-many. So, we have to create a new table with the primary keys. The RDB is as shown in Fig. 3.31.

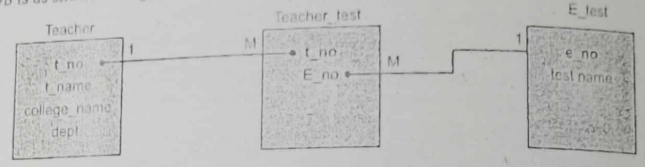

Fig. 3.31: Many-to-Many Relationship

Lets write the queries.

Query 1: Give the name of teachers who have passed either "SET" or "NET".

```
SELECT t_name, college_name, test_name, dept
FROM teacher AS t, e_test AS e, relate AS r
WHERE r.t_no=t.t_no and r.e_no=e.e_no AND
(test_name='SET' or test_name='NET');
```

Query 2: Count the no. of teachers who passed 'SET' exam of 'computer science'.

```
//Query to find the teachername of computer science dept.//
SELECT t_no, t_name
FROM teacher
WHERE dept='COMPUTER SCI';
SELECTt_name
FROM relate, query2a, e_test
WHERE teacher.t_no=relate.t_no
      AND e_test.e_no=relate.e_no And test_name='SET';
```

Query 3: Delete all the teachers details of 'Physics' dept.

```
DELETE tname
FROM teacher
WHERE dept='Physics';
```

Query 4: To list exam wise list of teachers who have passed the respective exams.

```
SELECT test_name, t_name, college_name, dept
FROM teacher AS t, e_test AS e, relate AS r
WHERE r.t_no=t.t_no and r.e_no=e.e_no
GROUP BYtest_name, t_name, college_name, dept;
```

Query 5: To print the total number of teachers passing the respective exam.

```
SELECT test_name, count (*) AS No_of_teacher
FROM teacher AS t, e_test AS e, relate AS r
WHERE r.t_no=t.t_no and r.e_no=e.e_no
GROUP BY test_name;
```

Case Study 6: Consider the following entities and relationships: Oct.-17

Country (con_code, name, capital)

Population (Pop_code, population)

Country and Population are related with one-to-one relationships.

Create a RDB for the above and solve the following queries postgre SQL.

Solution: The relationship given is one-to-one. There is no need to create a new table or take primary key and add it to other table. Here, primary keys of both tables are directly related with each other. This is shown in Fig. 3.32.

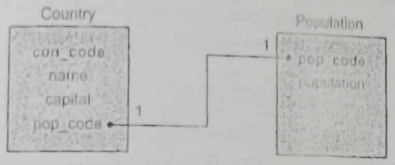

Fig. 3.32: One-to-One Relationship

Queries are:

Query 1: List the highest population country.

```
// Query to find the max population //
SELECT max (population) As maxi
FROM population;
// Query to find the names of country with max population
SELECT name, capital
FROM country AS c, Query1a, population As p
WHERE P.POP_CODE=C.POP_CODE And population=maxi;
```

Query 2: Give the name and population of country whose capital is 'Tokyo'.

```
SELECT name, population
FROM population as p, country as c
WHERE P.pop_code=c.pop_code and capital='Tokyo';
```

Query 3: Give the nameof all the countries whose population is greater than 50,00,000.

```
SELECT name,capital, population FROM population as p, country as c
WHERE P.pop_code=c.pop_code and population>5000000;
```

Query 4: To print country wise population.

```
SELECT name, population
FROM population as p, country as c
WHERE P.pop_code=c.pop_code
GROUP BY name, population;
```

Case Study 7: Consider the following entities and relationships:

Wholesaler (w_no, w_name, addr, city)

Product (p_no, p_name)

Wholesaler and Product are related with many-to-many relationships.

Create a RDB for the above and solve the following queries in Postgre SQL.

Solution: Many-to-many relationship allows us to create new table. The RDB is as shown in Fig. 3.33.

