MODULE –III –CHAPTER 1

Structured Query Language (SQL)

Syllabus

Structured Query Language (SQL): Introduction to SQL, Basic SQL Queries, DML,DDL,DCL,TCL commands, The set oriented commands like Union, Intersection, Except and Nested Queries, Aggregate operators, Null values, Relational set operators, SQL join operators.

Database Application Development: SQL functions, procedural SQL, embedded SQL, cursors, ODBC and JDBC, triggers and active database, designing active databases.

SQL Introduction

- Structure Query Language (SQL) is a database query language used for storing and managing data in Relational DBMS.
- SQL was the first commercial language introduced for E.F Codd's Relational model of database.
- Almost all RDBMS (MySql, Oracle, Infomix, Sybase, MS Access) use SQL as the standard database query language.
- SQL is used to perform all types of data operations in RDBMS. SQL is a standard language for accessing and manipulating databases.

Structured Query Language(SQL)

- Language for describing database schema and operations on tables
- DDL, DML, DCL and TCL are considered sublanguages of SQL

Data Definition Language

- DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema.
- A data definition language (DDL) is a computer language used to create and modify the structure of database objects in a database. These database objects include views, schemas, tables, indexes, etc.

Examples of DDL commands:

- **CREATE** – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- **DROP** – is used to delete objects from the database.
- **ALTER**-is used to alter the structure of the database.
- **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed.
- **RENAME** –is used to rename an object existing in the database.

CREATE TABLE

- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n))
- A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE   DEPARTMENT
      (        DNAME          VARCHAR(10)  NOT NULL,
               DNUMBER        INTEGER      NOT NULL,
               MGRSSN         CHAR(9),
               MGRSTARTDATE       CHAR(9)  );
```

Modifying a Table (Alter Command)

- To change the definition of the table.
- This can be done by using ALTER TABLE command. This command may have one of the following clauses.

- ADD| MODIFY| DROP

Alter Command

Alter command is used for altering the table structure, such as,

- To add a column to existing table
- To rename any existing column
- To change data type of any column or to modify its size.
- To drop a column from the table.

- The general syntax for the ALTER TABLE is as follows:
  ALTER TABLE <table name>[ADD|MODIFY| DROP]
  (Constraint | | column specification)

Alter Command (ADD)

- **ADD Clause:** The ADD clause is used to add a column and/or constraints to an existing table.

- To add a column say part_full_time to emp table, the syntax will be as follows:

  ALTER TABLE emp ADD (part_full_time CHAR(1));

- If this table is having some existing data corresponding to other columns, then it will take NULL for this column corresponding to those records.

Alter Command (ADD)

Adding Multiple Columns:

```
ALTER TABLE emp ADD (part_time CHAR (1),
  full_time varchar(2));
```

MODIFY Clause

- This clause is used to modify the column specifications and the constraints.

- In case of **constraints**, only possibilities are to modify a NULL to NOT NULL and NOT NULL to NULL. Other constraints should be first deleted and then recreated with the modification.

- Suppose we want to **increase** the width of a column, syntax is
  ALTER TABLE emp MODIFY (sal NUMBER(5));

- It will increase the width of sal column from NUMBER(4) to NUMBER(5).

However, for decreasing the width of a column or changing the data type of the column, the column must not contain any data.

MODIFY Clause

- For modification in the constraint, the syntax is:
  ALTER TABLE emp MODIFY(sal NUMBER(5)
  NOT NULL);

- Now, sal column will become NOT NULL column.

DROP Clause

- We can remove a column from table directly by using the DROP clause

  For example:

  ALTER TABLE emp DROP COLUMN part_full_time;

- This will drop the part_full_time column from the table along with the data.

DROP Clause

- To drop a constraint, the different syntaxes are
  ALTER TABLE emp DROP CONSTRAINT constraint_name;

// To drop Constraint.

- ALTER TABLE emp DROP PRIMARY KEY;

- DROP TABLE emp; // To drop table along with data permanently.

- ALTER TABLE emp DROP PRIMARY KEY;

//This command will drop all the dependencies of the Primary Key (i.e. the Foreign Key constraints in different table, which are based on this Primary Key) and then will drop this Primary key in a single step.

TRUNCATE Command

- The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table but structure remains.

- Syntax: TRUNCATE TABLE table_name;
- DROP TABLE command delete complete table and remove complete table structure from the database, while TRUNCATE TABLE do not effect table structure .

RENAME

RENAME TABLE:
RENAME TABLE tbl_name TO new_tbl_name;
Where tbl_name is table that exists in the current database, and new_tbl_name is new table name.

RENAME COLUMN: ALTER TABLE tablename RENAME COLUMN OldName TO NewNam

Data Manipulation Language (DML)

- DML is short name of Data Manipulation Language which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.
- SELECT - retrieve data from a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - Delete all records from a database table

INSERT Command

INSERT Command:- This command is used to insert rows into table.

The basic syntax of this command is as given

INSERT INTO <tablename> (Column1, Column2,.... Column n) VALUES (Value1, Value2...., Value n)

For example ,

INSERT INTO emp(empno, ename, job, sal, hiredate, deptno) VALUES (‘1’,’neha’, ‘student’, ‘34566’, 12-jul-2013,’cse’);

This statement will add a new record in the emp table.

INSERT Command

- When data is not to be entered into every column in the table, then either enter NULL corresponding to all those columns which do not require the value or specify only those columns which require a value.

Example: INSERT INTO emp(empno, ename, job, sal, hiredate, deptno) VALUES ('1','neha', 'NULL', '34566', 12-jul-2013,'NULL');

INSERT INTO emp(empno, ename, sal, hiredate) VALUES ('1','neha', '34566', 12-jul-2013);

INSERT Command

- One more style for INSERT command is without specifying column names as given in the following:

  INSERT INTO emp VALUES (‘1’, ’neha’, ‘student’, ‘34566’, 12-jul-2013,’cse’);
- In this case, values must be in the same sequence as specified at the table creation time.

Changing Table Contents

To change the value of a column or a group of columns in a table corresponding to some search criteria, UPDATE command is used.

- Update can be used for:
  - All the rows from a table
  - A select set of rows from a table.

Update Command

The general syntax of this command is as:

U**PDATE<tablename> SET <columnname1> = <newvalue1>, <columnname2> = <newvalue2> WHERE <search criteria>**

For example to update the salary of king to 6000 in emp table, the statement is:

```
UPDATE emp SET sal = 6000 WHERE
   ename= 'king';
```

- Example 2:

```
Update emp   set Net_sal = Net_sal +
  basic_sal*0.15;
```

Deleting records from the table

- Records from the table can be deleted individually or in groups by using DELETE Command. Delete can be used to:

- Delete all rows from a table.
- A select set of rows from a table.

The general syntax is :

DELETE FROM <tablename> WHERE <search condition>

For example:

DELETE FROM emp WHERE deptno=10;

In absence of WHERE Clause, this syntax will delete all the records from the table. **Ex: DELETE FROM emp;**

Delete Command

The subqueries can also be used in DELETE Command for example, if we have to delete all the records from Accounts Department then the syntax will be as follows:

DELETE FROM emp where deptno IN (SELECT deptno from dept WHERE dname= 'Accounts');

SQL Select

- To view the global table data.
- To view filtered table data
- Selected column and all rows
- Selected rows and all columns
- Selected columns and selected rows.

To view the global table data

Syntax:

SELECT [DISTINCT] select-list
FROM from-list
WHERE qualification

- The **from-list** in the `FROM` clause is a list of table names. A table name can be followed by a **range variable**; a range variable is particularly useful when the same table name appears more than once in the from-list.

- The **select-list** is a list of (expressions involving) column names of tables named in the from-list. Column names can be prefixed by a range variable.

- The **qualification** in the `WHERE` clause is a boolean combination (i.e., an expression using the logical connectives `AND`, `OR`, and `NOT`) of conditions of the form *expression* `op` *expression*, where `op` is one of the comparison operators $\{<, <=, =, <>, >=, >\}$.[2] An *expression* is a *column* name, a *constant*, or an (arithmetic or string) expression.

- The `DISTINCT` keyword is optional. It indicates that the table computed as an answer to this query should not contain *duplicates*, that is, two copies of the same row. The default is that duplicates are not eliminated.

Description

1. Compute the cross-product of the tables in the from-list.
2. Delete rows in the cross-product that fail the qualification conditions.
3. Delete all columns that do not appear in the select-list.
4. If DISTINCT is specified, eliminate duplicate rows.

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

Basic Queries

- Find all sailors with a rating above 7.

```
SELECT S.sid, S.sname, S.rating, S.age
FROM   Sailors AS S
WHERE  S.rating > 7
```

Find the names of sailors who have reserved boat number 103

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid = R.sid AND  R.bid=103
```

Find the sids of sailors who have reserved a red boat.

```
SELECT  R.sid
FROM    Boats B, Reserves R
WHERE   B.bid = R.bid AND B.color = 'red'
```

Find the colorS of boats reserved by Lubber.

```
SELECT B3.color
FROM   Sailors S, Reserves R, Boats B3
WHERE  S.sid = R.sid AND R.bid = B.bid AND S.sname = 'Lubber'
```

Find the names of sailors who have Reserved at least one boat.

