12- SQL – STRUCTURED QUERY LANGUAGE

Processing capabilities of SQL, DDL, DML

SQL – is a simple query language used for accessing, handling and managing data in relational databases.

Or

SQL – is a language that enables the user to create and operate on relational databases, which are sets of related information stored in tables.

SQL – STRUCTURED QUERY LANGUAGE

SQL- is a non-procedural computer language, aimed to store, manipulate and query data stored in a relational database and also used to create interface between user and database.

Advantages :

High speed, Easy to learn and understand, No Coding, Portability

SQL – STRUCTURED QUERY LANGUAGE

Rules for SQL :

1.Keywords cannot be abbreviated, not case sensitive, comma (,) is used to separate parameters, character and date constants should be placed within single quotes, we can work with multiple table using single command.

PROCESSING CAPABILITIES OF SQL

DDL – Data Definition Language
IDML – Interactive Data Manipulation Language
EDML – Embedded Data Manipulation Language
VD – View Definition
Authorization
Integrity
Transaction Control

DDL - DATA DEFINITION LANGUAGE

DDL – provides a set of definitions to specify the storage structure and access methods used by the database system.

Whenever data is read or modified in the database system, the data dictionary is consulted

Data Dictionary is a file that contains “metadata”

FUNCTIONS OF DDL

- Identify the data division
- Naming
- Data types
- Length
- Range
- Validation/error identification
- Logical data definition

DATA TYPES

NUMBER(n,d)	n is precision (max of 38 digits) and d is the scale value
CHAR(n)	for character string of n characters enclosed within single quotes(‘ ‘)
DECIMAL(n,s)	n is precision (max of 38 digits) and s is the scale value
DATE	Any valid date
TIME	Any valid time

DML – DATA MANIPULATION LANGUAGE

A Data Manipulation Language is a language that enables the users to access or manipulate data as organized by the appropriate data model.

Or

DML we mean

- retrieval
- Insertion
- Deletion
- Modification of data stored in database

TYPES OF DML

Specify – what data is needed and how to get it.
(Procedural DML)
Specify – what data is needed and without specifying how to get it.
(Non-procedural DML)

SQL COMMANDS AND FUNCTIONS

Keywords – words with special meaning
Commands or Statements - instructions
Clauses – begin with a keyword and consists of
keywords and arguments
Symbols used –
| - or
() – to treat within as a unit
[] – optional
<> SQL and other special terms in angle brackets

DDL COMMANDS

Create table –
Create table <table-name>(<column name> <data type> [<size>], >(<column name> <data type> [<size>],….);
Create table employee
(ecode integer, ename char(20),gender char(1),grade char(2),gross decimal);

1) Create a database
2) Create the following table in the database (Student)

Stdno(int)	Stdname(Varchar-25)	Mark1(Decimal)	DOB(Date)	Gender(Char)
1001	Gayathri	89	2000/12/13	F
1002	Harini	77	2001/6/28	F
1003	Jayesh	90	2001/4/4	M

1) Display all records whose gender is F
2) Display all records whose mark is above 85
3) Drop the table
4) Exit of SQL

DDL - CADV - Create table, Alter table,Drop table,View table - creating structure
DML - SIDU - Select,Insert,Delete,Update

DDL COMMANDS

```
ALTER TABLE <TABLE NAME> ADD <COLUMN NAME>
<DATA TYPE> <SIZE>;

Alter table employee modify (ename char(40));
```

DDL COMMANDS

Constraints- A constraint is a condition or check applicable on a field or set of fields.

NULL

NOT NULL

UNIQUE

PRIMARY KEY

DEFAULT

CHECK

DDL COMMANDS

Constraints

TABLE CONSTRAINT – Is to be applied on a group of columns of the table

Eg :

```
Create table employee
(
ecode integer not null constraint
descp char(30),
Marks integer,
Unique(ecode, descp));
```

Column Constraint – is applicable only to a column.