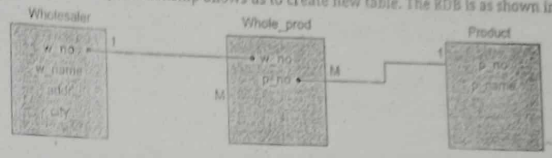

Fig. 3.33: Many-to-Many

Query 1: List all the wholesaler of product "books".

```
SELECT w_name, city
FROM wholesaler as w, product as p, relate as r
WHERE w.w_no=r.w_no and p.p_no=r.p_no and p_name= "books";
```

Query 2: Count the no of wholesaler in the city "ahmednagar".

```
SELECT COUNT (*) as wholesaler_from_ahmednagar
FROM wholesaler
WHERE city="ahmednagar";
```

Query 3: Insert a record in wholesaler relation.

```
INSERT INTO wholesaler
VALUES (111, "ajay", "ghajgajs", "ahmednagar");
```

Query 4: To print wholesalerwise product.

```
SELECT DISTINCT w_nameAS wholesaler, city, p_nameAS
product
FROM wholesaler as w, product as p, relate as r
WHERE w.w_no=r.w_no and p.p_no=r.p_no;
```

Case Study 8: Consider the following entities and relationships:

Politician (p_no, p_name, p_desc, constituency)

Party (partycode, partyname)

Politician and party are related with many-to-one relationships. Create a RDB for the above and solve the following queries in Postgre SQL.

Solution: The relationship is many-to-one. Therefore, the primary key of one relation is taken as foreign key in many table. The relationship is shown in Fig. 3.34.

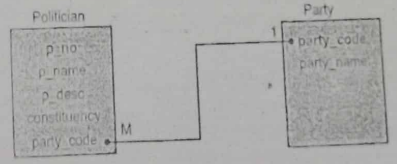

Fig. 3.34: Many-to-One

Queries are:

Query 1: List all the politicians of the party 'NCP'.

```
SELECT Politician.name
FROM Party, Politician
WHERE Party_name='NCP' AND politician.party_code=party.party_code;
```

Query 2: Count the no of politicians having political description as a 'member of parliament'.

```
SELECT count (name) AS Total_MPs
FROM Politician
WHERE description='MEMBER OF PARLIAMENT';
```

Query 3: Give the party name of politician 'Dr. Amol Kholhe'.

```
SELECT Party_name
FROM Party, politician
WHERE Politician.name='Dr.Amol Kholhe'
    AND Politician.party_code=Party.party_code;
```

Query 4: List party wise list of politicians of 'pune' constituency.

```
SELECT politician.name, party.party_name, politician,constituency.
FROM party, politician
WHERE politician.constituency='pune' AND party.party_code=politician.party_code;
```

Query 5: Print the total no. of politician in each party for 'pune' constituency.

```
SELECT count (name) AS no_of_politician, party_name
FROM party, politician
WHERE party.party_code=politician.party_code AND politician.Constituency='pune'
GROUP BY party.party_name;
```

Case Study 9: Consider the following entities and relationships: **Oct-17**

Game (g_no, gname, no_of_Players, coach_name, captain)

Player (P_no, P_name)

Game and Players are related with many-to-many relationships.

Create a RDB for the above and solve the following queries in Postgre SQL.

Solution: The relationship is many-to-many so create a new table which is shown in Fig. 3.35.

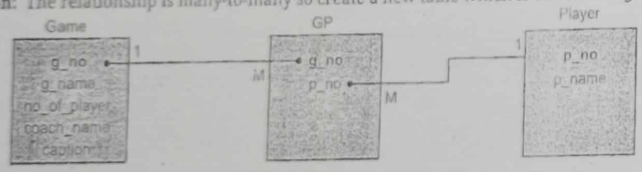

Fig. 3.35: Many-to-Many

Queries are:

Query 1: List the name of players playing 'basketball' and 'handball'.

```
SELECT p_name, g_name
FROM game as g, player as p, relate as r
WHERE p.p_no=r.p_no and g.g_no=r.g_no AND
    (g_name='basket' or g_name='handball');
```

Query 2: List the name of players playing game 'cricket'.

```
SELECT p_name as cricket_player
FROM game as g, player as p, relate as r
WHERE p.p_no=r.p_no and g.g_no=r.g_no AND (g_name= 'cricket');
```

Query 3: Count the total no. of players whose coach name is 'mr. sharma'

```
SELECT COUNT (*) as no_of_player
FROM game as g, player as p, relate as r
WHERE p.p_no=r,p_no and g.g_no=r.g_no AND (coach_name='mr sharma');
```


Query 4: List the game wise player list.

```
SELECT g_name as game, p_name as player
FROM game as g, player as p, relate as r
WHERE p.p_no=r.p_no and g.g_no=r.g_no
GROUP BY g_name, p_name
```

Case Study 10: Consider the following entities and relationships:

Item (I_no, I_name, I_qty,I_cost)

PO (P_no, P_date)

Supplier (S_no, S_name, S_addr)

Item and PO are related with one-to-many relationships along with descriptive cost and quantity. Supplier and PO are related with one-to-many relationship.

Create a RDB for the above and solve the following queries in postgre SQL.