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid = R.sid
```

LIKE Operator:

- SQL provides support for pattern matching through the LIKE operator, along with the use of the wild-card symbols % (which stands for zero or more arbitrary characters) and _(which stands for exactly one, arbitrary, character).
- E.g. ‘ _AB%’ denotes a pattern that will match every string that contains at least three characters, with the second and third characters being A and B respectively.

LIKE (Examples)

- SELECT * FROM emp WHERE ename LIKE ‘A%’;

  It gives the details of those employees whose name starts from character A. Example: Anu

- SELECT * FROM emp WHERE ename LIKE ‘%a’;

  It returns the rows in which the value in ename column ends with character ‘a’. Example: deepika

- SELECT * FROM emp WHERE ename LIKE ‘%r%’;

  It will display the information about those employees who include ‘r’ in their names.

  Example: priya

Find the ages of sailors whose name begins and ends with B and has atleast three characters.

- SELECT S.age FROM WHERE Sailors S S.sname LIKE 'B_%B’

LIKE allows for a comparison of one string value with another string value

UNION, INTERSECT, AND EXCEPT

SQL supports these operations under the names UNION, INTERSECT, and EXCEPT.

Find the names of sailors who have reserved a red or a green boat.

```
SELECT  S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid
        AND (B.color = 'red' OR B.color = 'green')
```

```
SELECT  S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND  R.bid = B.bid AND  B.color = 'red'
UNION
SELECT  S2.sname
FROM    Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid = R2.sid AND  R2.bid = B2.bid AND  B2.color = 'green'
```

Find the names of sailors who have reserved both a red and a green boat.

```
SELECT S.sname
FROM   Sailors S, Reserves R, Boats B
WHERE  S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
INTERSECT
SELECT S2.sname
FROM   Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

Find the sids of all sailors who have reserved red boats but not green boats.

```
SELECT S.sid
FROM   Sailors S, Reserves R, Boats B
WHERE  S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
EXCEPT
SELECT S2.sid
FROM   Sailors S2, Reserves R2, Boats B2
WHERE  S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

```
SELECT  R.sid
FROM    Boats B, Reserves R
WHERE   R.bid = B.bid AND  B.color = 'red'
EXCEPT
SELECT  R2.sid
FROM    Boats B2, Reserves R2
WHERE   R2.bid = B2.bid AND  B2.color = 'green'
```

Find all sids of sailors who have a rating of 10 or have reserved boat 104.

```
SELECT  S.sid
FROM    Sailors S
WHERE   S.rating = 10
```

```
UNION
SELECT  R.sid
FROM    Reserves R
WHERE   R.bid = 104
```

NESTED QUERIES

- A nested query is a query that has another query embedded within it; the embedded query is called a subquery.
- A subquery typically appears within the WHERE clause of a query. Subqueries can sometimes appear in the FROM clause or the HAVING clause.
- **IN:** To compare a list of values against a column, we have to use IN operator.

Find the names of sailors who have reserved boat 103.

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN ( SELECT R.sid
                  FROM   Reserves R
                  WHERE  R.bid = 103 )
```

Find the names of sailors who have reserved a red boat.

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid IN ( SELECT R.sid
                   FROM   Reserves R
                   WHERE  R.bid IN ( SELECT B.bid
                                     FROM   Boats B
                                     WHERE  B.color = 'red' )
```

Find the names of sailors who have not reserved a red boat. To find the names of sailors who have not reserved a red boat, we replace the outermost occurrence of IN by NOT IN:

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid NOT IN ( SELECT R.sid
                       FROM   Reserves R
                       WHERE  R.bid IN ( SELECT B.bid
                                         FROM   Boats B
                                         WHERE  B.color = 'red' )
```

Correlated Nested Queries

- In the nested queries that we have seen thus far, the inner subquery has been completely independent of the outer query.
- In general the inner subquery could depend on the row that is currently being examined in the outer query.
- The EXISTS operator is another set comparison operator, such as IN. It allows us to test whether a set is nonempty

(Q1) Find the names of sailors who have reserved boat number 103.

```
SELECT S.sname
FROM   Sailors S
WHERE  EXISTS ( SELECT *
                FROM   Reserves R
                WHERE  R.bid = 103
                       AND R.sid = S.sid )
```

- For each Sailor row S, we test whether the set of Reserves rows R such that *R.bid = 103* AND *S.sid = R.sid* is nonempty.
- If so, sailor S has reserved boat 103, and we retrieve the name. The subquery clearly depends on the current row S and must be re-evaluated for each row in Sailors.
- The occurrence of S in the subquery (in the form of the literal *S.sid*) is called a *correlation*, and such queries are called *correlated queries*.
- By using NOT EXISTS instead of EXISTS, we can compute the names of sailors who have not reserved a red boat.