DML COMMANDS

INSERT INTO <tablename>[<column list>] values (<value><value>.....);

Insert into employee
values(1001,'xkxk','M','C1',56844.99);
Insert into employee(ecode,ename,sex,grade,gross)
values(1002,'skdlfs','F','A1',68009.99);

DML COMMANDS

SELECT <column name>[,<column-name>….] from <table-name> <where condition>;

Select ecode,ename from employee;

DML COMMANDS

```
DELETE FROM <tablename> [where <predicate>];

Delete from employee where gender='M';
```

DML COMMANDS

```
Update <table name> set <column
name=value/expr>;

Update employee set grade='A1' where
ecode=1001;
```

DML COMMANDS

```
DELETE FROM EMPLOYEE;

DDL COMMAND

DROP TABLE EMPLOYEE;
```

RELATIONAL OPERATORS USED

CLAUSES USED

Where, Order by, Group by, Having

Select from employee where ename like “A%”;

Select from employee group by city having city in (‘Delhi’,’Chennai’);

Select from employee where ename like ‘A_ _y” order by asc;

Select from employee where gross between 45000 and 50000 order by desc;

MySQL - Database System - MySQL server instance + MySQL database.

MySQL - Client/Server architecture - server runs on the machine containing the databases and clients connect to the server over a network.

Key features of MySQL :(refer XI textbook - Chapter 15 - Relational Databases)

- Speed
- Ease of Use
- Cost
- Query Language Support
- Portability
- Data Types
- Security
- Scalability and limits
- Connectivity
- Localization
- Clients and Tools

LOGICAL OPERATORS USED

Not

Select from employee where (not grade='A1');

Or

Select from employee where (grade='A1' or grade='B1');

And

Select from employee where (grade='A1' and gross>10000);

CONDITION BASED ON A RANGE, LIST, PATTERN

Between

Select from employee where gross between 10000 and 50000;

Select from employee where city in (‘Delhi’,’Agra’);

And

Select from employee where ename like=‘B%’;

KEYWORDS USED

All, distinct

Select all city from employee;

Note : Output will have duplicates

Select distinct city from employee;

Note : Output will not have duplicates

SQL FUNCTIONS

Avg – to compute average values

Select avg(gross) from employee where city in('Delhi','Chennai');

sum – to find total

Select sum(gross) from employee where city in('Delhi','Chennai');

Min – to find the minimum value

Max – to find the maximum value

Count – to count not-null values in a column

Select count(All city) from employee;

Count() – to count total number of rows in a table

Select ecode,ename,count(*) from employee group by city;

TABLE : STORE

DDL COMMANDS

TABLE : STORE

ITEMNO	ITEM	SCODE	QTY	RATE	LASTBUY
2001	ERASER SMALL	22	220	6	2009-01-19
2002	GEL PEN PREMIUM	21	250	20	2009-03-11
2005	SHARPNER CLASSIC	23	60	8	2009-06-30
2009	BALL PEN 0.5	21	180	18	2009-11-03
2004	ERASER BIG	22	110	8	2009-12-02
2003	BALL PEN 0.25	22	50	25	2010-02-01

DDL COMMANDS

TABLE :
STORE

1. To display details of all Items in the STORE table in ascending order of LastBuy.
2. To display ItemNo and ItemName of those items from STORE table whose rate is more than Rs. 15
3. To display the details of those Items whose supplier code(SCode) is 22 or Quantity in store(Qty) is more than 110 from the table STORE
4. Display minimum rate of items for each supplier individually as per Scode from the table STORE.
5. Add one more column Supplier_Name of 25 characters

TABLE : STORE

DDL COMMANDS

1. select * from store order by LastBuy;
2. select itemno, item from store where rate>15;
3. select * from store where Scode=22 or Qty>110;
4. select Min(Rate), Scode from store group by Scode;
5. alter table store add Supplier_Name char(25);

Defining Primary Key through Alter Table command

Alter Table Table_name ADD primary key(Field_name);

Example :

Alter table Employee ADD primary key(Dept_no CHAR(10));

Note :

Before using Alter Table to add primary key make sure that the field is defined as NOT NULL - in other words, NULL cannot be an accepted value for that field.