Solution: The RDB (relational database) is shown in Fig. 3.36.

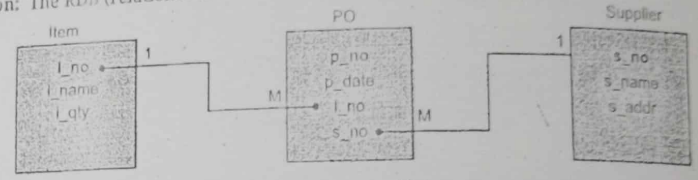

Fig. 3.36: One-to-Many

Queries are:

Query 1: List the name of supplier to whom po is given for "mouse".

```
SELECT s_name, i_qty
FROM po, item AS I, supplier AS S
WHERES.s_no=po.s_no And po.i_no-I.i_no And I. i_name="mouse";
```

Query 2: List the name of supplier and item_name in pos generated on "3-sept-2019".

```
SELECT S_NAME,i_NAME
FROM po, item AS I, supplier AS  S
WHERES.s_no-po.s_no And po.i_no=I .i_no AND p_date=' 3-sept-2019';
```

Query 3: List the names of suppliers who is going to supply "monitor" with minimum cost.

```
// To select the minimum cost for the monitor //
SELECT min(I_cost) AS mincost
FROM  po, item
WHERE po.i_no=po.i_no And i_name="Monitor";
// TO LIST THE NAME OF SUPPLIER//
SELECT s_name
FROM po, supplier
WHERE supplier.s_no = po.s-no AND I_cost in (SELECT min(I_cost) AS mincost
FROM  po, item
HAVINGpo.i_no=po.i_no And  i_name="Monitor");
```

Query 4: Find out po number, po date and supplier name of the po which is of maximum amount.

```
SELECT p_no, p_date,s_name
FROM po, supplier
WHERE po.s_no=supplier.s_no AND cost in(select max(cost) from po);
```

Query 9: Display all the po which contains the number, date, supplier name of the po details of all items included in that i.e. name of item, qty and rate).

```
SELECT p_no; p_date, s_name, i_name, qty, cost
FROM po, supplier, item
WHERE po.s_no=supplier.s_no AND item.i_no=po.i_no;
```

Case Study 11:Consider the following relational database:

lives (person_name, street, city)
works (person_name, company_name, salary)
located_in (company_name, city)
manager (person_name, manager_name)
Write the SQL statements for each of the following.

Solution: Queries are:

Query 1: Find all employees who have more than the average salary of employee in their company.

```
SELECT distinct W.person_name
FROM works W
WHERE W.salary >= (SELECT avg (t.salary)
    FROM works t
    WHERE t.company_name=w.company_name);
```

Query 2: Find the company with least number of employees.

```
SELECT t.company_name
FROM works t
GROUP BY t.company_name
HAVING count (t.person_name) <= all
  (SELECT count (S.person_name)
  FROM works S
  GROUP BY S.company_name);
```

Case Study 12:Consider the following relational database:

Sailors (sid, sname, rate, age)
Boats (bid, bname, colour)
Reserves (sid, bid, day)
Write SQL statement for each of the following queries.

Solution: From the given database, we can directly recognize that the relationship between sailors and boats is many-to-many i.e. a new table called Reserves is created by the primary keys of both tables. Therefore, the Relational database diagram is as shown in Fig. 3.37.

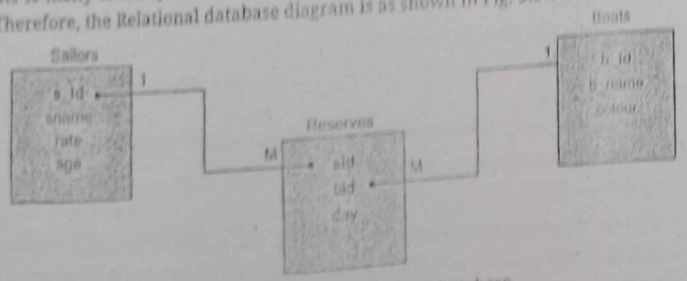

Fig. 3.37: Many-to-Many Sailors Database

Lets write queries on the above database.