- SQL also supports op ANY and op ALL, where op is one of the arithmetic comparison operators (<;<=;=; <>;>=;>).

Find sailors whose rating is better than some sailor called Horatio.

```
SELECT  S.sid
FROM    Sailors S
WHERE   S.rating > ANY ( SELECT  S2.rating
                         FROM    Sailors S2
                         WHERE   S2.sname = 'Horatio' )
```

- On instance $S3$, this computes the *sid*s 31, 32, 58, 71, and 74.
- Just replace ANY with ALL in the WHERE clause of the outer query. On instance $S3$, we would get the *sid*s 58 and 71.
- What will be the output if Horatio is not present in the table?

Find the sailors with the highest rating.

```
SELECT  S.sid
FROM    Sailors S
WHERE   S.rating >= ALL ( SELECT  S2.rating
                          FROM    Sailors S2 )
```

Relational Operators

=	**Equal to**
< or >	Less than or greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

Expressions and Strings in the SELECT Command.

Compute increments for the ratings of persons who have sailed two different boats on the same day

```
SELECT S.sname, S.rating+1 AS rating
FROM   Sailors S, Reserves R1, Reserves R2
WHERE  S.sid = R1.sid AND S.sid = R2.sid
       AND R1.day = R2.day AND R1.bid <> R2.bid
```

Arithmetic Operators

- Create expressions with number and date data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Using Arithmetic Operators

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

	LAST_NAME	SALARY	SALARY+300
1	Whalen	4400	4700
2	Hartstein	13000	13300
3	Fay	6000	6300
4	Higgins	12000	12300
5	Gietz	8300	8600
6	King	24000	24300
7	Kochhar	17000	17300
8	De Haan	17000	17300
9	Hunold	9000	9300
10	Ernst	6000	6300

...

Operator Precedence

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

1

	LAST_NAME	SALARY	12*SALARY+100
1	Whalen	4400	52900
2	Hartstein	13000	156100
3	Fay	6000	72100

...

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

2

	LAST_NAME	SALARY	12*(SALARY+100)
1	Whalen	4400	54000
2	Hartstein	13000	157200
3	Fay	6000	73200

...

Defining a Column Alias

- A column alias:
  - Renames a column heading
  - Is useful with calculations
  - Immediately follows the column name (There can also be the optional `AS` keyword between the column name and alias.)
  - Requires double quotation marks if it contains spaces or special characters, or if it is case-sensitive

Using Column Aliases

```
SELECT last_name AS name,
commission_pct comm
FROM   employees;
```

	NAME	COMM
1	Whalen	(null)
2	Hartstein	(null)
3	Fay	(null)

...

```
SELECT last_name "Name" , salary*12 "Annual
Salary"
FROM   employees;
```

	Name	Annual Salary
1	Whalen	52800
2	Hartstein	156000
3	Fay	72000

...

AGGREGATE OPERATORS:

Aggregation operators calculate the average, total, minimum, or maximum value of the numeric attributes in a collection of objects, or the number of objects in a collection. Aggregation operators compute a value from a collection of values.

1. COUNT ([DISTINCT] A): The number of (unique) values in the A column.
2. SUM ([DISTINCT] A): The sum of all (unique) values in the A column.
3. AVG ([DISTINCT] A): The average of all (unique) values in the A column.
4. MAX (A): The maximum value in the A column.
5. MIN (A): The minimum value in the A column.

Find the average age of all sailors.

```
SELECT AVG (S.age)
FROM   Sailors S
```

Find the average age of sailors with a rating of 10.

```
SELECT AVG (S.age)
FROM   Sailors S
WHERE  S.rating = 10
```

Find the name and age of the oldest sailor.

```
SELECT S.sname, MAX (S.age)
FROM   Sailors S
```

Count the number of different sailor names.

```
SELECT COUNT ( DISTINCT S.sname )
FROM   Sailors S
```

Count the number of sailors.

```
SELECT COUNT (*)
FROM   Sailors S
```

Find the names of sailors who are older than the oldest sailor with a rating of 10.