Foreign key constraint

Tables reference one another through common fields to ensure validity of reference, referential integrity is used.
Referential Integrity is a system of rules that a DBMS uses to ensure that relationships between records in related tables are valid, and that users don't accidentally delete or change related data.
Referential integrity is ensured through FOREIGN KEY constraint.

Example :
Table : Employee

Column name	Constraint
Emp_id	Primary key
Emp_name	
Salary	

Table : Department

Column name	Constraint
Dept_no	Primary key
Dept_Emp_id	Primary key
Experience	

Example
Create table employee (emp_id integer primary key,emp_name char(25),salary float(10,2)):

```
Create table Department
(Dept_no integer primary key,Dept_Emp_id integer primary key,Experience double
foreign key(Dept_Emp_id) references Employee(Emp_id));
```

Foreign key constraint - Alter table command

Syntax :

```
Alter table table_name
ADD FOREIGN KEY(column-to-be-designated-as-foreign-key>)
references master-table(primary-key-of-master-table);
```

Example

```
Alter table Department
ADD foreign key(Dept_Emp_id) references
Employee(Emp_id);
```

Alter Table command - change

To change the name of one of the columns in a table CHANGE clause of ALTER TABLE command can be used:

ALTER TABLE CHANGE(COLUMN) old_col_name new_col_name;

Example :

ALTER TABLE Department CHANGE Experience Years_of_service integer;

DROP TABLE - IF EXISTS -Clause

DROP TABLE command of SQL is DDL command that lets the user to drop a table from the database.

SYNTAX :

DROP TABLE [IF EXISTS] <TABLE_NAME>;

EXAMPLE :

DROP TABLE Department;

DROP TABLE IF EXISTS Department;

SQL JOINS

An SQL join is a query that fetches data from two or more tables whose records are joined with one another based on a condition.

Syntax:

```
SELECT <field_list>
FROM <table1>,<table2>[,<table3>...]
WHERE <join condition for the tables>
```

Example :(EQUI-JOIN)

```
SELECT Emp_id,Dept_no,Years_of_service from Employee e,Department d
WHERE  Dept_no=1001 and e.Emp_id = Department.Dept_Emp_id;
```

Tables : Employee, Orders

Structure of Employee :

Field	Type	Null	Key	Default	Extra
Id	int	NO	PRI	NULL	
Name	char(30)	YES		NULL	
Salary	float	YES		NULL	

Structure of Orders :

Field	Type	Null	Key	Default	Extra
ordid	int	NO	PRI	NULL	
emp_id	int	YES		NULL	
amount	float	YES		NULL	

Types of JOINS

Cartesian Product : An SQL join query without any join condition returns all the records of joined with all the records of the other table.
It is known as a Cartesian Product also called CROSS JOIN.

Example : select id,name,ordid,amount from employee,orders;

id	name	ordid	amount
101	Sri	1001	3000
102	Helen	1001	3000
103	Karan	1001	3000
101	Sri	1002	560
102	Helen	1002	560
103	Karan	1002	560
101	Sri	1008	5600
102	Helen	1008	5600
103	Karan	1008	5600

Types of JOINS

Equi join : An SQL join query that joins two or more tables based on a condition using equality operator.

Example :
select id,name,salary,ordid from employee,orders where employee.id = orders.emp_id;

id	name	salary	ordid
102	Helen	55000	1001
101	Sri	45000	1002
103	Karan	85000	1008

Types of JOINS

Inner join : An INNER JOIN implements an equi join. In this join, only those rows are returned from both the tables that satisfy the join condition. The join conditions can match records based on one or more columns.