Query 1: Find names and ages of all sailors.

```
SELECT DISTINCT Sname, age
FROM sailors;
```

Query 2: Find all the sailors with a rating above 8.

```
SELECT *
FROM sailors
where rate > 7;
```

Query 3: Find the names of sailors who have reserved boat no. 103.

```
SELECT S.sname
FROM sailors AS S.Reserves AS R
WHERE S.sid=R.sid
AND R.bid=103;
```

Query 4: Find the Sids of sailors who have reserved a green boat.

```
SELECT R.sid
FROM Boats As B, Reserves R
WHERE B.bid=R.bid
  AND B.colour="green";
```

Query 5: Find colours of boats reserved by Ashmit.

```
SELECT B.colour
FROM sailors AS S, boats AS B, Reserves AS R
WHERE S.sid=R.sid
    AND R.Bid=B.bid
    AND S.Sname="Ashmit";
```

Query 6: Find the names of sailors who have reserved atleast one boat.

```
SELECT S.Sname
FROM sailors As S, Reserver AS R
WHERE S.sid=R.sid;
```

Query 7: Find the ages of sailors whose name begins and ends with P.

```
SELECT S.age
FROM sailors as S
WHERE S.sname like "P%P";
```

The name may be pop, P... P where ...means lot of characters considered in place of %.

Query 8: Find name of sailors who have reserved red and green boat.

```
SELECT S.sname
FROM sailors as"S, Reserves as P, boats as B
WHERE S.sid=R.sid AND R.bid=B.bid
AND (B.colour="red" OR B.colour="green");
```

Case Study 13:Consider the following relational database:

Doctor (D_no, D_name, address, city)
Hospital (H_no, name, street, city)
Many doctors are working into many hospitals. Doctors visits to hospital on certain date. Create a RDB and answer following queries.

Solution: Many to many relationships allows us to create a new table in which we will consider date as one of the attributes. The RDB is as shown in Fig. 3.38.

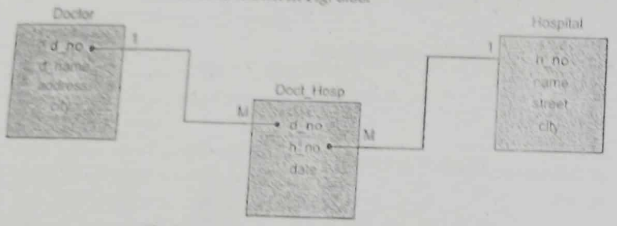

Fig. 3.38: Many-to-Many with Attribute Date

Queries into SQL are:

Query 1: Count number of doctors visited to "Ruby Hospital" on 1st March 2003.

```
SELECT COUNT (distinct d_no)
FROM Hospital, doct_hosp
WHERE doct_hosp.H_no=Hospital.H_no
AND name = "Ruby Hospital"
AND date = "01-MAR-03";
```

Query 2: Find out how many times 'Dr. Sathe' visited to Ruby Hospital.

```
SELECT COUNT (d_no)
FROM doctor AS d, hospital as h, doct-hosp AS m
WHERE D.dname = "Dr. Sathe"
  AND H.name = "Ruby Hospital"
  AND D.dno = m.dno
  AND H.Hno = m.hno;
```

Query 3: List the doctors in city pune.

```
SELECT *
FROM Doctor
WHERE city = "pune";
```

Case Study 14: Consider the following database:

Employee

Department

Dept-Locations

Project

Works on

Dependent

Fig. 3.39

Queries are:

Query 1: Retrieve the birthdate and address of the employee whose name is "John B. Smith".

```
SELECT B_date, Addr
FROM employee
WHERE F_name="John" AND MINIT="B" AND  L_name="Smith";
```

Query 2: Retrieve the name and address of all employees who work for the research department.

```
SELECT F_name, L_name, Addr
FROM Employee, Department
WHERE D_name="Research" AND D_number=DNO;
```

Query 3: For every project located in "Pune", list the project number and controlling dept-number and department manager's name.

```
SELECT P_number, D_num, L_name
FROM Project, Department, Employee
WHERE D_num=D_number AND
mgr_Aadhar_no=Aadhar_noand P_location="Pune";
```

Query 4: For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
SELECT E.F_name, E.L_name, S.F_name, S.L_name
FROM Employee E, Employee S
WHERE E.superSS N=S.AADHAR_NO;
```

Query 5: Select all employee AADHAR_NO's.

```
SELECT AADHAR_NO
FROM Employee;
```

Query 6: Retrieve all the attribute values of employer tuples who work in dept-number 5.

```
SELECT*
FROM Employee
WHERED_NO=5;
```

Query 7: Retrieve the salary of every employee.

```
SELECT salary "
FROM Employee;
```

If we are interested only in distinct salary values then we write,

```
SELECT distinct salary
FROM Employee;
```

Query 8: Find the list of all project numbers for projects that involve an employee whose last name is "Smith", either as a worker or as a manager of the department that controls the project.

