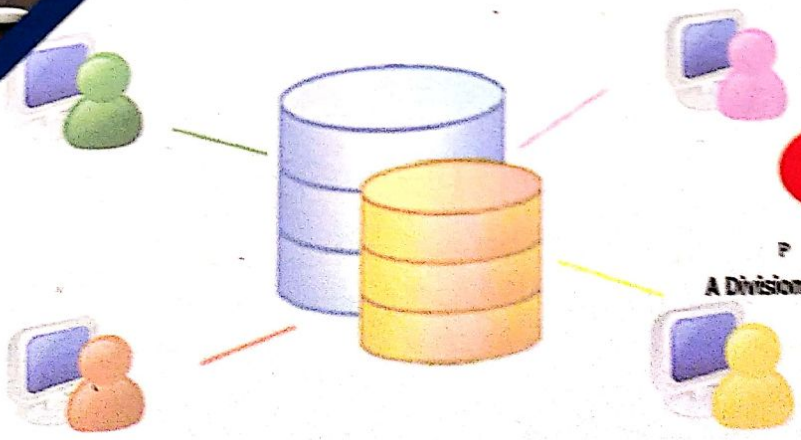
Database Management System-II

with CC-212 Practicals

BCA Semester-4 | Subject CC-208

- Introduction to SQL
- Transaction Management and Concurrency Control
- Distributed Database Management System
- Advanced SQL



COMPUTER WORLD PUBLICATION
A Division of Live Education System Pvt. Ltd.

BCA | Semester-4

Database Management System-II

with CC-212 Practicals

Subject Code : CC-208 & CC-212 (Practicals)

Dr. Dharmeshkumar Bhavsar
PhD [Computer Science]
Director, Chimanbhai Patel Institute of Computer Applications,
SG Highway, Ahmedabad.

Dr. Bhavik Pandya
Ph.D [Computer Science]
Navgujarat College of Computer Applications,
Ashram Road, Ahmedabad

Shri Keyur Shah
MCA [Master in Computer Application], M.Phil.
System Analyst-GTU, Formerly Coordinator- CPICA.

Prof. Sejal Vaghela
MCA (Master in Computer Applications)
Lokmanya College of Computer Applications,
Satellite, Ahmedabad

COMPUTER® WORLD
PUBLICATION
A Division of Live Education System Pvt. Ltd.

COMPUTER WORLD
43, 5th Floor, SANIDHYA Complex, Nr. M. J. Library,
Opp. Sanyas Ashram, Ashram Road, Ahmedabad-09.
Ph. : +91- 79 26580723, 26580823 | Mobile : 9724011150, 9725022917
URL : www.computerworld.ind.in | e-Mail :info@computerworld.ind.in

Publish By :
COMPUTER World
A Division of Live Education System Pvt. Ltd.
An ISO 9001 : 2008 Certified Company
Kalpesh Patel - kalpesh@computerworld.ind.in

Prepare by : Computer World Research Department
Dr. Dharmeshkumar Bhavsar, Dr. Bhavik Pandya,
Shri Keyur Shah, Prof. Sejal Vaghela, Ravindra Parmar

Price : ₹ 100/-

ISBN : 978-93-88092-20-3

Book Code : CEBCA114

Edition : 1st

Contact : 43, 5th Floor, Sanidhya Complex, Opp. Capital Comm. Center, Ashram Road, Ahmedabad-380009
Phone : +91-79-26580823 / +91-79-26580723, | Mobile : 97240 11150, 97250 22917
URL : www.computerworld.ind.in | e-Mail : info@computerworld.ind.in, info.computerworld@gmail.com

INDEX

Unit -1

Introduction to SQL

- Introduction to SQL
- Data Definition Commands
- Data Manipulation Commands
- Select Query
- Advanced Data Definition Commands

Unit-1 Intoduction to SQL

1.1 Introduction to SQL:

SQL stands for Structured Query Language. It is a language consisting of commands to work on data stored in RDBMS (Relational Database Management System). It allows us to create database and table structures, to perform basic data management operations and to perform queries on it. It is mandatory for every RDBMS system to support SQL.

SQL is divided into following sublanguages:

SQL				
DDL (Data Definition language)	DQL (Data Query language)	DML (Data Manipulation language)	TCL (Transaction Control language	DCL (Data Control language)
CREATE ALTER DROP TRUNCATE RENAME	SELECT	INSERT UPDATE DELETE	COMMIT ROLLBACK SAVEPOINT	GRANT REVOKE

1. DDL (Data Definition Language)

These statements are used to create tables and databases and define field properties or table properties. Examples of commands that fall in this category are CREATE,ALTER and DROP statements

2. DML (Data Manipulation Language)

The statements that falls under this category are used to update data or add or remove data from tables. UPDATE, DELETE and INSERT commands fall under this category.

3. DCL (Data Control Language)

It is used to control who access the data. The commands that come under this category are GRANT and REVOKE

4. TCL (Transaction Control Language)

This language is used to commit data and restore data. COMMIT and ROLLBACK falls under this category.

5. DQL (Data Query Language)

This is to retrieve data from RDBMS. SELECT statement falls in this category.

1.2 Data Definition Commands:

Command or options	Description
CREATE	Creates a new database object in the database schema. Database objects are: table, view, constraint, index, sequence, synonym etc.
ALTER	It is used to make changes in already existing database objects. For example, if we want to add a new column in a table or we want to add a constraint in a table.
DROP	It is used to delete database object permanently from database. For example, deleting a table or a view or a constraint.
TRUNCATE	It is used with table. It deletes table data and keeps table structure as it is.
RENAME	It is used to change name of database object.

Data Manipulation commands:

Command or Option	Description
INSERT	It is used to insert a row in a table
UPDATE	It is used to make changes in table data
DELETE	It is used to delete rows from a table.

1.2.1 Data Types:

Data types are required while creating tables. In a table, there are columns. Every column contains datatype and size. This datatype and size defines what type of data and what size of data can be stored in a column for every row. Few very common data types are listed below:

Datatype	Description
Number(P,S) or numeric(P,S)	It is used to store numbers. P indicates maximum length including decimal place and sign and S indicates no. of digits after decimal point. For example, number(7,2) can store numbers with 2 digits after decimal point and maximum 7 digits long. Ex: 1234.56, -135.77.
Integer or int	It is used to store numbers without decimal places.

Smallint	It is like integer but upto six digits only.
Dec(P,S) or decimal(P,S)	It is like number but the storage length is minimum specification.
Float	It is used to store floating point numbers.
Real	It is like float.
Char(size)	It is used to store character data i.e. strings of fixed length. Max. 255 characters are allowed. Every character will occupy one byte.
Varchar(size) / Varchar2(size)	Varchar stands for variable character. It is dynamic allocation i.e. if we have defined varchar(30) and will give 10 character value, then only 10 bytes are occupied. So it saves memory. If we use Varchar(max), then any no. of characters can be stored.
Nvarchar(size) and Nvarchar2(size)	It stores character data. It is especially used for multi-lingual support.
Date	It is used to store date. Default date format in oracle is dd-mon-yy.
Time	It is used to store time.
Long	It is used to store character data upto 2 GB. Only one long column is allowed per table.
Raw / Long raw	The Raw / Long raw data types are used to store binary data such as pictures or images. Raw data type can have upto 255 bytes. Long raw data can be upto 2 GB.

1.2.2 Creating Table Structures:

Table is one of the database object. It stores the data in a tabular format i.e. row and column format. To store data, we first need to create a table using CREATE TABLE command which will define table structure i.e. columns, their datatypes, size etc. Basic syntax of CREATE TABLE syntax is shown below:

```
CREATE TABLE tablename (
        Column1-name datatype(size),
        column2-name datatype(size),
        . . .
        );
```

Example:

```
SQL> Create table employee (
  empno number(4),
```

```
ename varchar2(30),
designation varchar2(30),
doj date,
salary number(8,2),
deptno number(2) );
```

Table created:

In the example above, we have created a table named employee with 6 columns: employee number, employee name, designation, date-of-join, salary and its department number. Note that employee number is one of the column whose value will be different for every employee. It is good to add at least one column in every table which have unique values for every record. Such column value can be used to identify every row i.e. employee.

```
SQL> create table dept (
   deptno number(2),
   dname varchar2(30),
   locationid number(2)
  );
```

Table created:

In the above example, we have created another table dept with 3 columns: department number, department name and location id – this column is used to identify a location of a department because in large companies departments are located at various places… production dept at Ahmedabad, marketing at delhi, sales at Noida, accounts at Vadodara etc.

Important:

To see the list of tables created in our account,

```
SQL> Select * from tab;
```

is used – where tab stands for table space.

Important:

To see the table structure (column names, data types sizes etc.),

```
SQL> desc table-name;
```

is used – where desc stands for describe.

For example,

```
SQL> desc employee ;
```

Name	Null?	Type
EMPNO		NUMBER(4)
ENAME		VARCHAR2(30)
DESIGNATION		VARCHAR2(30)
DOJ		DATE
SALARY		NUMBER(8,2)
DEPTNO		NUMBER(2)

1.2.3 SQL Constraints:

All business of the world run on business data being gathered, stored and analysed. On such data, business rules are applied prior to store it in database tables. For example, all employees working in a company can have a salary of not less than 10000. Such rules have to be enforced on data stored. Same way, every employee must have a employee number and should not be repeated. Business rules, which are enforced on data being stored in a table are called Constraints.

SQL constraints are broadly divided into three categories:

Domain Integrity Constraints	Entity Integrity Constraints	Referential Integrity Constraints
NOT NULL CHECK DEFAULT	PRIMARY KEY UNIQUE	FOREIGN KEY

Applying Data Constraints:

Every RDBMS system permits data constraints to be attached to table columns via SQL syntax that checks data for integrity before storing it in tables. So when a data entered, it has to pass this checks then only it is stored in a particular column otherwise the data is rejected. Even if a single column of record being entered into table fails a constraint, the entire record is rejected and not stored in a table.

Both **Create Table** and **Alter Table** commands can be used to define constraints to a table column. Entity and referential integrity constraints can be applied at column-level (i.e. while defining column details) or table-level (i.e. separate constraint definition is given in a table)

Domain Integrity Constraints:

- **Check:**

Check integrity constraint is used to validate the column data when it is entered. Generally, some condition is defined with check constraint for validation. If the condition is found to be false, an error message is generated, and the data are not accepted. Check constraint can be added with column definition as under:

Column-name datatype(size) check (condition)

For example, salary should be greater than 10000 and deptno should be 10,20,30,40,50 only.

```
SQL> create table employee (
  empno number(4),
  ename varchar2(30),
  designation varchar2(30),
  doj date,
  salary number(8,2) check ( salary > 10000),
  deptno number(2) check (deptno in (10,20,30,40,50))
  );
```

Important:

If employee table is already created, then we can add check constraint using **Alter Table…add constraint** command.

For example,

alter table employee add constraint deptno_chk check(deptno in (10,20,30,40,50))

alter table employee add constraint salary_chk check(salary > 10000);

- **Default:**

Default constraint is used to assign a default value to a column. So if user will not enter any value for this column while inserting a row then column will keep default value.

Column-name datatype(size) default default-value

Here, the default-value should be of same type that of datatype of a column.

For example, by default designation should be 'programmer'.

```
SQL> create table employee (
  empno number(4),
  ename varchar2(30),
  designation varchar2(30) default 'programmer' ,
  doj date,
  salary number(8,2) check ( salary > 10000),
  deptno number(2) check (deptno in (10,20,30,40,50))
  );
```

Important:

If employee table is already created, then we can add default constraint using Alter **Table…modify** command.

SQL> alter table employee modify (designation varchar2(30) default 'programmer') ;

➢ **NOT NULL:**

Not null constraint ensures that a column does not accept null value (null means no value. It is different from 0 or blank). In other words, user must have to enter some value for this column, otherwise row cannot be entered in a table.

Column-name datatype(size) not null

For example, ename should not be null.

```
SQL> create table employee (
   empno number(4),
   ename varchar2(30) not null,
   designation varchar2(30) default 'programmer' ,
   doj date,
   salary number(8,2) check ( salary > 10000),
   deptno number(2) check (deptno in (10,20,30,40,50))
   );
```

Important:

If employee table is already created, then we can add not null constraint using **Alter Table…modify** command.

SQL> alter table employee modify(ename varchar2(30) not null);

Entity Integrity Constraints:

➢ **Primary Key:**

Generally, in every table we create one column with primary key constraint. Primary key is used to distinguish every row in a table. Primary key constraint doesn't allow to enter null value and allows only unique values in a column. This constraint can be added at column-level and table-level.

Note: Only one PRIMARY KEY constraint is allowed per table.

At column-level:

Column-name datatype(size) primary key

For example,

In employee table, we want to set empno as a primary key.

```
SQL> create table employee (
   empno number(4) primary key,
   ename varchar2(30) not null,
   designation varchar2(30),
   doj date,
   salary number(8,2),
   deptno number(2)
   );
```

At table-level :

Primary key (column-name)

OR

Constraint constraint-name primary key(column-name)

For example,

```
SQL> create table employee (
   empno number(4),
   ename varchar2(30),
   designation varchar2(30),
   doj date,
   salary number(8,2),
   deptno number(2),
   constraint emp_pk primary key(empno)
   );
```

Important:

If table already exists, then we can use **Alter table... Add** option to add table-level constraint.

```
SQL> alter table employee add primary key(empno);
                    OR
SQL> alter table employee add constraint emp_pk primary key(empno);
```

- **UNIQUE:**

Unique constraint does not allow duplicate values in a column but allows multiple NULL values in a column (Remember that NULL means unspecified value so NULL is not equal to NULL). UNIQUE constraint can be added at column-level and table-level.

Things to remember for UNIQUE:

- Not allow duplicate values
- Unique index is created automatically

- Can have more than one UNIQUE constraints in a table which is not possible in primary key.
- Can be composite up to 16 columns.

At column-level:

Column-name datatype(size) UNIQUE

For example,

In employee table, we want to set empno as a primary key.

```
SQL> create table employee (
  empno number(4) primary key,
  ename varchar2(30) UNIQUE,
  designation varchar2(30),
  doj date,
  salary number(8,2),
  deptno number(2)
  );
```

At table-level :

UNIQUE (column-name)

OR

Constraint constraint-name UNIQUE(column-name)

For example,

```
SQL> create table employee (
  empno number(4),
  ename varchar2(30),
  designation varchar2(30),
  doj date,
  salary number(8,2),
  deptno number(2),
  constraint emp_uk UNIQUE (ename)
  );
```

Important:

If table already exists, then we can use **Alter table... Add** option to add table-level constraint.

```
SQL> alter table department add unique(dname);
```

OR

```
SQL> alter table department add constraint dept_uk unique(dname);
```

Referential Integrity Constraints:

- **Foreign Key:**

Foreign key is generally used to establish relationship between tables. A foreign key is a column or group of columns whose values are derived from the **primary key** or **unique key** of some other table or same table.