Example :

select id,name,ordid from employee left join orders on employee.id=orders.emp_id;

id	name	ordid
101	Sri	1002
102	Helen	1001
103	Karan	1008

Types of JOINS

Left join : The LEFT JOIN is a particular type of join that selects rows from both left and right tables that are matched, plus all rows from the left table even with no matching rows found in the right table.
Example :

select id,name,ordid from orders left join employee on employee.id=orders.emp_id;

id	name	ordid
102	Helen	1001
101	Sri	1002
NULL	NULL	1004
103	Karan	1008

Types of JOINS

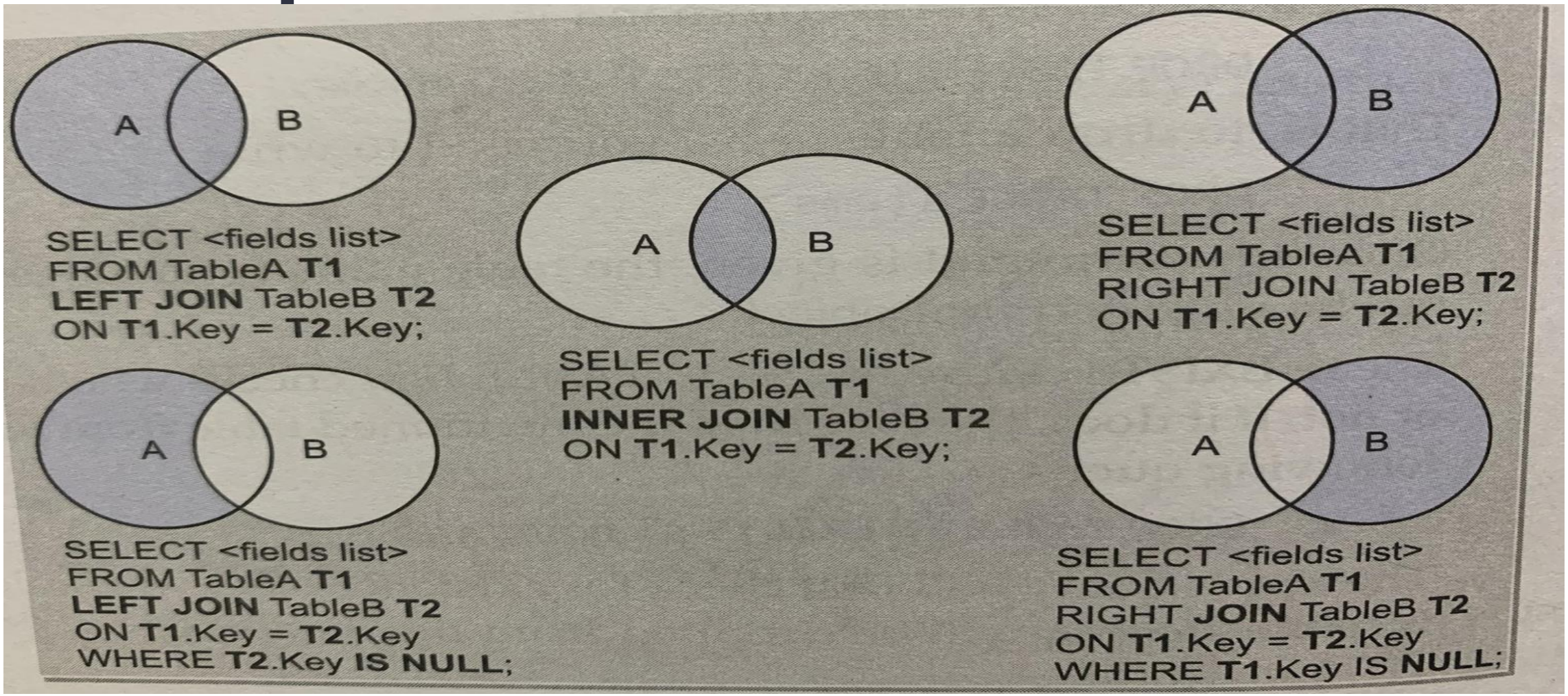
Right join : The RIGHT JOIN is a particular type of join that selects rows from both left and right tables that selects rows from both left and right tables that are matched, plus all rows from the right table even with no matching rows found in the left table.