```
SELECT P_number
FROM Project, Department, Employee
```

```
WHERE D_num=D_number and Mgr_AADHAR_NO=AADHAR_NO
AND L_name="Smith"
UNION
SELECT P_number
FROM Project, works_on, Employee
WHERE P_number=PNO and EAADHAR_NO=AADHAR_NO
AND L_name="Smith";
```

Query 9: Retrieve the names of employees whose salary is greater than the salary of all the employees in department 5.

```
SELECT L_name, Fname
FROM Employee
WHERE salary > all
(SELECT salary from Employee
WHERE D_NO=5);
```

Query 10: Retrieve the name of each employee who has a dependent with the same first name and same sex as the employee.

```
SELECT E.F_name, E.name
FROM Employee E
WHERE E.AADHAR_NO in
(SELECT EAADHAR_NO
FROM Dependent
WHERE EAADHAR_NO=E.AADHAR_NOAND
E.FnameDep-name AND sex=E.sex);
        OR
SELECT E-Fname, E.Lname
FROM employee E, Dependent D
WHERE  E.AADHAR_NO=D.EAADHAR_NOAND
       E.sex=D.sex AND
       E.Fname=D.Dep-name;
```

Query 11: Find the sum of salaries of all employees, the maximum, minimum and average salary.

```
SELECT sum (salary), max (salary), min (salary) avg (salary)
FROM Employee
```

Query 12: Retrieve the names of all employees who have two or more dependents.

```
SELECT Lname, Fname
FROM Employee
WHERE (select count (*)
FROM Dependent
WHEREAADHAR_NO=(EAADHAR_NO)≥2;
```

Query 13: Retrieve the dependent number, the number of employees in the department and their average salary.

```
SELECTDNO, count (*), avg (salary)
FROM Employee
GROUPBY D_NO;
```


Query 14: Retrieve the project number, the project name and the number of employees who work on that project.

```
SELECT P_number, P_name, count (*)
FROM Project, works-on
WHERE P_number = PNO
GROUPBY P_number, P_name;
```

Query 15: For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

```
SELECT P_number, p_name, count (*)
SELECT Project, works-on
WHERE P_number = PNO
GROUP BY P_number, P_name
HAVING count (*) > 2;
```

Query 16: Retrieve the project number, name and the number of employees from department 5 who work on the project.

```
SELECT P_number, P_name, count (*)
FROM Project, works-on Employee
WHERE P_number = P_No ANDAADHAR_NO = EAADHAR_NO
AND DNO = 5
GROUP BY P_number, P_name;
```

Query 17: For each department having more than five employees, retrieve the department number and the number of employees making more than Rs. 50,000.

```
SELECT D_name, count (*)
FROM Department, Employee
WHERE D_number=DNo and salary > 50000 and
DNO in (SELECT DNO
FROM Employee
GROUPBY DNO
HAVING count (*) > 5)
GROUP BY D_name;
```

Query 18: Retrieve all employees whose address is in Pune.

```
SELECT F_name, L_name
FROM Employee
WHERE Addr like "%Pune%";
```

Query 19: Show the resulting salaries if every employee working on the "Product X" project is given a 10% raise.

```
SELECT F_name, L_name, 1.1 * salary
FROM Employee, works-on, Project
WHEREAADHAR_NO=EAADHAR_NO and P_no=P_number
AND P_name="Product X";
```

Query 20: Retrieve a list of employees and the projects they are working on, ordered by department in descending order and within each department, ascending of name.

```
SELECT D_name, L_name, F_name, P_name
FPOM Department, Employee, Works-on, Project
```