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.age > ( SELECT MAX ( S2.age )
                  FROM    Sailors S2
                  WHERE   S2.rating = 10 )
```

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.age > ALL ( SELECT  S2.age
                      FROM    Sailors S2
                      WHERE   S2.rating = 10 )
```

SQL ORDER BY

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the query results in an ascending order by default.

Syntax

The basic syntax of the ORDER BY clause is as follows −

```
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 the column-list.

SQL ORDER BY

The following code block has an example, which would sort the result in the descending order by NAME.

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

This would produce the following result –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
+----+----------+-----+-----------+----------+
```

SQL ORDER BY

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 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code block has an example, which would sort the result in an ascending order by the NAME and the SALARY −

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

SQL ORDER BY

This would produce the following result –

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
7	Muffy	24	Indore	10000.00
1	Ramesh	32	Ahmedabad	2000.00

(Q31) Find the age of the youngest sailor for each rating level.

If we know that ratings are integers in the range 1 to 10, we could write 10 queries of the form:

```
SELECT MIN (S.age)
FROM   Sailors S
WHERE  S.rating = i
```

where $i = 1, 2, \ldots, 10$. Writing 10 such queries is tedious. More importantly, we may not know what rating levels exist in advance.

To write such queries, we need a major extension to the basic SQL query form, namely, the `GROUP BY` clause. In fact, the extension also includes an optional `HAVING` clause

```
SELECT    [ DISTINCT ] select-list
FROM      from-list
WHERE     qualification
GROUP BY  grouping-list
HAVING    group-qualification
```

Using the `GROUP BY` clause, we can write Q31 as follows:

```
SELECT    S.rating, MIN (S.age)
FROM      Sailors S
GROUP BY  S.rating
```

(Q32) Find the age of the youngest sailor who is eligible to vote (i.e., is at least 18 years old) for each rating level with at least two such sailors.

```
SELECT   S.rating, MIN (S.age) AS minage
FROM     Sailors S
WHERE    S.age >= 18
GROUP BY S.rating
HAVING   COUNT (*) > 1
```

Evaluation Steps:

- The first step is to construct the cross-product of tables in the **from-list**. Because the only relation in the from-list in Query is Sailors result is just the instance shown in Figure

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.10 Instance $S3$ of Sailors

The second step is to apply the qualification in the `WHERE` clause, $S.age >= 18$. This step eliminates the row $\langle 71, zorba, 10, 16 \rangle$. The third step is to eliminate unwanted columns. Only columns mentioned in the `SELECT` clause, the `GROUP BY` clause, or the `HAVING` clause are necessary, which means we can eliminate *sid* and *sname* in our example. The result is shown in Figure 5.11. The fourth step is to sort the table according to the `GROUP BY` clause to identify the groups. The result of this step is shown in Figure 5.12.

rating	*age*
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
9	35.0
3	25.5
3	63.5

Figure 5.11 After Evaluation Step 3

rating	*age*
1	33.0
3	25.5
3	63.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

Figure 5.12 After Evaluation Step 4

The fifth step is to apply the group-qualification in the HAVING clause, that is, the condition COUNT (*) > 1. This step eliminates the groups with *rating* equal to 1, 9, and 10. Observe that the order in which the WHERE and GROUP BY clauses are considered is significant: If the WHERE clause were not considered first, the group with *rating=10* would have met the group-qualification in the HAVING clause. The sixth step is to generate one answer row for each remaining group. The answer row corresponding to a group consists of a subset of the grouping columns, plus one or more columns generated by applying an aggregation operator. In our example, each answer row has a *rating* column and a *minage* column, which is computed by applying MIN to the values in the *age* column of the corresponding group. The result of this step is shown in Figure 5.13.

rating	*minage*
3	25.5
7	35.0
8	25.5

Figure 5.13 Final Result in Sample Evaluation

If the query contains DISTINCT in the SELECT clause, duplicates are eliminated in an additional, and final, step.

Find the names of sailors who are older than the oldest sailor with a rating of 10.

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.age > ( SELECT MAX ( S2.age )
                  FROM    Sailors S2
                  WHERE   S2.rating = 10 )
```

- For each red boat, find the number of reservations for this boat.