Example :

SELECT ID,NAME,ORDID FROM EMPLOYEE RIGHT JOIN ORDERS ON EMPLOYEE.ID=ORDERS.EMP_ID;

ID	NAME	ORDID
102	Helen	1001
101	Sri	1002
NULL	NULL	1004
103	Karan	1008

Types of JOINS



Exercise

Consider the following table : GARMENT write SQL commands for i) to iv)

GCODE	GNAME	SIZE	COLOR	PRICE
101	SKIRT	XL	PINK	678.00
102	T-SHIRT	XXL	WHITE	870.00
103	TROUSERS	L	BROWN	1300.00
104	LADIES TOP	M	BLACK	1000.00
105	JEANS	L	BLUE	1400.00

i) To display names of those garments that are available in XL size
ii) To display codes and names of those garments that have their names ending with ‘RT’
iii) To display garment names, codes and prices of those garments that have price in the range 1000.00 to 1500.00 (both values included)
iv) To change the color of garment with code as 104 to WHITE

Exercise

Consider the following table : GARMENT write SQL commands for i) to iv)

i) To display names of those garments that are available in XXL size

Select GNAME from GARMENT WHERE SIZE = ‘XXL’;

ii) To display codes and names of those garments that have their names ending with ‘RT’

Select GCODE,GNAME from GARMENT WHERE GNAME LIKE ‘%RT’;

iii) To display garment names, codes and prices of those garments that have price in the range 1000.00 to 1500.00 (both values included)

Select GNAME,GCODE,PRICE from GARMENT WHERE PRICE BETWEEN 1000.00 AND 1500.00;

iv) To change the color of garment with code as 104 to WHITE

Update Garment set color =’White’ WHERE GCODE = 104;

Exercise

Consider the following table : SOFTDRINK write SQL commands for i) to iv)

DRINKCODE	DNAME	PRICE	CALORIES
101	LIME AND LEMON	20.00	150
102	ORANGE SQUASH	45.00	115
103	APPLE FIZZ	49.00	130
104	AAM PANNA	23.00	110
105	MANGO DELIGHT	38.00	138

i) To display names and drink codes of those drinks that have more than 120 calories
ii) To display drink codes,names and calories of all drinks in descending order of calories
iii) To display names and price of drinks that have price in the range 12 to 18(both inclusive)
iv) To increase the price of all drinks by 10%

Exercise

i) To display names and drink codes of those drinks that have more than 120 calories

select dname,drinkcode from softdrink where calories >120;

ii) To display drink codes,names and calories of all drinks in descending order of calories

select drinkcode,dname,calories from softdrink order by calories desc;

iii) To display names and price of drinks that have price in the range 12 to 18(both inclusive)
select dname,price from softdrink where price between 12 and 18;

iv) To increase the price of all drinks by 10%

update softdrink set price = price + 0.10*price;

Ordering data on multiple columns

Syntax:
SELECT FIELDNAME1,FIELDNAME2,.... FROM TABLENAME1 ORDER BY <FIELDNAME1> [ASC/DESC],<FIELDNAME2>[ASC/DESC];

EXAMPLE

```
mysql> SELECT * FROM STUDENT;
+--------+-----------+-------+
| studid | name      | marks |
+--------+-----------+-------+
|      1 | Abinav    | 56.70 |
|      2 | Adithya M | 76.70 |
|      3 | Kanish    | 87.67 |
|      4 | Thilak    | 47.39 |
|      9 | Punith    | 57.48 |
+--------+-----------+-------+
```

```
mysql> select * from student order by marks asc,name asc;
+-------+----------+-------+
| studid | name      | marks |
+-------+----------+-------+
|      4 | Thilak    | 47.39 |
|      1 | Abinav    | 56.70 |
|      9 | Punith    | 57.48 |
|      2 | Adithya M | 76.70 |
|      3 | Kanish    | 87.67 |
+-------+----------+-------+
```

ORDERING DATA BASED ON EXPRESSION

```
mysql> SELECT * FROM STUDENT;
+--------+-----------+-------+
| studid | name      | marks |
+--------+-----------+-------+
|      1 | Abinav    | 56.70 |
|      2 | Adithya M | 76.70 |
|      3 | Kanish    | 87.67 |
|      4 | Thilak    | 47.39 |
|      9 | Punith    | 57.48 |
+--------+-----------+-------+
```

```
mysql> SELECT STUDID,NAME,MARKS*0.35 FROM STUDENT WHERE MARKS >70 ORDER BY NAME,MARKS*0.35;
+--------+-----------+-------------+
| STUDID | NAME      | MARKS*0.35 |
+--------+-----------+-------------+
|      2 | Adithya M |    26.8450 |
|      3 | Kanish    |    30.6845 |
+--------+-----------+-------------+
```