Table containing foreign key is called **detail table**. The table where primary or unique key is defined and referenced by the foreign key is known as **master table**. Note : you should create master table first to establish master-detail relationship between two tables.

Foreign key constraint can be defined at table level or column level. It can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

The master table can be referenced in the foreign key definition by using the clause REFERENCES table-name(column-name) while defining foreign key in the detail table.

At column-level:

```
Column-name datatype(size) REFERENCES table-name(column-name)
                                [ON DELETE CASCADE]
```

For example,

Add a foreign key in deptno column in emp table which will refer deptno column in dept table.

```
SQL> create table department (
  deptno number(2) primary key,
  dname varchar2(30) not null,
  locationid number(2)
  );
```

Table created.

```
SQL> create table employee (
  empno number(4) primary key,
  ename varchar2(30) not null,
  designation varchar2(30),
  doj date,
  salary number(8,2) check(salary > 10000),
  deptno number(2) references department(deptno)
  );
```

Table created.

Important: It is not mandatory to have same names for foreign key in detail table and primary key in master table but their datatype should be same.

At table-level (Can be used in CREATE TABLE OR ALTER TABLE):

FOREIGN KEY (detail table column-name)
REFERENCES master-table-name(column-name) [ON DELETE CASCADE];

For example,

Location table defines location of all depatments located in a company and department table maintains department details. Locationid in department table (detail table) should reference locationid in location master table.

```
SQL> create table location (
  locationid number(2) primary key,
  loc_detail varchar2(100)
  );
```

Table created.

SQL> **Alter table department add constraint dept_fk foreign key(locationid) references location(locationid);**

Table altered.

OR

```
SQL> create table department (
  deptno number(2) primary key,
  dname varchar2(30),
  locationid number(2),
  foreign key (locationid) references location(locationid)
  );
```

Table created.

Things to remember for Foreign key constraint:

- Rejects an INSERT or UPDATE of a value, if a corresponding value does not currently exist in the master table key
- If the ON DELETE CASCADE option is set, a DELETE operation applied in the master table will also delete all corresponding records in detail table as well.
- Rejects a DELETE from the master table if corresponding records in the detail table exists.
- Foreign key must reference PRIMARY KEY or UNIQUE column in master table

1.3 Data Manipulation Commands:

1.3.1 Adding Table Rows:

INSERT:

Insert command is used to add rows in a table. Its basic syntax is :

```
INSERT INTO table-name VALUES (value1,value2, … , valuen);
```

For example,

```
SQL> insert into employee values (1001, 'Vikram', 'Programmer', '10-oct-11', 22500,
10);
```

Another syntax which allows to enter data for few columns only is :

```
INSERT INTO table-name(column1-name,column2-name,…)
                    VALUES (value1,value2, … );
```

For example,

```
SQL> insert into employee(empno,ename,designation,deptno)
     values (1001,'Parth','Lab assistant', 20);
```

1.3.2 Saving Table Changes:

COMMIT:

Commit command is used to save the changes done so far. Its syntax is :

```
COMMIT;
```

1.3.3 Listing Table Rows:

SELECT:

SELECT command is used to list data from one or more table. Its basic syntax is :

```
SELECT column-list | * FROM table-list ;
```

For example,

```
SQL> select * from employee ;
```

This command will list all rows from employee table.

```
SQL> select empno, ename, designation from employee ;
```

This command will list employee number, name and designation for all rows.

1.3.4 Updating Table Rows:

UPDATE:

UPDATE command is used to modify data in a table. Its syntax is:

```
UPDATE    table-name
SET       column-name = new-value
```

[WHERE condition] ;

Note: If WHERE is not used then it will make changes in all rows.

For example,
Set salary of all employees as 12000 who are operators (i.e. whose designations are operators)

```
SQL> UPDATE employee
     SET salary = 12000
     WHERE designation = 'operator' ;
```

Set all department names as Production in department table

```
SQL> UPDATE department
     SET dname = 'Production' ;
```

Increase salary of Pratik by 1000.

```
SQL> UPDATE employee
     SET salary = salary + 1000
     WHERE ename = 'Pratik';
```

1.3.5 Restoring Table Contents:

ROLLBACK:

If you have not used COMMIT then ROLLBACK command is used to undo all changes done in table data so far. Its syntax is:

```
ROLLBACK;
```

Note: It can undo changes in table data not in table structures i.e. it can undo only INSERT, UPDATE and DELETE commands.

1.3.6 Delete Table Row:

DELETE:

It is used to delete rows from a table. Its syntax is:

```
DELETE FROM table-name
[WHERE conditions] ;
```

Note: If WHERE is not used then it will delete all rows from a table.

Example:

Delete all records from employee whose department no. is 20

```
SQL> DELETE FROM employee
```

```
WHERE deptno=20;
```

Delete all employees whose salary is between 7000 and 9000.

```
SQL> DELETE FROM employee
     WHERE salary >= 7000 and salary <= 9000
```

OR

```
SQL> DELETE FROM employee
     WHERE salary between 7000 and 9000.
```

1.4 SELECT QUERY:

1.4.1 With Conditional Restrictions:

We can select partial table contents by placing restrictions on the rows to be included in the output. This is done by using WHERE clause in SELECT statement.

```
SELECT      column-list | *
FROM        table-list
[WHERE      condition-list] ;
```

```
SQL> select * from department ;
```

DEPTNO	DNAME	LOCATIONID
10	production	1
20	stores	1
30	accounts	2
40	admin	3
50	marketing	3

If we don't use WHERE clause, then it will display all rows.

Examples with WHERE clause:

List all departments whose department no. is greater than 25.

```
SQL> select * from department where deptno > 25 ;
```

DEPTNO	DNAME	LOCATIONID
30	accounts	2
40	admin	3
50	marketing	3

List all departments whose name begins with a.

SQL> select * from department where dname like 'a%' ;

DEPTNO	DNAME	LOCATIONID
30	accounts	2
40	admin	3

List department names whose location id is 3.

SQL> select dname from department where locationid = 3 ;

DNAME
admin
marketing

List empno, ename and salary of all employees whose salary is 20000 or more.

SQL> select empno, ename, salary from employee where salary >= 20000 ;

EMPNO	ENAME	SALARY
1001	Vikram	22500
1005	Yashesh	45000

1.4.2 Arithmetic Operators:

Following is the list of arithmetic operators used in conditions.

Arithmetic operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
^	Power

Examples:

Display Bonus (12 % of salary) for all employees

SQL> select empno, ename, salary*0.12 "Bonus" from employee ;

EMPNO	ENAME	Bonus
1001	Vikram	2700

1002	Pratik	2220
1003	Shikha	1800
1004	Dipti	1980
1005	Yashesh	5400

Increase the salary of all employees by 500.

```
SQL> update employee set salary = salary + 500 ;
5 rows updated.
```

1.4.3 Logical Operators: AND, OR and NOT:

Logical operators are used with conditions. Sometimes, to filter data, we need to write multiple conditions. To combine more than one conditions, we can use logical operators AND and OR.

Logical operator NOT is used to inverse the result of a condition i.e. if the condition result is true, it will produce false and vice versa.

Logical operator	Syntax	Result
AND	Condition1 AND condition2	Both conditions should be true
OR	Condition1 OR condition2	Minimum one condition should be true
NOT	NOT condition	If condition is false, it will make it true and vice versa.

Examples:

Display empno, ename and designation of all employees whose designation is manager and works in department no. 10.

```
SQL> select empno, ename, designation from employee
        where designation = 'Manager' and deptno=10 ;

EMPNO   ENAME        DESIGNATION
---------  ----------  ----------
   1005  Yashesh      Manager
```

Display employee name, designation of all employees who are Programmer or Lab assistant.

```
SQL> select ename, designation from employee
        where designation = 'Programmer' or designation = 'Lab assistant' ;

ENAME          DESIGNATION
```

Vikram	Programmer
Pratik	Lab assistant

Display employee no, employee name and department no. of all employees who are not working in department no. 20.

```
SQL> select empno, ename, deptno from employee
        where not (deptno=20);
```

EMPNO	ENAME	DEPTNO
1001	Vikram	10
1003	Shikha	30
1004	Dipti	30
1005	Yashesh	10

1.4.4 Special Operators:

Operator	Meaning
BETWEEN...AND	Used to check whether column-value in a given range or not
IS NULL	Used to compare column value with NULL
LIKE	Used to compare column-value with strings using wildcard characters % (one or more characters) and _ (one character)
IN	Used to compare column-value with any value matching with given list.
EXISTS	Used to check whether a subquery returns any rows.

Examples:

Display employee name and salary of all employees whose salary is between 10000 and 20000

```
SQL> select ename, salary from employee
        where salary between 10000 and 20000 ;
```

ENAME	SALARY
Pratik	19000
Shikha	15500
Dipti	17000

Display employee no. and name of employees whose designation is not given.

```
SQL> select empno, ename from employee
        where designation is NULL ;

EMPNO      ENAME
---------  ---------------
1006       Shikha
```

Display employee no. and name of employees whose names begins with s.

```
SQL> select empno, ename from employee
        where ename like 'S%' or ename like 's%' ;

EMPNO      ENAME
---------  ---------------
1003       sarika
1006       Shikha
```

Display employee no. and name of employees whose names contains i as second character.

```
SQL> select empno, ename from employee
        where ename like '_i%' ;

EMPNO      ENAME
---------  ---------------------
1001       Vikram
1004       Dipti
```

Display employee names who are Programmer, Manager or Accountant.

```
SQL> select ename,designation from employee
        where designation in ('Programmer', 'Manager', 'Accountant')

ENAME        DESIGNATION
-----------  --------------------
Vikram       Programmer
Sarika       Accountant
Yashesh      Manager
```

1.5 Advanced Data Definition Commands:

These commands are useful if we want to make changes in a table structure. Previously, we have already seen DESC command to see table structure.

SQL> desc employee ;

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME	NOT NULL	VARCHAR2(30)
DESIGNATION		VARCHAR2(30)
DOJ		DATE
SALARY		NUMBER(8,2)
DEPTNO		NUMBER(2)

All changes in a table structure can be done using ALTER TABLE command.

1.5.1 Changing a Column's Data Type:

ALTER TABLE can change datatype of an existing column. Its syntax is:

```
ALTER TABLE table-name
      MODIFY (column-name datatype(size));
```

Example:

Set employee name size to 20 in employee table.

```
SQL> alter table employee modify (ename varchar2(20));

Table altered.
```

1.5.2 Changing a Column's Data Characteristic:

If the column to be changed already have data, you can change its characteristics if i does not alter data type. If it does not contain data, you can change datatype also Reducing column size may result in truncation of data. ALTER TABLE.... MODIFY command is used for this purpose.

Example:

Increase the salary column size to 10,2 from 8,2.

```
SQL>ALTER TABLE employee MODIFY (salary number(10,2)) ;
```

1.5.3 Adding a Column:

You can add column in existing table using ALTER TABLE... ADD command.

Example:

Add mobile number in employee table.

```
SQL> ALTER TABLE employee ADD (mobileno varchar2(13)) ;
```

Important: If you add a column in a table which already have data then the new column should not have NOT NULL option.

1.5.4 Dropping a Column:

You can delete a column from a table using ALTER TABLE .... DROP COLUMN command.

Example:

Drop mobileno column from employee table

```
SQL> ALTER TABLE employee DROP COLUMN mobileno ;
```

Important: You cannot drop a column which involves in foreign key relationship or it is an only column in a table.

1.5.5 Advanced Data Update:

All updates in the table data can be done using UPDATE command.

Example:

Increase employee salary by 10 percent who joined before 2015.

```
SQL> update employee
  set salary = salary + 0.10 * salary
  where doj < '01-jan-15' ;
2 rows updated.
```

Decrease salary of all employees by 5 percent who are working in department no. 30 or 40.

```
SQL> update employee
  set salary = salary - 0.5 * salary
  where deptno in (30,40) ;

2 rows updated.
```

1.5.6 Copying Parts of Table:

If we want to create a new table based on existing table with few columns and/or rows then we can use CREATE TABLE ..... AS command. Sometimes, we don't require to use whole table but need few columns/rows only to use frequently. In such case, we can create separate table from existing table using this command. Its syntax is:

```
CREATE TABLE new-table-name
AS
select-sql statement.
```

The output of select-sql statement will be the contents of new table.

Example:

Create a new table emp_desig which contains empno, designation and deptno only.

```
SQL> create table emp_desig
```

```
as
select empno, designation, deptno from employee ;

Table created.

SQL> select * from emp_desig ;

EMPNO      DESIGNATION          DEPTNO
---------  -------------------- ---------
    1001   Programmer                10
    1002   Lab assistant             20
    1003   Accountant                30
    1004   Sales executive           30
    1005   Manager                   10
    1006                             10

6 rows selected.
```

1.5.7 Adding Primary and Foreign Key Designations:

When we create new table, based on existing table, the new table doesn't contai constraints. To add primary key and foreign key constraints to any table, we can us ALTER TABLE ... ADD command. It is similar to adding a table-level constraint.

Example:

Define primary key on empno column in emp_desig table.

```
SQL> alter table emp_desig add primary key (empno) ;

Table altered.
```

Define foreign key on deptno of emp_desig which refers deptno of department table.

```
SQL> alter table emp_desig
  add foreign key (deptno) references department (deptno) ;

Table altered.
```

1.5.8 Deleting Table From The Database:

You can delete any table using DROP TABLE command. Its syntax is :

```
DROP TABLE table-name
```

Example:

Drop emp_desig table

```
SQL> drop table emp_desig;
```

Important: Drop table command will delete all rows and columns. Table will no longer exist. It is not possible to delete master table before deleting detail table.

Deleting Table Data Only:

Sometimes, we want to empty the table i.e. we want to delete all data but not the table structure. In such case, we can use TRUNCATE TABLE command. Its syntax is:

TRUNCATE TABLE table-name ;

Example:

SQL> TRUNCATE TABLE employee ;

It will delete all rows of employee table but columns remains as it is.