```
SELECT B.bid, COUNT (*) AS  NOR

 FROM Boats B, Reserves R

WHERE R.bid = B.bid AND B.color = ‘red’

GROUP BY B.bid;
```

SELECT queries

SQL SELECT statement syntax- It is the most frequently used SQL command and has the following general syntax

SELECT [DISTINCT| ALL] { * | [fieldExpression [AS newName]}
FROM tableName [alias]
[WHERE condition]
[GROUP BY fieldName(s)]
[HAVING condition] ORDER BY fieldName(s)

Here

- **SELECT** is the SQL keyword that lets the database know that we want to retrieve data.

- **[DISTINCT | ALL]** are optional keywords that can be used to fine tune the results returned from the SQL SELECT statement. If nothing is specified then ALL is assumed as the default.
- **{*| [fieldExpression [AS newName]}** at least one part must be specified, "*" selected all the fields from the specified table name, field Expression performs some computations on the specified fields such as adding numbers or putting together two string fields into one.
- **FROM** tableName is mandatory and must contain at least one table, multiple tables must be separated using commas or joined using the JOIN keyword.

- **WHERE** condition is optional, it can be used to specify criteria in the result set returned from the query.
- **GROUP BY** is used to put together records that have the same field values.
- **HAVING** condition is used to specify criteria when working using the GROUP BY keyword.
- **ORDER BY** is used to specify the sort order of the result set.

Null Values

- We use null when the column value is either unknown or inapplicable
- If we compare two null values using , =, and so on, the result is always unknown. For example, if we have null in two distinct rows of the sailor relation, any comparison returns unknown.
- SQL also provides a special comparison operator IS NULL to test whether a column value is null and also IS NOT NULL.

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	
7	Muffy	24	Indore	

SELECT *
FROM CUSTOMERS
WHERE SALARY IS NOT NULL;

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

SELECT * FROM CUSTOMERS
WHERE SALARY IS NULL;

This would produce the following result

ID	NAME	AGE	ADDRESS	SALARY
6	Komal	22	MP	
7	Muffy	24	Indore	

Logical operators AND, OR, and NOT with NULL

- Once we have null values, we must define the logical operators AND, OR, and NOT using a three-valued logic in which expressions evaluate to true, false, or unknown.
- The expression NOT unknown is defined to be unknown.
- OR of two arguments evaluates to true if either argument evaluates to true, and to unknown if one argument evaluates to false and the other evaluates to unknown.
- AND of two arguments evaluates to false if either argument evaluates to false, and to unknown if one argument evaluates to unknown and the other evaluates to true or unknown.

NULL in SQL

- In SQL,the qualification in the WHERE clause eliminates rows (in the cross-product of tables named in the FROM clause) for which the qualification does not evaluate to true. Therefore, in the presence of null values, any row that evaluates to false or to unknown is eliminated.
- We can disallow null values by specifying NOT NULL as part of the field definition, for example, sname CHAR(20) NOT NULL.
- In addition, the fields in a primary key are not allowed to take on null values. Thus, there is an implicit NOT NULL constraint for every field listed in a PRIMARY KEY constraint.

Joins

- An SQL **join clause combines columns from one or more tables in a relational database.**

- There are 4 different types of SQL joins:
- SQL INNER JOIN (simple join)
- SQL LEFT OUTER JOIN (LEFT JOIN)
- SQL RIGHT OUTER JOIN (RIGHT JOIN)
- SQL FULL OUTER JOIN (FULL JOIN)

Inner join

- It is the most common type of join. SQL Server INNER JOINS return all rows from multiple tables where the join condition is met.

- **Syntax:**

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

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Select * from S1 INNER JOIN S2 ON S1.sid = S2.sid ;

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;
```

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Select * from S1 LEFT OUTER JOIN S2 ON S1.sid = S2.sid ;