SELECT STUDID,NAME,MARKS*0.35 AS N FROM STUDENT WHERE MARKS >70 ORDER BY NAME,N;

SPECIFYING CUSTOM SORT ORDER

```
mysql> SELECT * FROM STUDENT;
+--------+-----------+-------+-------+
| studid | name      | marks | grade |
+--------+-----------+-------+-------+
|      1 | Abinav    | 56.70 | B     |
|      2 | Adithya M | 76.70 | A     |
|      3 | Kanish    | 87.67 | O     |
|      4 | Thilak    | 47.39 | C     |
|      9 | Punith    | 57.48 | B     |
+--------+-----------+-------+-------+
```

```
mysql> SELECT * FROM STUDENT ORDER BY FIELD (GRADE,'C','B','O','A');
+--------+-----------+-------+-------+
| studid | name      | marks | grade |
+--------+-----------+-------+-------+
|      4 | Thilak    | 47.39 | C     |
|      1 | Abinav    | 56.70 | B     |
|      9 | Punith    | 57.48 | B     |
|      3 | Kanish    | 87.67 | O     |
|      2 | Adithya M | 76.70 | A     |
+--------+-----------+-------+-------+
```

AGGREGATE FUNCTIONS

Group functions or Aggregate functions work upon groups of rows, rather than on single rows - these functions are sometimes also called multiple row functions.

Many group functions accept the following options:

DISTINCT - this option causes a group function to consider only distinct values of the argument expression.
ALL - this option causes a group function to consider all values including all duplicates.

avg () - computes the average of given data

Syntax :

avg([distinct / all n] - returns average value of parameter(s) n.

Argument Type - numeric

Return Type - numeric

Example :

```
mysql> select * from student;
+--------+-----------+-------+-------+
| studid | name      | marks | grade |
+--------+-----------+-------+-------+
|      1 | Abinav    | 56.70 | B     |
|      2 | Adithya M | 76.70 | A     |
|      3 | Kanish    | 87.67 | O     |
|      4 | Thilak    | 47.39 | C     |
|      5 | Indu      | 57.48 | B     |
|      9 | Punith    | 57.48 | B     |
+--------+-----------+-------+-------+
```

```
mysql> select avg(distinct marks) 'Average' from student;
+-----------+
| Average   |
+-----------+
| 65.188000 |
+-----------+
```

```
mysql> select avg(marks) 'Average' from student;
+------------+
| Average    |
+------------+
| 63.903333 |
+------------+
```

COUNT() - counts the number of rows in a given column or expression

Syntax :

COUNT({*[DISTINCT/ALL] expr})

Returns the number of rows in the query.

If you specify argument expr, this functions returns rows where expr is not null.

You can count either all rows or only distinct values of expr.

If * is specified the function return all rows, including duplicates and nulls.

```
mysql> select * from student;
+--------+-----------+-------+-------+
| studid | name      | marks | grade |
+--------+-----------+-------+-------+
|      1 | Abinav    | 56.70 | B     |
|      2 | Adithya M | 76.70 | A     |
|      3 | Kanish    | 87.67 | O     |
|      4 | Thilak    | 47.39 | C     |
|      5 | Indu      | 57.48 | B     |
|      9 | Punith    | 57.48 | B     |
+--------+-----------+-------+-------+
```

```
mysql> select count(distinct marks) 'Marks' from student;
+-------+
| Marks |
+-------+
|     5 |
+-------+
```

```
mysql> select count(distinct marks>60) 'Marks' from student;
+-------+
| Marks |
+-------+
|     2 |
+-------+
```

MAX() - this function returns the maximum value from a given column or expression

Syntax :

MAX([DISTINCT/ALL] expr)