Practicals

SET-I

Create following tables and solve queries given below:

Passenger table

Field name	Data type and size	Description
Pid	Number(4)	Passenger id PRIMARY KEY.
Pname	Varchar2(30)	Passanger name NOT NULL
Age	Int	Passenger age

Bus table

Field name	Data type and size	Description
Bid	Number(3)	Bus id PRIMARY KEY
Color	Varchar2(15)	Bus color COLOR MUST BE 'red', 'yellow' or 'blue'

Queries:

1. Write create table statements with given details.
2. Insert at least 5 rows in each table
3. Add field gender in passenger table and the value must be either 'm' or 'f'
4. Change size of color field to 20 in bus table
5. Update color for the busid =11 with blue color.
6. Find passenger whose age is between 20 to 30.
7. Display all bus whose color is blue.

Solution 1

```
SQL> create table passenger (
  pid number(4) primary key,
  pname varchar2(30) not null,
  age int
  );
Table created.

SQL> create table bus (
  bid number(3) primary key,
  color varchar2(15) check (color in ('red','yellow','blue'))
  );

Table created.
```

Solution 2

```
SQL> insert into passenger values (&pid,'&pname',&age);
Enter value for pid: 1
Enter value for pname: rakesh
Enter value for age: 28
old   1: insert into passenger values (&pid,'&pname',&age)
new   1: insert into passenger values (1,'rakesh',28)

1 row created.

SQL> /
Enter value for pid: 2
Enter value for pname: dinesh
Enter value for age: 32
old   1: insert into passenger values (&pid,'&pname',&age)
new   1: insert into passenger values (2,'dinesh',32)
```

1 row created.

SQL> /
Enter value for pid: 3
Enter value for pname: disha
Enter value for age: 21
old 1: insert into passenger values (&pid,'&pname',&age)
new 1: insert into passenger values (3,'disha',21)

1 row created.

SQL> /
Enter value for pid: 4
Enter value for pname: vaishakhi
Enter value for age: 29
old 1: insert into passenger values (&pid,'&pname',&age)
new 1: insert into passenger values (4,'vaishakhi',29)
1 row created.

SQL> /
Enter value for pid: 5
Enter value for pname: dipak
Enter value for age: 35
old 1: insert into passenger values (&pid,'&pname',&age)
new 1: insert into passenger values (5,'dipak',35)

1 row created.

SQL> select * from passenger ;

PID	PNAME	AGE
1	rakesh	28
2	dinesh	32
3	disha	21
4	vaishakhi	29
5	dipak	35

SQL> insert into bus values (&bid,'&color');
Enter value for bid: 11
Enter value for color: yellow
old 1: insert into bus values (&bid,'&color')
new 1: insert into bus values (11,'yellow')

```
1 row created.

SQL> /
Enter value for bid: 12
Enter value for color: green
old   1: insert into bus values (&bid,'&color')
new   1: insert into bus values (12,'green')
insert into bus values (12,'green')
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C007023) violated

SQL> /
Enter value for bid: 12
Enter value for color: blue
old   1: insert into bus values (&bid,'&color')
new   1: insert into bus values (12,'blue')

1 row created.

SQL> /
Enter value for bid: 13
Enter value for color: red
old   1: insert into bus values (&bid,'&color')
new   1: insert into bus values (13,'red')

1 row created.

SQL> /
Enter value for bid: 14
Enter value for color: yellow
old   1: insert into bus values (&bid,'&color')
new   1: insert into bus values (14,'yellow')

1 row created.

SQL> /
Enter value for bid: 15
Enter value for color: blue
old   1: insert into bus values (&bid,'&color')
new   1: insert into bus values (15,'blue')

1 row created.
```

```
SQL> select * from bus ;

       BID COLOR
---------- --------------
        11 yellow
        12 blue
        13 red
        14 yellow
        15 blue
```

Solution 3

```
SQL> alter table passenger add (gender char(1) check (gender in ('m','f'))) ;

Table altered.
```

Solution 4

```
SQL> alter table bus modify (color varchar2(20) check (color in
        ('red','yellow','blue'))) ;

Table altered.
```

Solution 5

```
SQL> update bus
  set color='blue'
  where bid=11;

1 row updated.

SQL> select * from bus ;

       BID COLOR
---------- --------------------
        11 blue
        12 blue
        13 red
        14 yellow
        15 blue
```

Solution 6

```
SQL> select * from passenger where age between 20 and 30;
```

PID	PNAME	AGE
1	rakesh	28
3	disha	21
4	vaishakhi	29

Solution 7

```
SQL> select * from bus where color='blue' ;

       BID COLOR
---------- --------------------
        11 blue
        12 blue
        15 blue
```

SET – II

Create following tables and solve queries given below:

Emp table

Field name	Data type and size	Description
Eid	Number(4)	Employee id PRIMARY KEY
Ename	Varchar2(20)	Employee name NOT NULL

Company

Field name	Data type and size	Description
cid	Number(2)	Company id
Cname	Varchar2(30)	Company name NOT NULL
City	Varchar2(20)	Cannot start with 'z'

Works

Field name	Data type and size	Description
Eid	Number(4)	Employee id Foreign key to emp(eid)
cid	Number(2)	City id
Salary	Number(9,2)	Salary must be >= 20000 and <= 80000

Queries:

1. Create above tables with necessary constraints
2. Add PRIMARY KEY on cid in company table
3. Add foreign key on cid in works table which refers company(cid)
4. Insert at least 5 records in all tables
5. Add field age in emp table and must be between 18 and 60
6. List all employees whose names contains maximum upto 3 character
7. List all employees whose names starts with either 'a' or 'r'.
8. List all companies in ahmedabad.
9. Increase salary of all employees by 500.
10. Delete all companies which are in Vadodara.

Solution 1

```
SQL> create table emp (
   eid number(4) primary key,
   ename varchar2(20) not null
   );
Table created.
SQL> create table company (
   cid number(2),
   cname varchar2(30),
   city varchar2(20) check (city not like 'z%')
   );

Table created.

SQL> create table works (
   eid number(4) references emp(eid),
   cid number(2),
   salary number(9,2) check (salary between 20000 and 80000)
   );
Table created.
```

Solution 2

```
SQL> alter table company add primary key (cid) ;

Table altered.
```

Solution 3

```
SQL> alter table works add foreign key (cid) references company (cid) ;
Table altered.
```

Solution 4

```
SQL> insert into emp values (&eid,'&ename');
SQL> insert into company values (&cid, '&cname', '&city');
SQL> insert into works values (&eid, &cid, &salary);
```

Solution 5

```
SQL> alter table emp add (age number(3) check (age between 18 and 24))
```

Solution 6

```
SQL> select * from emp where ename like '___' ;

       EID ENAME                AGE
---------- ---------- ----------
      1004 roy
```

Solution 7

```
SQL> select * from emp where ename like 'a%' or ename like 'r%' ;

       EID ENAME                AGE
---------- ---------- -------
      1004 roy
      1005 anand
```

Solution 8

```
SQL> select * from company where city = 'ahmedabad' ;

       EID CNAME                CITY
---------- ---------- ----------
         1 meghmani             ahmedabad
         3 hcc                  ahmedabad
         4 arvind               Ahmedabad
```

Solution 9

```
SQL> update works
```

```
set salary = salary + 500 ;

rows updated.
```

Solution 10

```
SQL> delete from company
  where city='vadodara' ;

1 row deleted.
```

Exercises

FILL IN THE BLANKS:

1. A _____ is a database object that holds user data.
2. Table is created using _____ command.
3. Character data placed within insert into statement must be enclosed in _____ quotes.
4. _____ command is used to filter data on rows retrieved.
5. Foreign key is used to establish relationship between _____ table and _____ table.
6. _____ command is used to change datatype or size of column.
7. _____ command is used to delete table data and structure both.
8. _____ command deletes table data only, table structure remains as it is.
9. _____ command is used to see table structure.
10. When we use wildcard characters & and _____ in condition then _____ operator is used in condition.
11. _____ operator is used to compare column value with list of values.
12. _____ operator is used to compare column value with range of values.
13. _____ command is used to add a column in a table.
14. _____ datatype can be used to store images in table column.
15. Default format of date in oracle is _____ .

Answers:

1 Table
2. Create table
3. ' ' (single quotes)
4. Select
5. Master, detail
6. Alter table...modify
7. Drop table
8. Truncate table
9. Desc
10. Like
11. In
12. Between..And
13. Alter table....add
14. Long raw
15. dd-mon-yy

Multiple Choice Questions:

1. DDL stands for _____
   a. Dynamic Linked Library
   b. Dynamic Language Library
   c. Dynamic Load Library
   d. Data Definition Language

2. A _____ indicates an absent value or unknown value or undefined value
   a. Empty tuple
   b. New value
   c. Null value
   d. None of the above

3. Truncate table will delete _____ .
   a. All rows
   b. All columns
   c. All constraints
   d. All of the above

4. _____ constraint ensures that all values on a column are unique.
   a. Primary key
   b. Foreign key
   c. Check
   d. None of the above

5. Like operator uses _____ and _____ symbols for pattern matching.
   a. * and ?
   b. % and _
   c. $ and @
   d. All of the above

6. _____ command is used to add rows in a table.
   a. Update
   b. Select
   c. Insert
   d. Delete

7. Which command is used to add PRIMARY KEY in existing table?
   a. ALTER TABLE
   b. MODIFY TABLE
   c. CHANGE TABLE
   d. All of the above

8. _____ operator is used to compare Null with Null.
   a. Like
   b. =
   c. Is
   4. In

9. Check constraint checks the value for specified condition during _____ command.

a. Delete
b. Truncate
c. Insert
d. Select

10. DML stands for _____.

a. Data More Language
b. Data Main Language
c. Data Manipulation Language
d. Data Method Language

Answers:

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

Practical Exercise

SET – I

1. Countries

Field name	Datatype	Constraint	description
Country_id	Number(3)	Primary key	
Country_name	Varchar2(50)	Not null	

2. Locations

Field name	Datatype	Constraint	description
Location_id	Number(4)	Primary key	
Street_address	Varchar2(100)		
Postal_code	Varchar2(6)		
City	Varchar2(30)		
State	Varchar2(30)		
Country_id	Number(3)	Foreign key	

3. Department

Field name	Datatype	Constraint	description
Department_id	Number(3)	Primary key	
Department_name	Varchar2(30)		
Manager_id	Number(4)		
Location_id	Number(4)	Foreign key	

4. Jobs

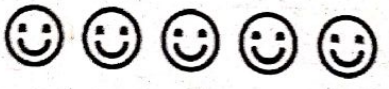
Field name	Datatype	Constraint	description
Job_id	Number(4)	Primary key	
Job_title	Varchar2(30)	Not null	
Min_salary	Number(9,2)		
Max_salary	Number(9,2)	Check – should not be more than 100000	

5. Employees

Field name	Datatype	Constraint	description
Employee_id	Number(4)	Primary key	
First_name	Varchar2(20)		
Email	Varchar2(60)		
Phone_number	Varchar2(30)		
Hire_date	Date		
Job_id	Number(4)	Foreign key	

Queries:

1. Create above tables with necessary constraints.
2. Add last_name column in employees table.
3. Add a constraint in Jobs table Min_salary such that it should Not be less than 7000.
4. Modify country_name in countries table, set its size to 30.
5. Add at least 5 rows in every table.
6. List all employees whose first name starts with 'v'.
7. List all jobs where salary is between 50000 and 70000.
8. List all locations of Ahmedabad city.
9. Update locations table, set city Surat for location_id 3.
10. Delete all countries with country_id more than 3.



CC-212 SQL Practical (UNIT – 1)

❖ **Create table structure with datatype and constraints:**

To create a table in Oracle database, CREATE TABLE command is used with tablename and all other required fields using following syntax.

```
CREATE TABLE tablename (
        Column1 datatype [constraint],
        Column2 datatype [constraint],
        Column n datatype [constraint]);
```

In the following example, we have created a COURSE and a STUDENT table.

Table Name: COURSE:

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> create table course(
  2  c_id integer primary key,
  3  c_name varchar2(5) not null);

Table created.

SQL>
```

In the COURSE table, the field c_id has a datatype integer and a constraint primary key, while the field c_name has datatype varchar2 with the constraint NOT NULL. Once the table is created with proper syntax and fields, the confirmation message "Table Created" will be displayed. Similarly, we can create a STUDENT table,

Table Name: STUDENT

```
SQL Plus

SQL> create table student(
  2  s_id integer primary key,
  3  s_name varchar2(10) not null,
  4  s_gender char(1) check(s_gender in('M','F')),
  5  s_dob date,
  6  s_city varchar2(20) default 'Ahmedabad',
  7  c_id integer,
  8  foreign key(c_id) references course(c_id));

Table created.

SQL>
```

In above example, STUDENT table is created with proper datatype and constra
Also, we have given a foreign key in c_id that refers the primary key of COU
table with the field c_id.

❖ View Description of a Table:

Once the table is created with proper datatype and constraint, we can view numbe
fields with its description like datatype and constraint using DESC command
displays description of a table.

Syntax:

DESC tablename;

Example:

In the following, the description of both the tables COURSE and STUDENT is displayed.

```
SQL Plus

SQL> desc course;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 C_ID                                      NOT NULL NUMBER(38)
 C_NAME                                    NOT NULL VARCHAR2(5)

SQL> desc student;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 S_ID                                      NOT NULL NUMBER(38)
 S_NAME                                    NOT NULL VARCHAR2(10)
 S_GENDER                                           CHAR(1)
 S_DOB                                              DATE
 S_CITY                                             VARCHAR2(20)
 C_ID                                               NUMBER(38)

SQL>
```

❖ Adding Table Rows:

After creating tables, we need to insert data in the table. To insert values in existing table, INSERT command is used.

To insert values in a table, following things must be considered.

- Values are separated by comma.
- Integer values are not enclosed in (‘ ‘)
- Character and date values are placed within the (‘ ‘)

We can insert rows in table using 3 ways.

Syntax:

- INSERT INTO tablename VALUES(value1,value2,value3,….value n);
- INSERT INTO tablename(‘column1,column2,column3...) VALUES(value1,value2,value3...);
- INSERT INTO tablename VALUES(&column1,&column2,&column3...);

In the following example, we have inserted records in a COURSE table using all the methods.

```
SQL Plus
SQL> insert into course(c_id,c_name) values(1,'BCA');

1 row created.

SQL> insert into course values(2,'Bcom');

1 row created.

SQL> insert into course values(&c_id,'&c_name');
Enter value for c_id: 3
Enter value for c_name: BBA
old   1: insert into course values(&c_id,'&c_name')
new   1: insert into course values(3,'BBA')

1 row created.