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;
```

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Select * from S1 RIGHT OUTER JOIN S2 ON S1.sid = S2.sid ;

Full outer join

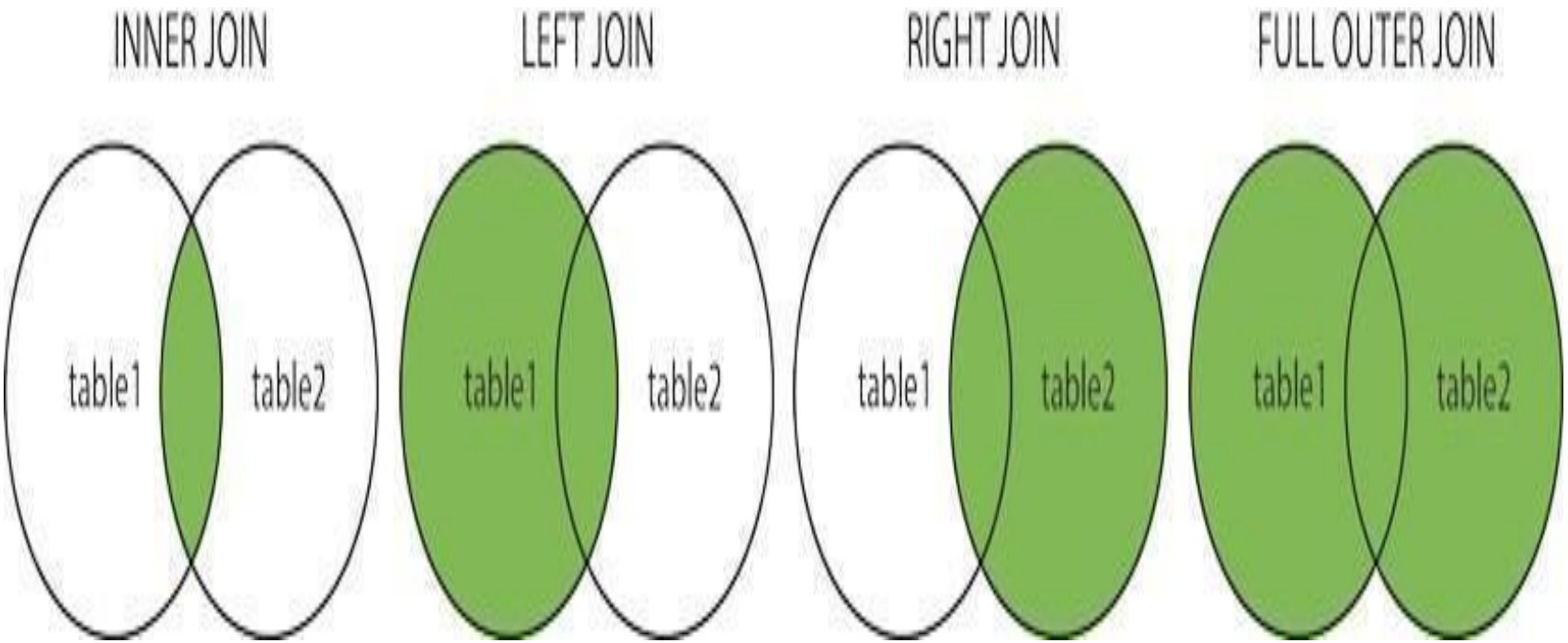
- This type of join returns all rows from the LEFT-hand table and RIGHT-hand table with nulls in place where the join condition is not met.

- **Syntax:** SELECT columns
  FROM table1
  FULL [OUTER] JOIN table2
  ON table1.column = table2.column;

Different Types of SQL JOINs

The different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table


INNER JOIN
table1
table2
LEFT JOIN
table1
table2
RIGHT JOIN
table1
table2
FULL OUTER JOIN
table1
table2

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

sid	sname	rating	age
28	yuppy	9	35
31	lubber	8	56
44	guppy	5	35
58	rusty	10	35

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;

Select *from sailors1 s1 left outer join sailors2 s2 on s1.sid=s2.sid;

sid	sname	rating	age	sid	sname	rating	age
22	dustin	7	45	NULL	NULL	NULL	NULL
31	lubber	8	55	31	lubber	8	56
58	rusty	10	35	58	rusty	10	35

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;
```

Select *from sailors1 s1 right outer join sailors2 s2 on s1.sid=s2.sid;

sid	sname	rating	age	sid	sname	rating	age
NULL	NULL	NULL	NULL	28	yuppy	9	35
31	lubber	8	55	31	lubber	8	56
NULL	NULL	NULL	NULL	44	guppy	5	35
58	rusty	10	35	58	rusty	10	35

Inner join

- It is the most common type of join. SQL Server INNER JOINS return all rows from multiple tables where the join condition is met.

- **Syntax:** SELECT columns
  FROM table1 INNER JOIN table2
  ON table1.column = table2.column;

Select *from sailors1 s1 inner join sailors2 s2 on s1.sid=s2.sid;

sid	sname	rating	age	sid	sname	rating	age
31	lubber	8	55	31	lubber	8	56
58	rusty	10	35	58	rusty	10	35

Data Control Language (DCL)

A Data Control Language (DCL) can be defined as a computer language that is used for controlling privilege in the database. The privileges are required for performing all the database operations, such as creating sequences, views or tables.

Types of Privileges

There are two main types of privileges in the database:

System Privileges- System privileges are used for performing a particular type of action on objects, such as cache groups, synonyms, materialized views, tables, views, sequences, indexes, replication schemes, and PL/SQL procedures & functions.

Object Privileges- Object privileges can be defined as the right for performing a special type of action on objects like materialized views, sequences, replication schemes, cache groups, synonyms, tables, views, etc. This type of privilege can be revoked and the owner of the object can grant object privileges.