Returns the maximum value of argument expr

```
mysql> select * from student;
+--------+-----------+-------+-------+
| studid | name      | marks | grade |
+--------+-----------+-------+-------+
|      1 | Abinav    | 56.70 | B     |
|      2 | Adithya M | 76.70 | A     |
|      3 | Kanish    | 87.67 | O     |
|      4 | Thilak    | 47.39 | C     |
|      5 | Indu      | 57.48 | B     |
|      9 | Punith    | 57.48 | B     |
+--------+-----------+-------+-------+
```

```
mysql> select max(marks) 'Maximum Marks' from student;
+---------------+
| Maximum Marks |
+---------------+
|         87.67 |
+---------------+
```

MIN() - this function returns the minimum value from a given column or expression

Syntax :

MIN([DISTINCT/ALL] expr)

Returns the minimum value of argument expr

```
mysql> select * from student;
+--------+----------+-------+-------+
| studid | name     | marks | grade |
+--------+----------+-------+-------+
|      1 | Abinav   | 56.70 | B     |
|      2 | Adithya M| 76.70 | A     |
|      3 | Kanish   | 87.67 | O     |
|      4 | Thilak   | 47.39 | C     |
|      5 | Indu     | 57.48 | B     |
|      9 | Punith   | 57.48 | B     |
+--------+----------+-------+-------+
```

```
mysql> select min(marks) 'Minimum Marks' from student;
+--------------+
| Minimum Marks |
+--------------+
|         47.39 |
+--------------+
```

SUM() - this function returns the sum of values in a given column or expression

Syntax :

SUM([DISTINCT/ALL] n)

Returns the sum of values of n

```
mysql> select * from student;
+--------+----------+-------+-------+
| studid | name     | marks | grade |
+--------+----------+-------+-------+
|      1 | Abinav   | 56.70 | B     |
|      2 | Adithya M| 76.70 | A     |
|      3 | Kanish   | 87.67 | O     |
|      4 | Thilak   | 47.39 | C     |
|      5 | Indu     | 57.48 | B     |
|      9 | Punith   | 57.48 | B     |
+--------+----------+-------+-------+
```

```
mysql> select sum(marks) from student;
+-------------+
| sum(marks)  |
+-------------+
|      383.42 |
+-------------+
```

```
mysql> select sum(distinct marks) 'Total Marks' from student;
+-------------+
| Total Marks |
+-------------+
|      325.94 |
+-------------+
```

GROUPING RESULT - GROUP BY

GROUP BY clause combines all those records that have identical values in a particular field or a group of fields.
GROUP BY clause is used in SELECT command to divide the table into groups.
Grouping can be done by a column name, or with aggregate functions in which case the aggregate produces a value for each group.

```
mysql> select grade,count(*) from student group by grade;
+-------+----------+
| grade | count(*) |
+-------+----------+
| B     |        3 |
| A     |        1 |
| O     |        1 |
| C     |        1 |
+-------+----------+
```

```
mysql> select grade,count(*) 'Total count' from student group by grade;
+-------+-------------+
| grade | Total count |
+-------+-------------+
| B     |           3 |
| A     |           1 |
| O     |           1 |
| C     |           1 |
+-------+-------------+
```

Nested Groups-Grouping on Multiple Columns

```
mysql> select * from employee;
+-----+---------+--------+-----------+------+
| Id  | Name    | Salary | desig     | dept |
+-----+---------+--------+-----------+------+
| 101 | Sri     |  45000 | Manager   |   10 |
| 102 | Helen   |  55000 | Manager   |   10 |
| 103 | Karan   |  85000 | President |   20 |
| 104 | Yogesh  |  78000 | President |   10 |
| 105 | Krish   |  25000 | Clerk     |   20 |
+-----+---------+--------+-----------+------+
```

```
mysql> select dept,count(id) 'Total employees' from employee group by dept,desig;
+------+-----------------+
| dept | Total employees |
+------+-----------------+
|   10 |               2 |
|   20 |               1 |
|   10 |               1 |
|   20 |               1 |
+------+-----------------+
```

```
mysql> select count(id) 'Total employees',dept from employee group by dept;
+-----------------+------+
| Total employees | dept |
+-----------------+------+
|               3 |   10 |
|               2 |   20 |
+-----------------+------+
```