SQL> /
Enter value for c_id: 4
Enter value for c_name: MCA
old   1: insert into course values(&c_id,'&c_name')
new   1: insert into course values(4,'MCA')

1 row created.
```

Similarly, we can insert records in a STUDENT table,

```
SQL Plus
SQL> insert into student(s_id,s_name,s_gender,s_dob,s_city,c_id) values(1,'Sita','F','15-nov-1988','Sur
at',1);

1 row created.

SQL> insert into student values(2,'Gita','F','12-feb-1990',' ' ,1);

1 row created.

SQL> insert into student values(&s_id,'&s_name','&s_gender','&s_dob','&s-city',&c_id);
Enter value for s_id: 3
Enter value for s_name: Ram
Enter value for s_gender: M
Enter value for s_dob: 3-mar-1990
Enter value for s: Junagadh
Enter value for c_id: 2
old   1: insert into student values(&s_id,'&s_name','&s_gender','&s_dob','&s-city',&c_id)
new   1: insert into student values(3,'Ram','M','3-mar-1990','Junagadh-city',2)

1 row created.

SQL>
```

```
SQL Plus
SQL> /
Enter value for s_id: 4
Enter value for s_name: laxman
Enter value for s_gender: M
Enter value for s_dob: 25-dec-1989
Enter value for s: Ahmedabad
Enter value for c_id: 2
old   1: insert into student values(&s_id,'&s_name','&s_gender','&s_dob','&s-city',&c_id)
new   1: insert into student values(4,'laxman','M','25-dec-1989','Ahmedabad-city',2)

1 row created.

SQL> /
Enter value for s_id: 5
Enter value for s_name: Vrund
Enter value for s_gender: M
Enter value for s_dob: 18-sep-1990
Enter value for s: Surat
Enter value for c_id: 2
old   1: insert into student values(&s_id,'&s_name','&s_gender','&s_dob','&s-city',&c_id)
new   1: insert into student values(5,'Vrund','M','18-sep-1990','Surat-city',2)

1 row created.

SQL>
```


❖ Listing Table Rows:

Once the records are inserted, the contents of a table can be displayed using SELECT command. SELECT command is used with a wildcard character asterisk (*) that is used to display all the records of a table.

Syntax:

SELECT * FROM tablename;

SELECT columnlist FROM tablename;

The columnlist represents number of attributes to be displayed.

```
SQL Plus
SQL> select * from course;

      C_ID C_NAM
---------- -----
         1 BCA
         2 Bcom
         3 BBA
         4 MCA

SQL> select c_name from course;

C_NAM
-----
BCA
Bcom
BBA
MCA

SQL>
```

Similarly, we can display the records of a STUDENT table.

SQL Plus

```
SQL> select * from Student;

      S_ID S_NAME     S S_DOB     S_CITY                                C_ID
---------- ---------- - --------- ------------------------------ ----------
         1 Sita       F 15-NOV-88 Surat                                   1
         2 Gita       F 12-FEB-90 Ahmedabad                               1
         3 Ram        M 03-MAR-90 Junagadh                                2
         5 Vrund      M 18-SEP-90 Surat                                   2

SQL> select s_name, s_gender, c_id from Student;

S_NAME     S       C_ID
---------- - ----------
Sita       F          1
Gita       F          1
Ram        M          2
Vrund      M          2

SQL>
```

❖ Saving Table Changes:

When we close the database, all the table contents are saved on the disk. To save changes in the database table, COMMIT command is used. When power offs suddenly without executing COMMIT command, all the changes will be lost. It permanently saves the changes.

Syntax:

COMMIT;

SQL Plus

```
SQL> commit;

Commit complete.

SQL>
```

❖ Updating Table Rows:

There can a situation where we have to modify the data in the existing table. To modify data, UPDATE command is used with SET keyword and WHERE clause.

Syntax:

UPDATE tablename
SET columnname = value
[WHERE condition];

When the condition is not specified, the data will be updated in all the rows of the specified column.

Example:

We are updating c_id as 3 whose s_id=3.

```
SQL Plus

SQL> select * from STudent;

      S_ID S_NAME     S S_DOB     S_CITY                    C_ID
---------- ---------- - --------- -------------------- ----------
         1 Sita       F 15-NOV-88 Surat                         1
         2 Gita       F 12-FEB-90 Ahmedabad                     1
         3 Ram        M 03-MAR-90 Junagadh                      2
         5 Vrund      M 18-SEP-90 Surat                         2

SQL> update Student
  2  set c_id=3
  3  where s_id=3;

1 row updated.

SQL> select * from Student;

      S_ID S_NAME     S S_DOB     S_CITY                    C_ID
---------- ---------- - --------- -------------------- ----------
         1 Sita       F 15-NOV-88 Surat                         1
         2 Gita       F 12-FEB-90 Ahmedabad                     1
         3 Ram        M 03-MAR-90 Junagadh                      3
         5 Vrund      M 18-SEP-90 Surat                         2

SQL>
```

❖ Restoring Table Content:

If we have not saved the changes after using UPDATE command, then all the changes will be undone using ROLLBACK command. The ROLLBACK command undoes the changes in the contents if COMMIT command is not performed.

Syntax:

ROLLBACK;

```
SQL Plus

SQL> select * from student;

      S_ID S_NAME     S S_DOB     S_CITY                    C_ID
---------- ---------- - --------- -------------------- ----------
         1 Sita       F 15-NOV-88 Surat                         1
         2 Gita       F 12-FEB-90 Ahmedabad                     1
         3 Ram        M 03-MAR-90 Junagadh                      3
         5 Vrund      M 18-SEP-90 Surat                         2

SQL> rollback;

Rollback complete.

SQL> select * from student;

      S_ID S_NAME     S S_DOB     S_CITY                    C_ID
---------- ---------- - --------- -------------------- ----------
         1 Sita       F 15-NOV-88 Surat                         1
         2 Gita       F 12-FEB-90 Ahmedabad                     1
         3 Ram        M 03-MAR-90 Junagadh                      2
         5 Vrund      M 18-SEP-90 Surat                         2