Data Control Languages (DCL) Commands

There are two types of commands in the data control languages:

1. **Grant Command-** Grant Command is used for offering access or privileges to the users on the objects of the database. Through this command, the users get access to the privileges in the database.

- The General Syntax for the Grant Command is mentioned below:

  GRANT privilege_name ON

  object_name TO {user_name } ;

- **For Example-** GRANT ALL ON workers TO MNO;

  In the given example, the permission to view and modify the details in the 'workers table' has been given to the user MNO.

Data Control Languages (DCL) Commands

Revoke Command- The main purpose of the revoke command is canceling the previously granted permissions. Through the revoke command, the access to the given privileges can be withdrawn. In simple words, the permission can be taken back from the user with this command.

The general syntax for the revoke command is mentioned below:

REVOKE<privilege list> ON <relation name or view name> From <user name>

For Example- REVOKE UPDATE ON worker FROM MNO;

- **Allow a User to create session-** When we create a user in SQL, it is not even allowed to login and create a session until and unless proper permissions/privileges are granted to the user.
- Following command can be used to grant the session creating privileges.

GRANT CREATE SESSION TO username;

- **Allow a User to create table-** To allow a user to create tables in the database, we can use the below command,

GRANT CREATE TABLE TO username;

- **Grant permission to create any table**- Sometimes user is restricted from creating come tables with names which are reserved for system tables. But we can grant privileges to a user to create any table using the below command,
  GRANT CREATE ANY TABLE TO username;
- **Grant permission to drop any table-** As the title suggests, if we want to allow user to drop any table from the database, then grant this privilege to the user,
  GRANT DROP ANY TABLE TO username;
- **To take back Permissions-** If we want to take back the privileges from any user, use the REVOKE command.
  REVOKE CREATE TABLE FROM username;

Differences between the Grant and Revoke Command

Grant Command	**Revoke Command**
A user is allowed to perform some particular activities on the database by using Grant Command.	A user is disallowed to performing some particular activities by using the revoke command.
The access to privileges for database objects is granted to the other users.	The access to privileges for database objects that is granted previously to the users can be revoked.

Transaction Control Language(TCL)

Commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions.

COMMIT command

- ✓ COMMIT command is used to permanently save any transaction into the database.
- ✓ When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.
- ✓ To avoid that, we use the COMMIT command to mark the changes as permanent.
- ✓ Following is commit command's syntax- **COMMIT;**

ROLLBACK

- ✓ This command restores the database to last committed state. It is also used with SAVEPOINT command to jump to a save point in an ongoing transaction.
- ✓ If we have used the UPDATE command to make some changes into the database, and realize that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not committed using the COMMIT command.

syntax- **ROLLBACK TO savepoint_name;**

SAVEPOINT

SAVEPOINT command is used to temporarily save a transaction so that we can rollback to that point whenever required.

- ✓ Following is savepoint command's syntax- **SAVEPOINT savepoint_name;**
- ✓ In short, using this command we can **name** the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.
- ✓ Using Savepoint and Rollback and Following is the table **class**,

id	name
1	Abhi
2	Adam
4	Alex

Using some SQL queries on the above table and results.

```
INSERT INTO class VALUES(5, 'Rahul');

COMMIT;

UPDATE class SET name = 'Abhijit' WHERE id =

'5'; SAVEPOINT A;

INSERT INTO class VALUES(6, 'Chris');

SAVEPOINT B;

INSERT INTO class VALUES(7, 'Bravo');

SAVEPOINT C;

SELECT * FROM class;
```

id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris
7	Bravo

SAVEPOINT

Use the ROLLBACK command to roll back the state of data to the **savepoint**

B. ROLLBACK TO B;

SELECT * FROM class;

Now **class** table will look like,

id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris

Schema of Employee:

Employee(eid,fname,lname,sal,dept-no)

1. Find the employee details whose name is smith?
2. Add phone-no column to emp table.
3. Display all details of employee without duplicate data.
4. Find name, sal of employee who working in dept-no 10.
5. Add 5000 plus salary to already existing salary for a employee called John.
6. Remove employee details with dept-no 30.

GROUP BY Syntax :

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
ORDER BY *column_name(s);*

Examples Queries :

1. lists the number of employees in each department.

1. lists the number of employees in each department, sorted high to low

1. the total salary of each employee.

Having clause

- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

HAVING Syntax:

SELECT *column_name(s)*
FROM *table_name*
GROUP BY *column_name(s)*
HAVING *condition*
ORDER BY *column_name(s);*

Example queries:

1. lists the number of employee in each department. Only include department with more than 3 employees.

1. lists the number of employees in each department, sorted high to low. And include department with more than 3 employees.