Placing conditions on group - Having clause

The HAVING clause places conditions on groups in contrast to WHERE clause that places conditions on individual rows.
WHERE conditions cannot include aggregate functions, HAVING conditions can do so.

```
mysql> select dept,count(id)'Total employees',desig from employee group by dept,desig;
+------+-----------------+-----------+
| dept | Total employees | desig     |
+------+-----------------+-----------+
|   10 |               2 | Manager   |
|   20 |               1 | President |
|   10 |               1 | President |
|   20 |               1 | Clerk     |
+------+-----------------+-----------+
```

```
mysql> select dept,count(id)'Total employees',desig from employee group by dept,desig having count(id)>1;
+------+-----------------+---------+
| dept | Total employees | desig   |
+------+-----------------+---------+
|   10 |               2 | Manager |
+------+-----------------+---------+
```

Placing conditions on group - Having clause with BETWEEN operator

```
mysql> select dept,count(id)'Total employees',desig from employee group by dept,desig having count(id) between 1 and 2;
+------+---------------+----------+
| dept | Total employees | desig    |
+------+---------------+----------+
|   10 |             2 | Manager  |
|   20 |             1 | President |
|   10 |             1 | President |
|   20 |             1 | Clerk    |
+------+---------------+----------+
```

EXERCISES

Consider the following tables GAMES and PLAYER. Write SQL commands for the following i) to iv) questions and also give the output for v) to viii)

gcode	gname	number	prizemoney	scheduledate
101	Carom Board	2	5000	2004-01-23
102	Badminton	2	12000	2003-12-12
103	Table Tennis	4	8000	2004-02-14

pcode	name	gcode
1	Ahmed	101
2	Ravi	108
3	Jessy	101
4	Karim	103

i) To display the name of all games with their codes
ii) To display details of those games which are having prizemoney more than 7000
iii) To display the contents of the games table in ascending order of schedule date
iv) To display sum of prizemoney for each of the number of participation grouping (as show in column number 2)
v) select count(distinct number) from games;
vi) select max(scheduledate),min(scheduledate) from games;
vii) select sum(prizemoney) from games;
viii) select distinct gcode from player;

EXERCISES

Consider the following tables GAMES and PLAYER. Write SQL commands for the following i) to iv) questions and also give the output for v) to viii)

gcode	gname	number	prizemoney	scheduledate
101	Carom Board	2	5000	2004-01-23
102	Badminton	2	12000	2003-12-12
103	Table Tennis	4	8000	2004-02-14

pcode	name	gcode
1	Ahmed	101
2	Ravi	108
3	Jessy	101
4	Karim	103

i) To display the name of all games with their codes

```
select gamename,gcode from games;
```

ii) To display details of those games which are having prizemoney more than 7000

```
select * from games where prizemoney>7000;
```

iii) To display the contents of the games table in ascending order of schedule date

```
select * from games order by scheduledate;
```

iv) To display sum of prizemoney for each of the number of participation grouping (as show in column number 2)

```
select sum(prizemoney) from games group by number;
```

EXERCISES

Consider the following tables GAMES and PLAYER. Write SQL commands for the following i) to iv) questions and also give the output for v) to viii)

v) select count(distinct number) from games;

```
+-----------------------+
| count(distinct number) |
+-----------------------+
|                     2 |
+-----------------------+
```

vi) select max(scheduledate),min(scheduledate) from games;

```
+------------------+------------------+
| max(scheduledate) | min(scheduledate) |
+------------------+------------------+
| 2004-02-14       | 2003-12-12       |
+------------------+------------------+
```

vii) select sum(prizemoney) from games;

```
+-----------------+
| sum(prizemoney) |
+-----------------+
|           25000 |
+-----------------+
```

viii) select distinct code from player;

```
+-------+
| gcode |
+-------+
|   101 |
|   108 |
|   103 |
+-------+
```