SQL>
```

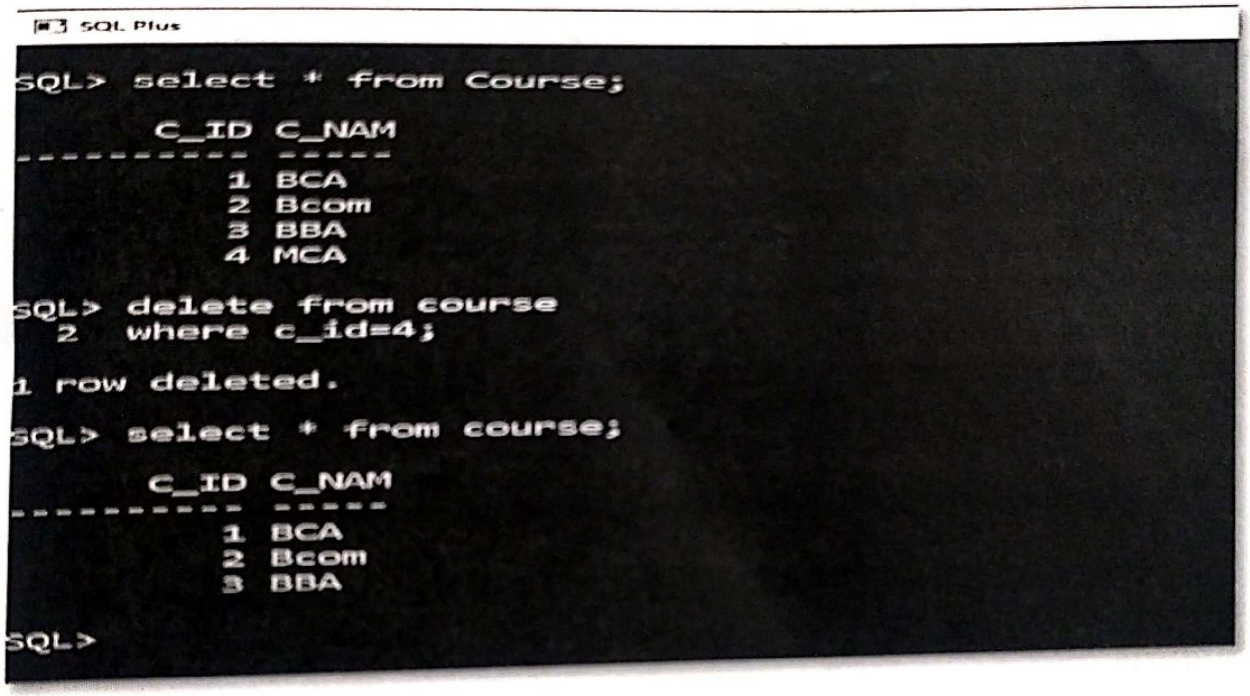
❖ **Deleting table Row:**

The DELETE statement is used to delete the records in a table. It is used to delete a table row.

Syntax:

DELETE FROM tablename
[WHERE condition];

If WHERE condition is not specified then all rows from the specified table will be deleted.



Exercise:

1. **Create following tables.**
   EMPLOYEE(e_id, e_name,e_desig,e_doj,e_city,e_Salary,d_id)
   DEPARTMENT(d_id, d_name)

- **constraints:**
  1) Apply primary and foreign key.
  2) Name should not be null
  3) Designation should be 'Manager','Clerk', and 'Supervisor' only.
  4) The default e_city should be 'Ahmedabad'.
- Insert minimum 8 records in tables.
- **Query:**
  1. Display all records of employee table.
  2. List name and salary of employee who are manager.
  3. Increase salary by 1000 and rename it as Increment.

4. Update designation as "Leader" whose id is '3'.
5. Save all the changes.

2. **Create following tables and perform the given queries.**
PRODUCT(p_id,p_name,p_price,mfgdt,p_qty,p_type,v_id)
VENDOR(v_id,v_name)

- **Constraints:**
  1. Add primary and foreign key.
  2. name should not be null.
  3. p_type should be food, stationery , electronics
  4. p_qty should be greater than 10.
- Insert minimum 10 records in product and 5 records in vendor table.
- **Query:**
  1. Display all records of product and vendor.
  2. Update p_type as 'Stationery_item' whose type is stationery.
  3. Display p_name, p_price, and v_id.
  4. Delete product whose p_id=2.
  5. Update p_price as 500 where p_id=1.
  6. Save the changes.

3. **Create following tables and perform the given queries.**
DOCTOR(did, dname, dept, qualification, salary)
PATIENT(pid, pname, disease, charges, did)

- **Constraints:**
  1. Provide primary foreign key.
  2. Name should not be null.
  3. Default value of charges is 100.
  4. Qualification should be 'MBBS' , 'MS' or 'Ortho'.
- Insert 10 records in both the tables.
- **Queries:**
  1. Display records of doctors and patient.
  2. List dname, qualification and salary of doctors belong
  3. List id, name and charges of a patient.
  4. Update charges=5000 to the Cancer patient.
  5. Save changes in tables.

Unit -2

Transaction Management and Concurrency Control

- TRANSACTION
- CONCURRENCY CONTROL
- Concurrency Control with Locking Methods
- Concurrency Control with Time Stamping Methods
- Concurrency Control with Optimistic Methods
- Database Recovery Management

Unit-2 Transaction Management and Concurrency Control

In this unit we are going to study about:

What is a Transaction and Concurrency Control, Concurrency Control with Locking Methods, Concurrency Control with Stamping Methods, Concurrency Control with Optimistic Methods and Database Recovery Management?

2.1 TRANSACTION:

The real-world database transactions are like transferring amount from one bank account to another, are formed by two or more SQL statements.

The action that reads from and/or writes to a database may consist of:-

- Simple SELECT statements to generate a list of table contents.
- Multiple UPDATE statements to change the values of attributes in various tables.
- Multiple INSERT statements to add rows to one or more tables
- or a combination of SELECT, UPDATE, and INSERT statements

A transaction can be defined as:

A logical unit of work that must be either entirely completed or aborted. The transaction should be completed entirely or if some statements are executed they must be rolled back.

Successful transaction changes the database from one consistent state to another that means all data integrity constraints are satisfied before and after the transaction.

Properties of Transaction:

- Atomicity
  - It requires that all operations (SQL requests) of a transaction be either completed or aborted. No intermediate stages are possible.
- Durability
  - It indicates permanence of database's consistent state
  - The database remains in the consistent state after the transaction completes and all integrity constraints are satisfied.
- Serializability

- It ensures that the concurrent execution of several transactions yields consistent results.

- Isolation
  - Data used during execution of a transaction cannot be used by second transaction until first transaction is completed

2.2 CONCURRENCY CONTROL:

Concurrency control refers to coordination of simultaneous transaction execution in a multiprocessing database system. Its objective is to ensure, transaction Serializability in a multiuser database environment.

The simultaneous execution of transactions occurring over a shared database between multiple users can create various data integrity and consistency problems like:

- lost updates
- uncommitted data
- inconsistent retrievals

Let us understand with an example:

Consider this normal execution of two transactions T1 and T2.
T1 : Purchase 100 units and add to the PROD_QOH
T2: Sell 30 units and subtract from PROD_QOH

TIME	TRASACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH=35+100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	PROD_QOH=135-30	
6	T2	Write PROD_QOH	105

LOST UPDATES:

If the execution does not occur in a serialized way means not having concurrency control the results of the transaction will be inconsistent. Here the transaction T2 reads the PROD_QOH before T1 has written the data to the database, so T2 operates on the value of 35 instead of 135 and T2 overwrites the value of PROD_QOH with 5, so there is a loss of 100 values during the transaction.

TIME	TRASACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	PROD_QOH=35+100	
4	T2	PROD_QOH=35-30	
5	T1	Write PROD_QOH (Lost Update)	135
6	T2	Write PROD_QOH	5

UNCOMMITTED DATA PROBLEM:

Consider this normal execution of two transactions T1 and T2.
T1 : Purchase 100 units and add to the PROD_QOH (Rollback)
T2: Sell 30 units and subtract from PROD_QOH

Here transaction T1 and T2 are concurrent transactions but the first transaction T1 is rolled back before the execution of transaction T2 and thus the addition of 100 units is rolled back. The transaction T2 reads the value of 35 from which 30 units are sold.

TIME	TRASACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH=35+100	
3	T1	Write PROD_QOH	135
4	T1	ROLLBACK	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH=35-30	
7	T2	Write PROD_QOH	5

Uncommitted Data Problem:

Problems of uncommitted data occur when two transactions, T1 and T2, are executed concurrently and the T1 is rolled back after the transaction T2 has already accessed the uncommitted data of transactions. For example, use the same transactions described in lost updates. However, this time the T1 transaction is rolled back to eliminate the addition of the 100 units but T2 reads the value of PROD_QOH before rollback occurs and then subtract 30 from the original 135 units resulting in 105 units while the correct answer should be 5.

TIME	TRASACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH=35+100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH (Read Uncommitted Data)	135
5	T2	PROD_QOH=35-30	
6	T1	ROLLBACK	35
7	T2	Write PROD_QOH	105

Inconsistent Retrievals:

Inconsistent retrievals occur when a transaction calculates some summary (aggregate) functions over a set of data while other transactions are updating the data. The problem is that the transaction might read some data before they are changed and other data after they are changed, thereby yielding inconsistent results.

For example:

1 T1 calculates the total quantity on hand of the products stored in the PRODUCT. table.

2 At the same time, T2 updates the quantity on hand (PROD_QOH) for two of the PRODUCT table's products.

T1 calculates the total quantity on hand (PROD_QOH) for all items, T2 represents the correction of a typing error: the user added 10 units to product P3 PROD_QOH, but meant to add the 10 units to product

P4 PROD_QOH. To correct the problem, the user subtracts 10 from product P4 PROD_QOH and add 10 to product P3 PROD_QOH.

PROD_CODE	PROD_QOH Before	PROD_QOH After
P1	100	100
P2	120	120
P3	70	70+10=80
P4	35	35-10=25
P5	100	100
P6	30	30

TIME	TRASACTION	STEP	VALUE	TOTAL
1	T1	Read PROD_QOH for Prod_Code='P1'	100	100
2	T1	Read PROD_QOH for Prod_Code='P2'	120	220
3	T2	Read PROD_QOH for Prod_Code='P3'	70	
4	T2	**PROD_QOH= 70+10**		
5	T2	Write PROD_QOH for Prod_Code='P3'	80	
6	T1	Read PROD_QOH for Prod_Code='P3'	80	300
7	T1	Read PROD_QOH for Prod_Code='P4'	35	335
8	T2	Read PROD_QOH for Prod_Code='P4'	35	
9	T2	**PROD_QOH=35-10**		
10	T2	Write PROD_QOH for Prod_Code='P4'	25	
11	T2	**COMMIT**		
12	T1	Read PROD_QOH for Prod_Code='P5'	100	435
13	T1	Read PROD_QOH for Prod_Code='P6'	30	465

The total of 465 is wrong and the correct answer is 455, such problems of concurrency control can result in inconsistency in information retrieved from the database.

2.3 Concurrency Control with Locking Methods:

Lock: It guarantees exclusive use of a data item to a current transaction It is required to prevent another transaction from reading inconsistent data

Lock manager: It is responsible for assigning and policing the locks used by the transactions

Lock Granularity: It indicates the level of lock use. Locking can take place at the following levels: Database, Table, Page, Row, Field (attribute)

- **Database-level lock**: Entire database is locked by the transaction. For example: If transaction T1 has locked the database, then transaction T2 will wait until T1 unlocks the database. T2 cannot access any other tables within the database. Thus two transactions cannot access the same database at the same time. This system remains slow as all other transactions would be waiting until execution of the current transaction completes.

- **Table-level lock:** Only entire table is locked which is used rest of the database can be locked by other transactions.
  For example: Transaction T1 has locked the table product, the concurrent transaction T2 can lock another table customer.
  If T2 wants to lock table product, it has to wait until T1 unlocks the table product. This type of locks are beneficial in multiuser environment.

- **Page-level lock**: Entire disk page is locked rather than entire database or entire table. A page is similar to a disk block of a fixed size such as 4K, 8K etc.. One table can consist of several pages and a page can have several rows and columns. If a page is locked by T1, it cannot be accessed by T2 until T1 releases the lock.

- **Row-level lock:** Allows concurrent transactions to access different rows of the same table, even if the rows are located on the same page. It increases the overhead on dbms.

- **Field-level lock:** It locks only a single attribute rather than entire table. It allows concurrent transactions to access the same row, as long as they require the use of different fields (attributes) within that row. It increases high level of overhead for processing.

Types of Locks:

Binary lock: It has only two states: locked (1) or unlocked (0)
If any database object like page, table, row or field are locked by a transaction, no other transaction can use the object until it is unlocked. After it is unlocked other transaction can lock it for its purpose. Thus every transaction requires a lock and unlock mechanism for successful execution.

TIME	TRASACTION	STEP	STORED VALUE
1	**T1**	**Lock Product**	
2	T1	Read PROD_QOH	35
3	T1	PROD_QOH=35+100	
4	T1	Write PROD_QOH	135
5	**T1**	**Unlock Product**	
6	**T2**	**Lock Product**	
7	T2	Read PROD_QOH	135
8	T2	PROD_QOH=135-30	
9	T2	Write PROD_QOH	105
10	**T2**	**Unlock Product**	

In the above example transaction T1 locks the table product and then operates on the value of PROD_QOH and after the write operation, the transaction completes and releases the lock.

After transaction T1 unlocks the table product, T2 can acquire the lock on T1, updates the value of PROD_QOH and then after completion of transaction, releases the lock.

Shared and Exclusive lock

- Exclusive Lock

  It exists when access is specifically reserved for the transaction that locked the object. It is used when the transactions are updating or inserting new values to the database.

- Shared lock

  It exists when Concurrent transactions are granted Read access on the basis of a common lock.

Two Phase Locking Protocol. (2PL)

Two Phase locking guarantees Serializability. Two-phase locking defines how transactions acquire and relinquish locks but it does not prevent deadlocks.

The two phases are:

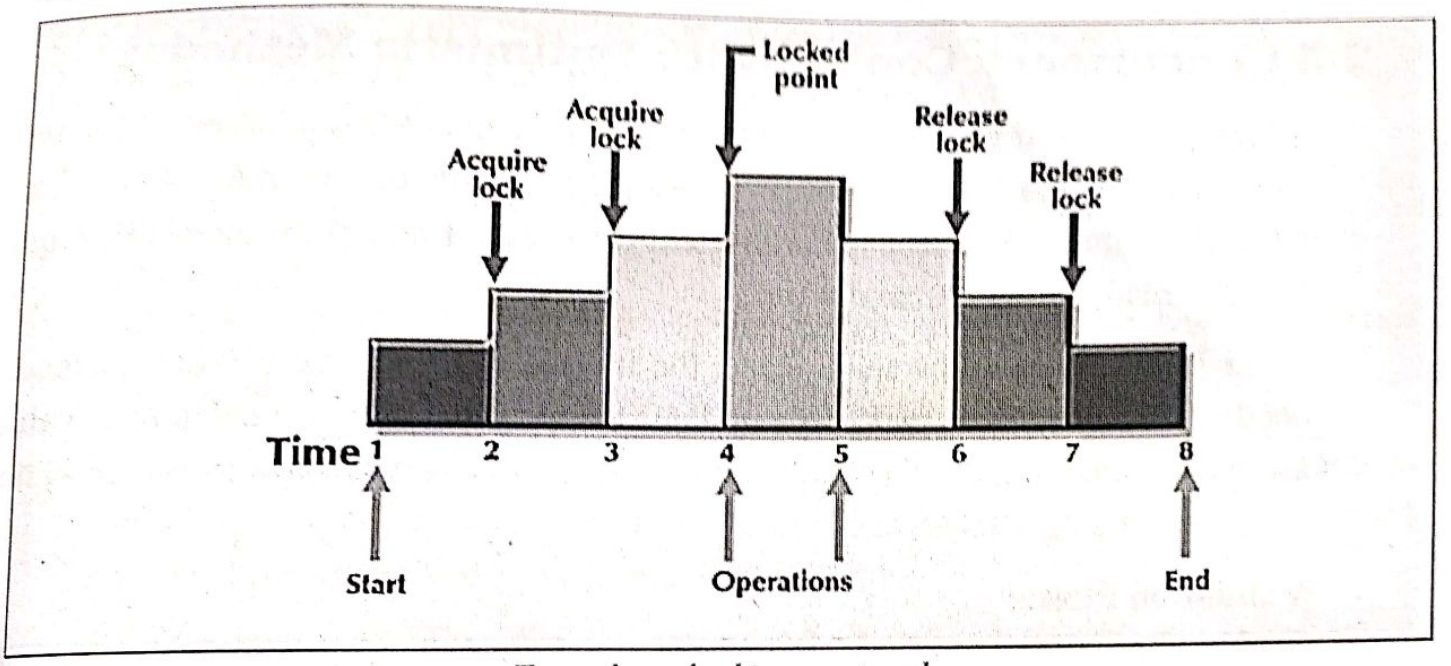
Growing Phase: A growing phase, in which a transaction acquires all the required locks without unlocking any data.

Lock Point: When all locks are acquired, the transaction is in its locked point.

Shrinking Phase: A shrinking phase, in which a transaction releases all locks and cannot acquire any new lock.

The rules of two-phase locking protocol are:

- No two transactions should have conflicting locks.
- No unlock operation can precede a lock operation in the same transaction,
- No data are modified until all locks are obtained that is, until the transaction reaches its locked point



Two-phase locking protocol.

The transaction obtains two locks it required and it reaches its locked point and when the locked point is reached, the data is modified to according to the transaction requirements. Then, the transaction is finished, it releases all of the locks it acquired.

4 Concurrency Control with Time Stamping Methods:

This method assigns a unique time stamp for scheduling all concurrent transactions. The time stamp value generates an order in which transactions are submitted to the database management system. All database operations within the same transaction must have the same time stamp. There are two basic properties of Time stamps: uniqueness and monotonicity.

- Uniqueness make sure that no equal stamp values can exist.
- Monotonicity indicates that time stamp values will always increase.

This properties are used to ensure the transactions execute in a serial order. The main disadvantage of the time stamping approach is that each value stored in the database requires two additional time stamp fields: one for the last time the field was read, and one for the last update and thus it increases the memory requirements and the increases database's processing overhead.

Concurrency Control using time stamping method includes two schemes:

- Wait/die
  - Older transaction waits and the younger is rolled back and rescheduled
- Wound/wait
  - Older transaction rolls back the younger transaction and reschedules it

2.5 Concurrency Control with Optimistic Methods:

In this approach, it is assumed that the most of the database operations do not conflic so it does not require locking or time stamping techniques and transaction is executed without limitations until it is committed. Here each transaction moves through three phases - read, validation, and write.

Read Phase: During the read phase, the transaction reads the database, executes the needed statements, and makes the updates to a private copy of the database values and all update operations of the transaction are done in a temporary file, which is not read by the remaining transactions

Validation Phase:
During the validation phase, the transaction is validated to check that the changes made will not affect the integrity and consistency of the database. If the validation phase result is positive, the transaction goes to the write phase. If the validation phase result is negative, the transaction is started again, and the changes are rolled back.

Write Phase:
During the write phase, the changes are permanently applied to the database.

This approach is applied for database systems that require very less update transactions.

2.6 Database Recovery Management:

In a heavily used DBMS environment, the management of database problems like deadlocks-their prevention and detection constitutes an important DBMS function. It is necessary to employ database recovery techniques to restore the database to a consistent state in case of any failure or deadlock situations.

Database Recovery restores database from an inconsistent, to a previously consistent state. Recovery techniques are based on the atomic transaction property
All portions of the transaction must be treated as a single logical unit of work, in which all operations must be applied and completed to produce a consistent database

If transaction operation cannot be completed, transaction must be aborted, and any changes to the database must be rolled back.

DBMS provides backup facilities for recovery of the database.

The types of backups are:

1. Full Backup: The copy of entire database is taken as a backup.
2. Differential Backup: Instead of copying the entire database again and again, only the last modifications done to the database are considered for backup.
3. Transaction Log Backup: The copy of transaction log file is considered for backup. This backup also takes into considerations of last modifications done to the database.

   These backups are kept at a secure place as they are required in case of database failure to restore the database.

These failures of the database can be due to hardware failures, software failures including operating system or dbms failure, programming errors or exceptions and deadlocks during the execution of transactions.

Transaction Recovery is carried out using deferred-write and write-through

- Deferred write: Transaction operations do not immediately update the physical database but only the transaction log is updated. The Database is physically updated only after the transaction reaches its commit point using the transaction log information
- Write-through : Database is immediately updated by transaction operations during the transaction's execution, even before the transaction reaches its commit point.

☺☺☺☺☺

Exercises

1. What is a Transaction and explain its properties.
2. What is Concurrency Control? Explain problems occurring due to lack of concurrency control.
3. Explain Concurrency Control with Locking Methods
4. Explain Concurrency Control with Stamping Methods
5. Explain Concurrency Control with Optimistic Methods
6. Explain Database Recovery Management.

1. Create table Employee_Details with data given below:

Employee_details: (emp_id, categoryid, empname, empcontactno, empemailid, salary)

Attempt the following Queries:

(i.) Add a field designation to employee_details
(ii.) Display empid, empname, empcontactno and salary of all employees where designation is 'manager'.
(iii.) display all employee details whose empname starts with 'a'
(iv.) Add primary key to empid

2. Create table Book_Category with data given below:

CategoryCode	CategoryName

Create table Book_Details with data given below:

BookCode	CategoryCode	BookName	Author	Publisher	Price	PublishedDate

[BookName should be varchar(30)]

Attempt the following Queries:

(i.) Add two more fields Discount and Remarks to the table Book_Details.
(ii.) Display Bookcode, BookName, Author, Publisher and Price of all books having price more than 150.
(iii.) Display all the book details whose author name starts with 'b'
(iv.) Change the datatype size of BookName to varchar(50)

Multipal Choice Que.:

1. _____ is a not a type of lock.
   a) Shared
   b) exclusive
   c) binary
   d) digital

2. _____ is not a problem occurred due to lack of concurrency control.
   a) Lost update
   b) Lost commit
   c) Uncommited data
   d) Inconsistent retrieval

3. _____ means transaction should either completed successfully or rolled back.
   a) Atomicity
   b) Durability
   c) Isolation
   d) Consistency

4. _____is known as a logical unit of work that must be either entirely completed or aborted.
   a) Transaction
   b) Process
   c) Concurrency control
   d) Deadlock

5. _____ refers to coordination of simultaneous transaction execution in a multiprocessing database system.
   a) Transaction
   b) Process
   c) Concurrency control
   d) Deadlock

6. If the execution does not occur in a serialized way means not having _____
   a) Transaction
   b) Process
   c) Concurrency control

d) Deadlock

7. _____ guarantees exclusive use of a data item to a current transaction.
   a) Lock
   b) Transaction
   c) Isolation
   d) deadlock

8. _____ is responsible for assigning and policing the locks used by the transactions.
   a) Transaction manager
   b) Lock Manager
   c) Transaction processor
   d) Data processor

9. A _____ is similar to a disk block.
   a) Database
   b) Dataset
   c) Page
   d) Table

10. _____ lock has only two states: locked (1) or unlocked (0)
   a) Digital
   b) Binary
   c) Shared
   d) Exclusive

11. Two Phase locking guarantees _____.
   a) Serializability
   b) Isolation
   c) Durability
   d) Stability

12. _____indicates that time stamp values will always increase.
   a) Monotonicity
   b) Scheduling
   c) Uniqueness
   d) Seriablizability

13. When all locks are acquired, the transaction reaches its _____
   a) Final point

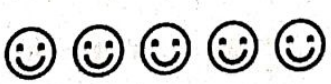
b) Lock point
c) Test point
d) Release point

14. Recovery techniques are based on the _____ transaction Property.
a) Atomicity
b) Durability
c) Isolation
d) Consistency

15. _____ Recovery is carried out using deferred-write and write-through.
a) Database
b) Transaction
c) Table
d) Row

Answers:

1-d	2-b	3-a	4-a	5-c
6-c	7-a	8-b	9-c	10-b
11-a	12-a	13-b	14-a	15-b



CC-212 SQL Practical (UNIT – 2)

Perform select queries on different tables.

- Writing an SQL SELECT query
- It has basic three keywords
  - SELECT
  - FROM
  - WHERE

select * from table_name;

Example:

Select and list all rows and columns from a table
SQL> **select * from checks;**

OUTPUT:

CHECKNO	PAYEE	AMOUNT	REMARKS
1	TTC	150	Textile Traders
2	Reading R.R.	245.34	Reader A/c
3	Bell	200.32	Cellular Phone Co.
4	Local Utilities	98	Gas
5	Joes Stale $ Dent	150	Groceries
6	Cash	25	Wild Night Out
7	James Gas	25.1	Gas

⇒ **7 rows selected.**

Example:

Select and list selected columns from a table
SQL> **SELECT payee, remarks, amount, checkno from checks;**

SQL> **SELECT CHECKNO, amount from checks;**

SQL> **select amount from checks;**

AMOUNT
150
245.34

SQL> **select DISTINCT amount from checks;**

AMOUNT
25
25.1

200.32	98
98	150
150	200.32
25	245.34
25.1	
7 rows selected.	6 rows selected.

SQL> **SELECT * FROM BIKES;**

NAME	FRAMESIZE	COMPOSITION	MILESRIDDEN	TYPE
TREK 2300	22.5	CARBON FIBER	3500	RACING
BURLEY	22	STEEL	2000	TANDEM
GIANT	19	STEEL	1500	COMMUTER
FUJI	20	STEEL	500	TOURING
SPECIALIZED	16	STEEL	100	MOUNTAIN
CANNONDALE	22.5	ALUMINUM	3000	RACING

⇒ **rows selected.**

SQL> **SELECT * FROM BIKES WHERE NAME = 'BURLEY';**

NAME	FRAMESIZE	COMPOSITION	MILESRIDDEN	TYPE
BURLEY	22	STEEL	2000	TANDEM

⇒ **1 rows selected.**

Use of Arithmetic Operators

SQL> **SELECT * FROM PRICE;**

OUTPUT:

ITEM	WHOLESALE
TOMATOES	.34
POTATOES	.51
BANANAS	.67
TURNIPS	.45
CHEESE	.89
APPLES	.23

⇒ **rows selected.**

SQL> **SELECT ITEM, WHOLESALE, WHOLESALE + 0.15 FROM PRICE;**

Here the + adds 15 cents to each price to produce the following:

ITEM	WHOLESALE	WHOLESALE+0.15
TOMATOES	.34	.49
POTATOES	.51	.66
BANANAS	.67	.82
TURNIPS	.45	.60
CHEESE	.89	1.04
APPLES	.23	.38

⇒ **rows selected.**

You can use +, -, *, / and % operators with the same query

SQL> **SELECT * FROM FRIENDS;**

OUTPUT:

LASTNAME	FIRSTNAME	AREACODE	PHONE	ST	ZIP
BUNDY	AL	100	555-1111	IL	22333
MEZA	AL	200	555-2222	UK	
MERRICK	BUD	300	555-6666	CO	80212
MAST	JD	381	555-6767	LA	23456
BULHER	FERRIS	345	555-3223	IL	23332

SQL> **SELECT * FROM FRIENDS WHERE FIRSTNAME = 'JD';**

LASTNAME	FIRSTNAME	AREACODE	PHONE	ST	ZIP
MAST	JD	381	555-6767	LA	23456

Greater Than (>) and Greater Than or Equal To (>=)

SQL> **SELECT * FROM FRIENDS WHERE AREACODE > 300;**

LASTNAME	FIRSTNAME	AREACODE	PHONE	ST	ZIP
MAST	JD	381	555-6767	LA	23456
BULHER	FERRIS	345	555-3223	IL	23332

SQL> **SELECT * FROM FRIENDS WHERE AREACODE >= 300;**