```
WHERE D_number=DNO ANDAADHAR_NO=EAADHAR_NOANDPNO=P_NUMBER
ORDER BY D_name desc, L_name Asc;
```

PRACTICE QUESTIONS

Q. I: Multiple Choice Questions:

1. SQL stands for _____.
   (a) Structured Query Language (b) Sequential Query Language
   (c) both (a) and b (d) None of these
2. Various SQL elements includes _____.
   (a) Queries (b) Statements and Expressions
   (c) Clauses (d) All of these
3. The statement which is used to retrieve the information is called as a _____.
   (a) query (b) expression
   (c) statement (d) None of these
4. Basic structure of an SQL expression consists of _____ clauses.
   (a) SELECT (b) FROM
   (c) WHERE (d) All of these
5. Which commands are mainly used for design and definition the structure of a database?
   (a) DML (b) DDL
   (c) DCL (d) DQL
6. Which command removes all rows from a table without logging the individual row deletions?
   (a) DELETE (b) DROP
   (c) TRUNCATE (d) REMOVE
7. Which of the following is not a DDL command?
   (a) UPDATE (b) TRUNCATE
   (c) ALTER (d) None of these
8. Which command defines its columns, integrity constraint in create table?
   (a) ALTER (b) DROP
   (c) CREATE (d) All of these
9. Which commands are used for manipulating data in database?
   (a) DML (b) DDL
   (c) DCL (d) DQL
10. Which clause in SQL is used to retrieve or fetch data from a database?
    (a) ORDER BY (b) HAVING
    (c) SELECT (d) All of these
11. Which clause is also used to find natural join of more than one table?
    (a) ORDER BY (b) FORM
    (c) SELECT (d) All of these
12. Which operators combine the results of two or more component queries into one result?
    (a) relational (b) logical
    (c) set (d) comparison
13. Which functions take a collection of values as inputs and return a single value
    (a) Aggregate (b) string
    (c) date (d) time
14. To connect or compare logical conditions we use _____ keywords.
    (a) AND (b) OR
    (c) NOT (d) All of these

15. A query within a query is called as _____.
    (a) nested query (b) database query
    (c) both (a) and b (d) None of these
16. Which are used to retrieve data from multiple tables?
    (a) sub-query (b) nested query
    (c) JOINS (d) None of these
17. Which type of join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal (join condition is met)?
    (a) LEFT OUTER JOIN (b) RIGHT OUTER JOIN
    (c) a FULL OUTER JOIN (d) All of these
18. What is a view?
    (a) A view is a special stored procedure executed when certain event occurs
    (b) A view is a virtual table which results of executing a pre-compiled query
    (c) A view is a database diagram
    (d) None of these
19. To create a view, we use which statement?
    (a) CREATE VIEW (b) ALTER VIEW
    (c) BEGIN VIEW (d) None of these

Answers

1. (a)	2. (d)	3. (a)	4. (d)	5. (b)	6. (c)	7. (a)	8. (c)	9. (a)	10. (c)
11. (b)	12. (c)	13. (a)	14. (d)	15. (a)	16. (c)	17. (b)	18. (b)	19. (a)	

Q.II: Fill in the Blanks:

1. SQL is a database computer language designed for the retrieval and management of data in a _____ database.
2. A _____ language for relational model is based on relational algebra.
3. _____ command is used to rename a table.
4. The _____ statement is used to add new rows (records) of data to a table in the database.
5. The _____ statement is used to delete rows (records) from a table.
6. The clause is used to sort the data in ascending or descending order, based on one or more columns.
7. Queries containing set operators are called _____ queries.
8. The _____ operator returns all rows that are selected by either query.
9. The _____ clause is used in collaboration with the SELECT statement to arrange identical data into groups.
10. _____ can be defined as "a query within a query".
11. _____ operation combines result of two Select statements and return only that result which belongs to first set of result.
12. The _____ selects all rows from both participating tables as long as there is a match between the columns.
13. _____ creates a virtual relation for storing the query.
14. The _____ operation eliminates duplicate values from result.
15. The commands which describe the structure of information in the database is called _____ command.
16. To avoid displaying duplicate values, use _____ keyword.
17. To specify range into any relational database, _____ operator is used.

18. The operator used for pattern matching in string operating is _____.
19. To sort the records in descending order, _____ keyword is used with order by clause.

Answers

1. Relational	2. Query	3. RENAME	4. INSERT	5. DELETE
6. ORDER BY	7. Compound	8. UNION	9. GROUP BY	10. Sub-query
11. Minus	12. INNER JOIN	13. View	14. intersection	15. DDL
16. distinct	17. between	18. like	19. DESC	

Q. III: State True or False:

1. SQL is the standard language for Relational Database System. T
2. Query processing consists of three-step parsing and translation, optimization and execution of the query submitted by the user. T
3. CREATE TABLE statement is used to create a new view. F
4. DDL commands are CREATE, ALTER, DROP, DESC. T
5. The UPDATE statement is used to modify the existing rows in a table. T