```
LASTNAME     FIRSTNAME     AREACODE PHONE    ST ZIP
------------ ------------ --------- -------- -- -----
MERRICK      BUD             300 555-6666 CO 80212
MAST         JD              381 555-6767 LA 23456
BULHER       FERRIS          345 555-3223 IL 23332
```

Less Than (<) and Less Than or Equal To (<=)

SQL> SELECT * FROM FRIENDS WHERE STATE < 'LA';

OUTPUT:

```
LASTNAME     FIRSTNAME     AREACODE PHONE    ST ZIP
------------ ------------ --------- -------- -- -----
BUNDY        AL              100 555-1111 IL 22333
MERRICK      BUD             300 555-6666 CO 80212
BULHER       FERRIS          345 555-3223 IL 23332
```

SQL> SELECT * FROM FRIENDS WHERE FIRSTNAME <> 'AL';

SQL> SELECT * FROM FRIENDS WHERE STATE != 'CA';

SQL> SELECT * FROM PARTS;

```
NAME         LOCATION     PARTNUMBER
------------ ------------ ----------
APPENDIX     MID-STOMACH           1
ADAMS APPLE  THROAT                2
HEART        CHEST                 3
SPINE        BACK                  4
ANVIL        EAR                   5
KIDNEY       MID-BACK              6
```

SQL> SELECT * FROM PARTS WHERE LOCATION LIKE '%BACK%';

```
NAME         LOCATION     PARTNUMBER
------------ ------------ ----------
SPINE        BACK                  4
KIDNEY       MID-BACK              6
```

SQL> SELECT * FROM PARTS WHERE LOCATION LIKE 'BACK%';

```
NAME         LOCATION     PARTNUMBER
------------ ------------ ----------
```

```
SPINE       BACK              4

SQL> SELECT * FROM PARTS WHERE NAME LIKE 'A%';

NAME        LOCATION      PARTNUMBER
----------- ------------- ----------
APPENDIX    MID-STOMACH       1
ADAMS APPLE THROAT            2
ANVIL       EAR               5

SQL> SELECT * FROM PARTS WHERE NAME LIKE 'a%';

no rows selected
```

Underscore (_)

```
SQL> SELECT * FROM FRIENDS;

LASTNAME    FIRSTNAME    AREACODE PHONE    ST ZIP
----------- ---------- ------- ------- -- -----
BUNDY       AL           100 555-1111 IL 22333
MEZA        AL           200 555-2222 UK
MERRICK     UD           300 555-6666 CO 80212
MAST        JD           381 555-6767 LA 23456
BULHER      FERRIS       345 555-3223 IL 23332
PERKINS     ALTON        911 555-3116 CA 95633
BOSS        SIR          204 555-2345 CT 95633

SQL> SELECT * FROM FRIENDS WHERE STATE LIKE 'C_';

LASTNAME    FIRSTNAME    AREACODE PHONE    ST ZIP
----------- ---------- ------- ------- -- -----
MERRICK     BUD          300 555-6666 CO 80212
PERKINS     ALTON        911 555-3116 CA 95633
BOSS        SIR          204 555-2345 CT 95633

SQL> SELECT * FROM FRIENDS WHERE PHONE LIKE'555-6_6_';

LASTNAME    FIRSTNAME    AREACODE PHONE    ST ZIP
----------- ---------- ------- ------- -- -----
MERRICK     BUD          300 555-6666 CO 80212
MAST        JD           381 555-6767 LA 23456

SQL> SELECT * FROM FRIENDS WHERE PHONE LIKE '555-6%';
```

```
LASTNAME     FIRSTNAME     AREACODE PHONE    ST ZIP
------------ ------------ --------- -------- -- -----
MERRICK      BUD             300 555-6666 CO 80212
MAST         JD              381 555-6767 LA 23456
```

SQL> **SELECT * FROM FRIENDS WHERE FIRSTNAME LIKE '_L%';**

```
LASTNAME     FIRSTNAME     AREACODE PHONE    ST ZIP
------------ ------------ --------- -------- -- -----
BUNDY        AL              100 555-1111 IL 22333
MEZA         AL              200 555-2222 UK
PERKINS      ALTON           911 555-3116 CA 95633
```

SQL> **SELECT LASTNAME || ',' || FIRSTNAME NAME FROM FRIENDS;**

```
NAME
--------------------------------------------------
BUNDY    , AL
MEZA     , AL
MERRICK  , BUD
MAST     , JD
BULHER   , FERRIS
PERKINS  , ALTON
BOSS     , SIR
7 rows selected.
```

Logical Operators

SQL> **SELECT * FROM VACATION;**

OUTPUT:

```
LASTNAME     EMPLOYEENUM   YEARS  LEAVETAKEN
------------ ----------- --------- ---------
ABLE             101        2         4
BAKER             104        5        23
BLEDSOE           107        8        45
BOLIVAR           233        4        80
BOLD             210        15       100
COSTALES          211        10       78
6 rows selected.
```

Company gives each employee 12 days of leave each year. Find all the employees whose names start with B and who have more than 50 days of leave coming.

SQL> **SELECT LASTNAME, YEARS * 12 - LEAVETAKEN REMAINING FROM VACATION WHERE LASTNAME LIKE 'B%' AND YEARS * 12 - LEAVETAKEN > 50;**

```
LASTNAME     REMAINING
------------ ---------
BLEDSOE           51
BOLD              80
```

AND

SQL> **SELECT LASTNAME FROM VACATION WHERE YEARS <= 5 AND LEAVETAKEN > 20;**

If you want to know which employees have been with the company for 5 years or more and have taken less than 50 percent of their leave

SQL> **SELECT LASTNAME WORKAHOLICS FROM VACATION WHERE YEARS >= 54 AND ((YEARS *12)-LEAVETAKEN)/(YEARS * 12) < 0.50;**

OR

If any of the comparisons is true, OR returns TRUE.

SQL> **SELECT LASTNAME WORKAHOLICS FROM VACATION WHERE YEARS >= 5 OR ((YEARS *12)-LEAVETAKEN)/(YEARS * 12) >= 0.50;**

OUTPUT:

```
WORKAHOLICS
--------------
ABLE
BAKER
BLEDSOE
BOLD
COSTALES
```

NOT

SQL> **SELECT * FROM VACATION WHERE LASTNAME NOT LIKE 'B%'**

OUTPUT:

```
LASTNAME     EMPLOYEENUM   YEARS LEAVETAKEN
```

```
---------- ---------- -------- ----------
ABLE          101       2        4
COSTALES      211       10       78
```

Changing a Column's Data Type

The datatype of existing columns of the table can be modified as:

```
ALTER TABLE TABLE_NAME
MODIFY (COLUMN_NAME DATATYPE);
```

Example:

The previous data type would be changed as:

```
SQL> ALTER TABLE CUSTOMERS
  MODIFY (CUSTOMER_NAME VARCHAR (30));
```

Changing a Column's Data Characteristic

The size of datatype of existing columns of the table can be modified as:

```
ALTER TABLE TABLE_NAME
MODIFY (COLUMN_NAME DATATYPE (SIZE));
```

Example:

The size of the column customer_name was 30 which can be update to 50.

```
SQL> ALTER TABLE CUSTOMERS
  MODIFY (CUSTOMER_NAME VARCHAR (50));
```

Adding a column

The new column can be added to the table as:

```
ALTER TABLE TABLE_NAME
ADD (COLUMN_NAME DATATYPE (SIZE));
```

Example:

```
SQL> ALTER TABLE CUSTOMERS
  ADD (CUSTOMER_EMAILID VARCHAR (50));
```

Dropping a column

The new column can be added to the table as:

```
SQL> ALTER TABLE TABLE_NAME
  DROP COLUMN COLUMN_NAME;
```

Example:

```
SQL> ALTER TABLE CUSTOMERS
  DROP COLUMN CUSTOMER_EMAILID;
```

Copying Parts of Table

The data from existing table can be used while creating a new table. Here the data type of the columns should be same.