6. The GROUP BY clause is used to sort the data in ascending or descending order, based on one or more columns. F
7. The DISTINCT keyword is used to return only distinct (different) values. T
8. UNION ALL operation is used to combine two SELECT statements, but it only retunes the records which are common from both SELECT statements. F
9. The HAVING clause enables you to specify conditions that filter which group results appear in the final results. T
10. To check this impact on nested queries we use keyword exists or unique. T
11. A sub-query or inner query or nested query is a query within another SQL query and embedded within the WHERE clause. T
12. FULL OUTER JOIN is a type of join returns all rows from the LEFT-hand table and RIGHT-hand table with nulls in place where the join condition is not met. T
13. A view is named query that provides another way to present data in the database tables. T

1. (T) 2. (T) 3. (F) 4. (T) 5. (T) 6. (F) 7. (T) 8. (T) 9. (T) 10. (T)
11. (T) 12. (T) 13. (T)

Q. IV: Answer the following Questions:

(A) Short Answer Questions:

1. What is SQL?
2. Define query language.
3. What is query?
4. What is DDL?
5. What is DML?
6. Which clauses are used to form SQL structure.
7. What is view?
8. What is meant by NULL values?
9. What is JOIN?
10. Enlist set operations in SQL.
11. Which aggregate functions are used by SQL?
12. How to create and drop view?

13. Give syntax of ORDER BY and GROUP BY clause.
14. What is the use of FORM clause?

(B) Long Answer Questions:

1. What are the features of SQL?
2. With the help of diagram describe architecture of SQL.
3. Explain basic structure of SQL with query processing.
4. How to create and drop a database? Explain with example.
5. Describe various DDL commands with syntax and example.
6. How to create and rename a table? Explain woth example
7. Write short note on: DML commands.
8. Write short note on: Impact on SQL Constructs.
9. How to update a table? Explain with example.
10. Write short note: INSERT INTO command.
11. With the help of example describe DELETE command.
12. Describe where clause with example.
13. What is string? Give various string operations.
14. Describe set operations with example.
15. Write short note on: NULL values.
16. Enlist various string functions with example.
17. Describe SQL mechanisms for joining relations.
18. Describe the term nested subquery with example.
19. Write short note on: Views.
20. Consider following database.
    customer (cust_no, cust_name, addr)
    plant (plant_code, pl_name, pl_type, pl_cost)
    Nutrients (N_name, N_quantity, time)
21. Customer and plants are related with one-to-many and plant and nutrients are also related with one-to-many relationship.
22. Create RDB and answer following queries.
    (i) List the customer who purchase plant 'rose'.
    (ii) List the plants whose cost is more than Rs. 500/-.
    (iii) List all those plants to whom nutrient 'manure' is given.

    Consider the following database:
    person (SS-No, name, address)
    car (lic, year, model)
    Accident (date, driver, damage-amount)
    owns (SS-No, lic)
    log (lic date, driver)

    Write SQL for the following :
    (i) Find the total number of people whose cars were involved in accidents in 2000.
    (ii) Find the number of accidents in which the cars belonging to "Atul" were involved.
    (iii) Retrieve the name of persons whose address contain "Pune".
    (iv) Find the name of persons having more than 2 cars.
    (v) Find the year in which the car owe.

UNIVERSITY QUESTIONS AND ANSWERS

April 2015

1. State the different types of outer join operations. (1 M)

Ans. Refer to Section 3.11.2.

2. Consider the following relations:
   Supplier (S_id, sname, address)
   Parts (P_id, Pname, Colour)
   Suppliers and parts are related with many to many relationship with the descriptive attribute cost. Create a relational database in 3NF and solve the following queries in SQL:
   (i) Find the names of suppliers who supply parts which are blue or pink in colour.
   (ii) Find total cost of all parts supplied by 'Shree Agencies'.
   (iii) Find the names and address of all suppliers who are supplying the item 'Bath towel'. (5 M)

Ans. Refer to Section 3.13.

3. Consider the following relations:
   Musician (m_no , m_name, age, city)
   Instrument (i_no , i_name)
   Musician and instrument are related with a many to many relationship. Create a relational database in 3NF and solve the following queries in SQL:
   (i) list all the 'tabala' players.
   (ii) Find all the musicians who study in Pune and play 'flute'.
   (iii) List all the instruments that are played by more than 3 musicians. (5 M)

Ans. Refer to Section 3.13.

4. What is DML? Explain procedural and non-procedural DML. (3 M)

Ans. DML is a language that provides a set of operations to support the basic data manipulation operations on the data held in the databases. DML are basically two types:

(i) The **procedural** or **low-level DML** requires user to specify what data is required and how to access that data by providing step-by-step procedure. For example, Relational Algebra (RA) is procedural query language, which consists of set of operations such as select, project, union, etc., to manipulate the data in the database.

(ii) The **non-procedural** or **high-level** or **declarative DML** requires a user to specify what data is required without specifying how to retrieve the required data. For example, SQL (Structured Query Language) is a non-procedural query language as it enables user to easily define the structure or modify the data in the database without specifying the details of how to manipulate the database.

April 2016

1. Define DDL and DML. (1 M)

Ans. Refer to Sections 3.3 and 3.4.

2. Consider the following relations:
   Employee (e_no, ename, address, salary)
   Department (d_no, dname, assets)
   Employee and Department are related with many to one relationship. Create a RDB and solve the following queries in SQL:
   (i) List all the employees belonging to the 'Production' department.

   (ii) Give the names and salaries of all employees working in the departments having assets greater than 2,00,000.
   (iii) Find the names of departments where more than 30 employees are working. (5 M)

Ans. Refer to Section 3.13.

3. Consider the following relations:
   Student (s_no, name, class, age)
   IQtest(t_no, tname, t_type, t_equipment, level)
   Student and IQ test are related with many to many relationship, with the descriptive attribute score. Create a RDB and solve the following queries in SQL :
   (i) Give the name of the student who has scored maximum marks in 'Cognitive Assessment' test.
   (ii) List the distinct types of IQ test available.
   (iii) List studentwise test and score for all students of age 5. (5 M)

Ans. Refer to Section 3.13.

4. Consider the following relations :
   Project (p_no, pname, location, budge)
   Member (m_no, name, department, specialization)
   Project and member are related with many to many relationship. Create a RDB and solve the following queries in SQL :
   (i) List all the members working on 'ACM development' project.
   (ii) Count the no. of projects in 'Kothrud'.
   (iii) Give name of the project with minimum budge. (5 M)

Ans. Refer to Section 3.13.

5. Consider the following relations :
   Dancer (d_no, name, age, phone_no.)
   Dance_form (f_no, name, state-of-origin)
   Dancer and Dance_form are related with many to many relationship. Create a RDB and solve the following queries in SQL.
   (i) Find names of dancers who know the dance form 'Bharatnatyam'.
   (ii) Count the no. of dance forms which have originated in the state of Karnataka.
   (iii) Give the names of dancers who know more than one dance form. (5 M)

Ans. Refer to Section 3.13.

October 2016

1. Explain the use of order by clause in SQL. (1 M)

Ans. Refer to Page 3.17.

2. Write a note on aggregate functions used in SQL. (5 M)

Ans. Refer to Section 3.7.1.

3. Loan (Loan_no, amount, period)
   Customer (C_name, city)
   Borrower (C_name, Loan_no)
   (i) Find out loans for the period of 10 years.
   (ii) Find customers having loan of the amount 1000000.
   (iii) List out the customers staying in 'Nagar'.
   (iv) List the details of customers along with loan details.
   (v) Find out customers not having loan. (5 M)

Ans. Refer to Section 3.13.

April 2017

1. What is Right Outer Join?

Ans. Refer to Section 3.11.2.3. (1 M)

2. List any two aggregate functions in SQL.

Ans. Refer to Section 3.7.1. (1 M)

3. Consider the following relations :

Wholesaler (wno, wname, address, city)

Product (Pno, Pname)

Wholesaler and product are related with many to many relationship. Create a relational database in 3NF and solve the following queries in SQL:

(i) List the wholesalers of product 'Mouse'.

(ii) Count the number of wholesaler from 'Pune' city.

(iii) Delete records of wholesaler where product name is 'Scanner'. (5 M)

Ans. Refer to Section 3.13.

October 2017

1. Explain the following command with example:

(i) Update.

Ans. Refer to Page 3.8 Point (2).

(ii) Alter. (5 M)

Ans. Refer to Page 3.6 Point (3).

2. Consider the following relations :

Company (c_id, c_product, c_name, region, state)

Branches (b_id, b_name, b_product, city)

Company and Branches are related with one to many relationship. Create a relational database in 3NF and solve the following in SQL:

(i) List all the cities having branch product 'CPU' and 'MOUSE'.

(ii) List all the states whose branch product is 'Pen Drive'.

(iii) Print citywise branches in descending order.

Ans. Refer to Section 3.13.

April 2018

1. Modification in table is a part of DDL statement. Justify true or false. (1 M)

Ans. Refer to Section 3.3.

2. What is DDL? Write any two examples of DDL. (3 M)

Ans. Refer to Section 3.3.

3. Consider the following relations :

Player (pno, pname, city)

Game (gno, gname, city)

Player-Game (pno, gno, date)

(i) Find all players playing 'Football".

(ii) List all games details played on 38/3/3018.

(iii) List all games details played in Jaipur.

(iv) List all players playing both football and basketball.

(v) List all players who are playing in the same city where they live. (5 M)

Ans. Refer to Section 3.13.