```
INSERT INTO TABLE_NAME1 (COLUMN1,COLUMN2) AS SELECT
COLUMN1, COLUMN2 FROM TABLE_NAME2;
```

Example:

```
SQL> INSERT INTO PRODUCT_LIST (PRODUCT_ID,PRODUCT_NAME)
AS SELECT PRODUCT_ID, PRODUCT_NAME FROM
PRODUCT_MASTER;
```

Adding Primary and Foreign Key Designations

The new constraint of primary key can be added to the existing table as:

```
  ALTER TABLE TABLE_NAME
  ADD PRIMARY KEY (COLUMN_NAME);
SQL> ALTER TABLE CUSTOMERS
  ADD PRIMARY KEY (CUSTOMER_ID);
```

Composite primary key can be added as:

```
SQL> ALTER TABLE STUDENT
  ADD PRIMARY KEY (SEMESTER_ID, ROLL_NO);
```

Deleting Table from the Database

The existing table can be completely removed from the database including the structure using:

```
DROP TABLE TABLE_NAME;

SQL> DROP TABLE CUSTOMERS;
```

Unit -3

Distributed Database Management System

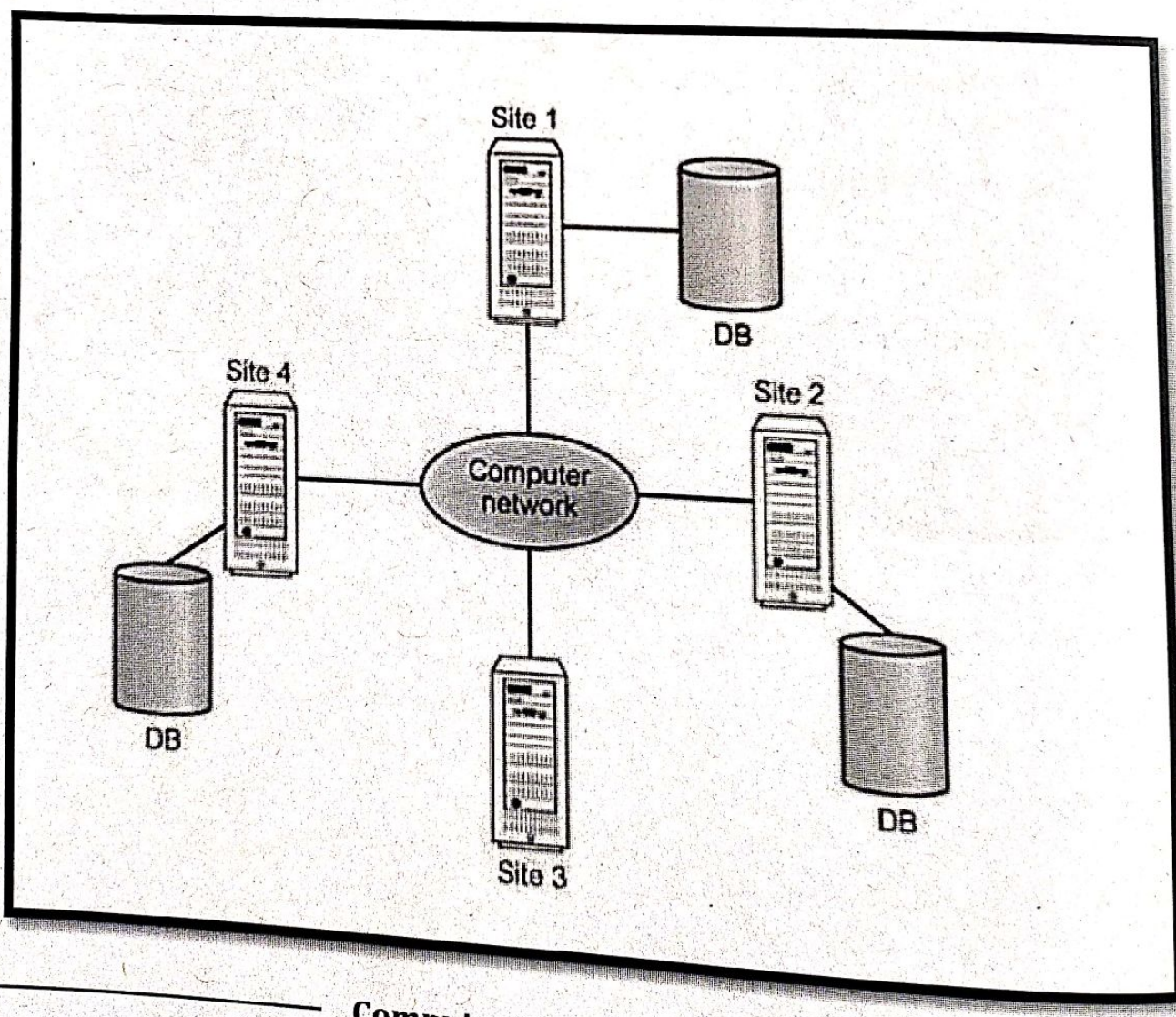
- Evolution of DDBMS
- Distributed Processing and Distributed Database
- Levels of Data and Process Distribution
- Distributed Database Transparency Features
- Distributed Transparency
- Transaction Transparency
- Performance Transparency and Query Optimization

Unit-3 Distributed Database Management System

3.1 Evolution of DDBMS:

A distributed database management system (DDBMS) governs the storage and processing of logically related data over interconnected computer systems in which both data and processing are distributed among several sites.

The use of a centralized database required that corporate data be stored in a single central site, usually a mainframe computer. Data access was provided through dumb terminals. The centralized approach worked well to fill the structured information needs of corporations, but it fell short when quickly moving events required faster response times and equally quick access to information. The slow progression from information request to approval to specialist to user simply did not serve decision makers well in a dynamic environment.



The factors influenced the evolution of the DDBMS:

The different factors influenced the evolution of the DDBMS are as follows.

- The growing acceptance of the Internet as the platform for data access and distribution which leads to maintain the repository for distributed data.
- The increased focus on data analysis that led to data mining and data warehousing. Although a data warehouse is not usually a distributed database, it does rely on techniques such as data replication and distributed queries that facilitate data extraction and integration.

The Problems with the Centralized Database Management System:

- Performance degradation because of a growing number of remote locations over greater distances.
- High costs associated with maintaining and operating large central (mainframe) database systems.
- Reliability problems created by dependence on a central site (single point of failure syndrome) and the need for data replication.
- Scalability problems associated with the physical limits imposed by a single location (power, temperature conditioning, and power consumption.)
- Organizational rigidity imposed by the database might not support the flexibility and agility required by modern global organizations.

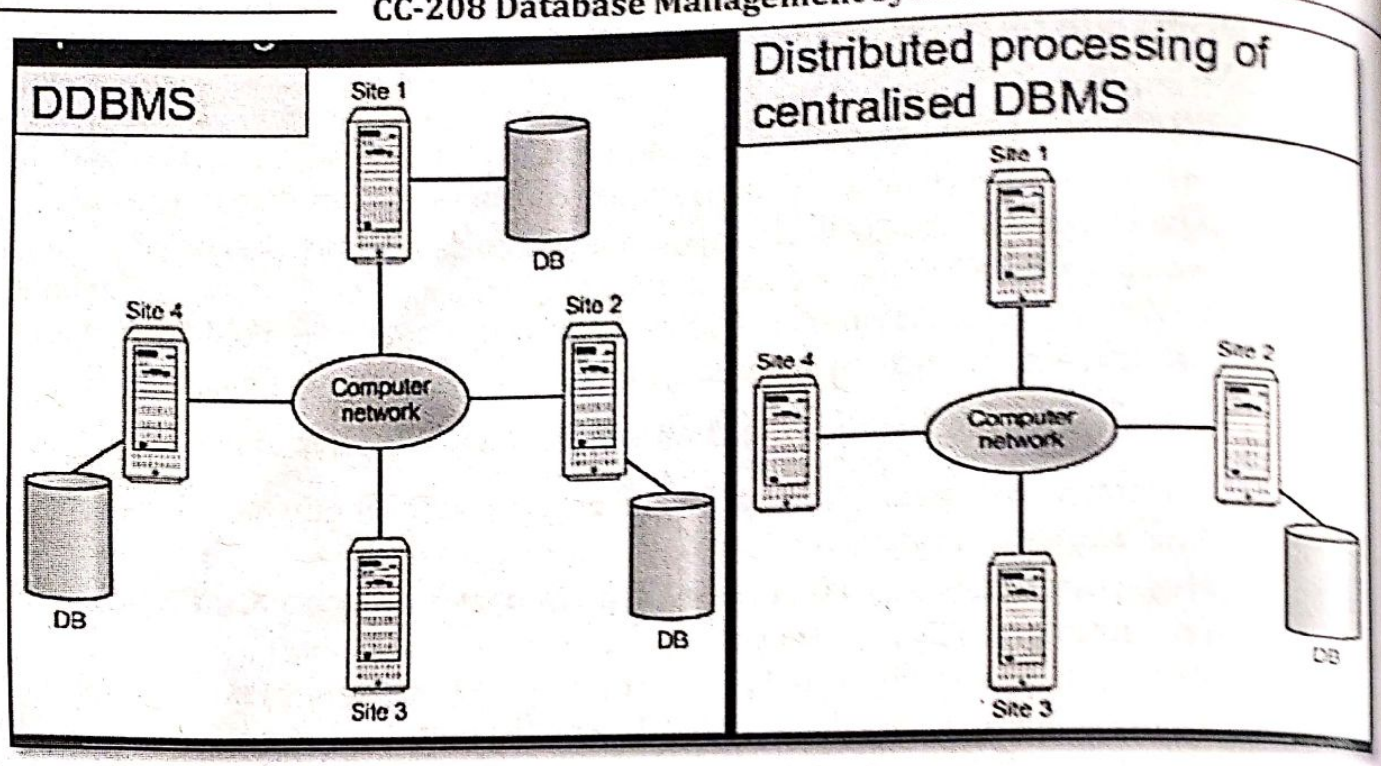
3.2 Distributed Processing and Distributed Database:

Distributed Processing:

In distributed processing, a database's logical processing is shared among two or more physically independent sites that are connected through a network. For example, the data input/output (I/O), data selection, and data validation might be performed on one computer, and a report based on that data might be created on another computer.

Distributed database:

A distributed database, on the other hand, stores a logically related database over two or more physically independent sites. The sites are connected via a computer network. In contrast, the distributed processing system uses only a single-site database but shares the processing chores among several sites. In a distributed database system, a database is composed of several parts known as database fragments. The database fragments are located at different sites and can be replicated among various sites. Each database fragment is, in turn, managed by its local database.



Distributed processing does not require a distributed database, but a distributed database requires distributed processing (each database fragment is managed by its own local database process).

Distributed processing may be based on a single database located on a single computer. For the management of distributed data to occur, copies or parts of the database processing functions must be distributed to all data storage sites.

Both distributed processing and distributed databases require a network to connect all components.

Difference:

Distributed Processing	Distributed Database
In distributed processing, a database's logical processing is shared among two or more physically independent sites that are connected through a network.	A distributed database, on the other hand, stores a logically related database over two or more physically independent sites.
Distributed processing does not require a distributed database	a distributed database requires distributed processing
Distributed processing may be based on a single database located on a single computer.	Distributed database may be based on a single database located on multiple computers.

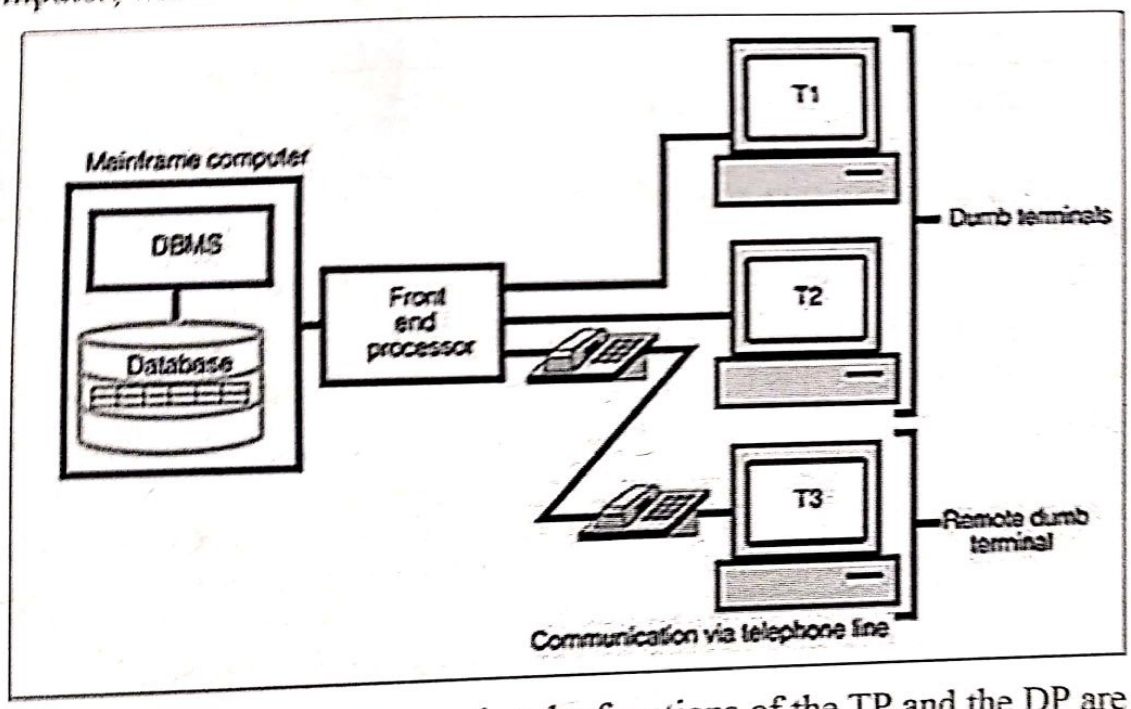
3.3 Levels of Data and Process Distribution:

Current database systems can be classified on the basis of how process distribution and data distribution are supported.

For example, a DBMS may store data in a single site (centralized DB) or in multiple sites (distributed DB) and may support data processing at a single site or at multiple sites. The different types of Data and Process distribution methods are as follows.

3.3.1 Single-Site Processing, Single-Site Data(SPSD):

In the single-site processing, single-site data (SPSD) scenario, all processing is done on a single host computer (single-processor server, multiprocessor server, mainframe system) and all data are stored on the host computer's local disk system. Processing cannot be done on the end user's side of the system. Such a scenario is typical of most mainframe and midrange server computer DBMSs. The DBMS is located on the host computer, which is accessed by dumb terminals connected to it.

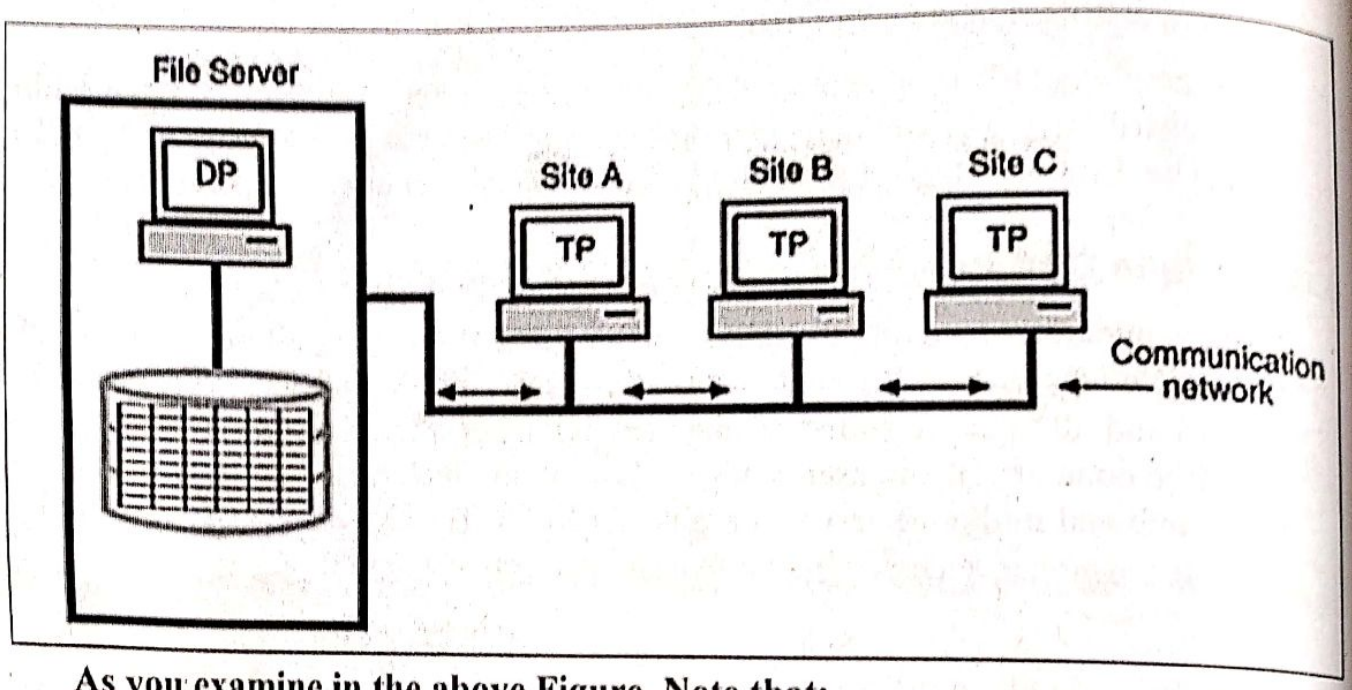


In the above figure you can see that the functions of the TP and the DP are embedded within the DBMS located on a single computer. The DBMS usually runs under a time-sharing, multitasking operating system, which allows several processes to run concurrently on a host computer accessing a single DP. All data storage and data processing are handled by a single host computer.

3.3.2 Multiple-Site Processing, Single-Site Data(MPSD):

Under the multiple-site processing, single-site data (MPSD) scenario, multiple processes run on different computers sharing a single data repository. Typically, the MPSD scenario requires a network file server running conventional applications that

are accessed through a network. Many multiuser accounting applications running under a personal computer network fit such a description. Consider the following figure.



As you examine in the above Figure, Note that:

- The TP on each workstation acts only as a redirector to route all network data requests to the file server.
- The end user sees the file server as just another hard disk. Because only the data storage input/output (I/O) is handled by the file server's computer, the MPSD offers limited capabilities for distributed processing.
- The end user must make a direct reference to the file server in order to access remote data. All record- and file-locking activities are done at the end-user location.
- All data selection, search, and update functions take place at the workstation, thus requiring that entire files travel through the network for processing at the workstation. Such a requirement increases network traffic, slows response time, and increases communication costs.

3.3.3 Multiple-Site Processing, Multiple-Site Data (MPSD):

The multiple-site processing, multiple-site data (MPMD) scenario describes a fully distributed DBMS with support for multiple data processors and transaction processors at multiple sites. Depending on the level of support for various types of centralized DBMSs, DDBMSs are classified as either homogeneous or heterogeneous.

Homogeneous DDBMSs integrate only one type of centralized DBMS over a network. Thus, the same DBMS will be running on different server platforms (single processor server, multiprocessor server, server farms, or server blades). In contrast, heterogeneous DDBMSs integrate different types of centralized DBMSs over a network. A fully heterogeneous DDBMS will support different DBMSs that may even support different data models (relational, hierarchical, or network) running under different computer systems, such as mainframes and PCs.

3.4 Distributed Database Transparency Features:

DDBMS transparency features have the common property of allowing the end user to feel like the database's only user. In other words, the user believes that (s) he is working with centralized DBMS; all complexities of a distributed database are hidden, or transparent to the user.

The DDBMS transparency features are:

- **Distribution transparency:** It allows a distributed database to be treated as a single logical database. If a DDBMS exhibits distribution transparency , the user does not need to know:
  - That the data are partitioned _ meaning the table's rows and columns are split Vertically or horizontally and stored among multiple sites.
  - That the data can be replicated at several sites.
  - The data location.
- **Transaction transparency**: It allows a transaction to update data at more than one network site Transaction transparency ensures that the transaction will be either entirely completed or aborted. Thus, maintaining database integrity.
- **Failure transparency**: It ensures that the system will continue to operate in the event of a node failure. Functions that were lost because of the failure will be picked up by another network node.
- **Performance transparency:** It allows the system to perform as if it were a centralized DBMS. The system will not suffer any performance degradation due to its use on a network or due to the network's platform differences. Performance transparency also ensures that the system will find the most cost-effective path to access remote data.
- **Heterogeneity transparency.** Which allows the integration of several? Different local DBMSs (relational. network, and hierarchical) under a Common, or global, schema. The DDBMS is responsible for translating The data requests from the global schema to the local DBMS schema.

3.5 Distributed Transparency:

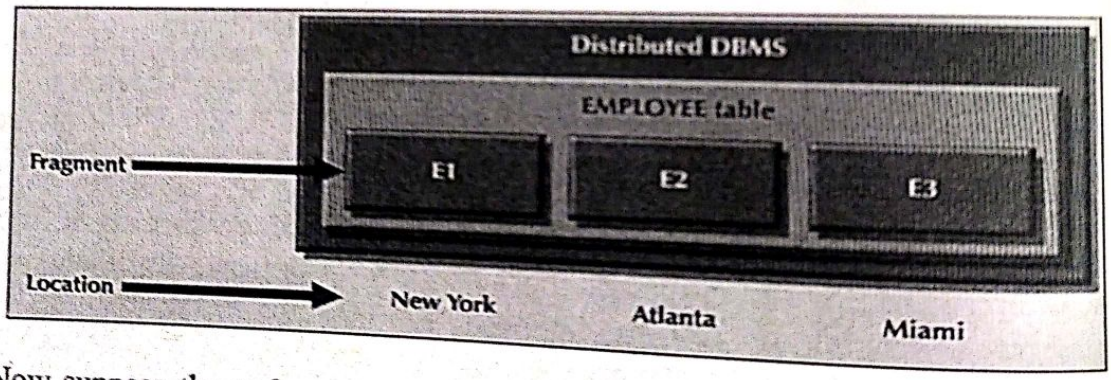
The level of transparency supported by the DDBMS varies from system to system. Three levels of distribution transparency are recognized:

- **Fragmentation transparency** is the highest level of transparency. The end user or programmer does not need to know that a database is partitioned. Therefore, neither fragment names nor fragment locations are specified prior to data access.
- **Location transparency** exists when the end user or programmer must specify the database fragment name but does not need to specify where those fragments are located.
- **Local mapping transparency** exists when the end user or programmer must specify both the fragment names and their locations.

Transparency features are summarized in the following Table

FRAGMENT NAME?	LOCATION NAME?	THEN THE DBMS SUPPORTS	LEVEL OF DISTRIBUTON TRANSPARENCY
Yes	Yes	Local mapping	Low
Yes	No	Location transparency	Medium
No	No	Fragmentation transparency	High

There is no reference to a situation in which the fragment name is "No" and the location name is "Yes." The reason for not including that scenario is simple: you cannot have a location name that fails to reference an existing fragment.



Now suppose the end user wants to list all employees with a date of birth prior to January 1, 1960. To focus on the transparency issues, also suppose the EMPLOYEE table is fragmented and each fragment is unique. Assume that no portion of the database is replicated at any other site on the network.

Case 1: The Database Supports Fragmentation Transparency

The query conforms to a non distributed database query format; that is, it does not specify fragment names or locations.

The query reads:

```
SELECT   *
FROM     EMPLOYEE
WHERE    EMP_DOB<'01-JAN-1960';
```

Case 2: The Database Supports Location Transparency

Fragment names must be specified in the query, but fragment location is not specified. The query reads:

```
SELECT   *
FROM     E1
WHERE    EMP_DOB < 'Ol-JAN-1960';
UNION
SELECT   *
FROM     E2
WHERE     EMP_DOB <'Ol-JAN-1960';
UNION
SELECT    *
FROME3
WHERE     EMP_DOB <'Ol-JAN-1960';
```

Case 3: The Database Supports Local Mapping Transparency

Both the fragment name and location must be specified in the query. Using pseudo-SQL:

```
SELECT   *
FROM     El NODE NY
WHERE    EMP_DOB < 'O1-JAN-1960;
UNION
SELECT   *
FROM     E2 NODE ATL
WHERE    EMP_DOB < O1-JAN-1960;
UNION
SELECT   *
FROM     E3 NODE MIA
WHERE    EMP_DOB <'O1-JAN-1960;
```

Distribution transparency is supported by a **distributed data dictionary (DDD)** or a **distributed data catalog(DDC)**. The DDC contains the description of the entire database as seen by the database administrator. The database description, known as the **distributed global schema**, is the common database schema used by local TPs to translate user requests into sub queries (remote requests) that will be processed by different DPs. The DDC is itself distributed and it is replicated at the network nodes. Therefore, the DDC must maintain consistency through updating at all sites.

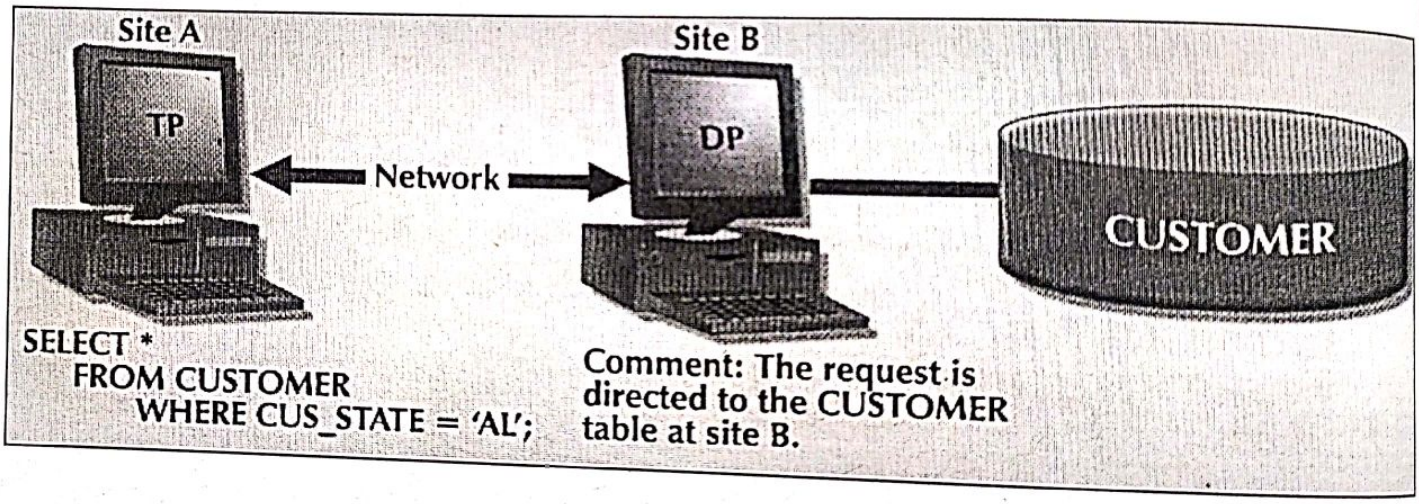
Keep in mind that some of the current DDBMS implementations impose limitations on the level of transparency support. For instance, you might be able to distribute a database, but not a table, across multiple sites. Such a condition indicates that the DDBMS supports location transparency but not fragmentation transparency.

3.6 Transaction Transparency:

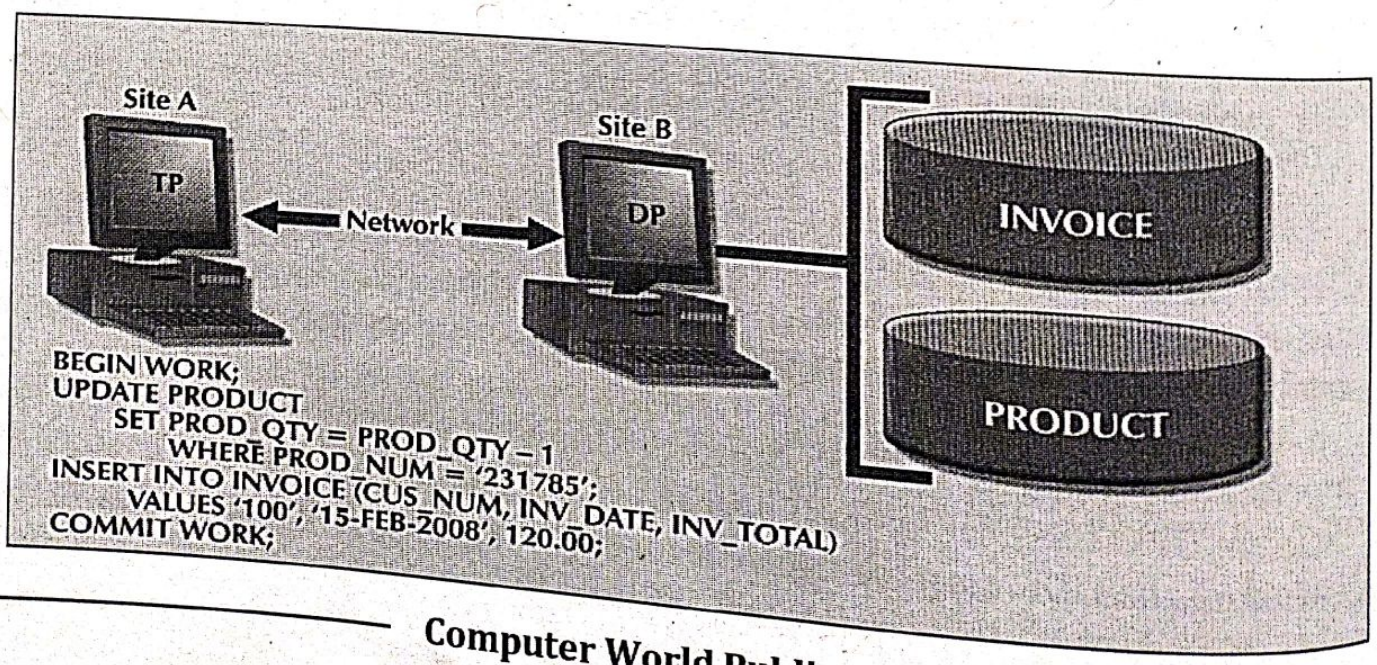
Distributed database systems require complex mechanisms to manage transactions and to ensure the consistency and integrity. To understand how the transactions are managed, you should know the basic concepts governing remote requests, remote transactions, distributed transactions, and distributed requests.

3.6.1 Distributed Requests and Distributed Transactions:

A **remote request**, lets a single SQL statement access the data that are to be processed by a single remote database processor. In other words, the SQL statement (or request) can reference data at only one remote site.



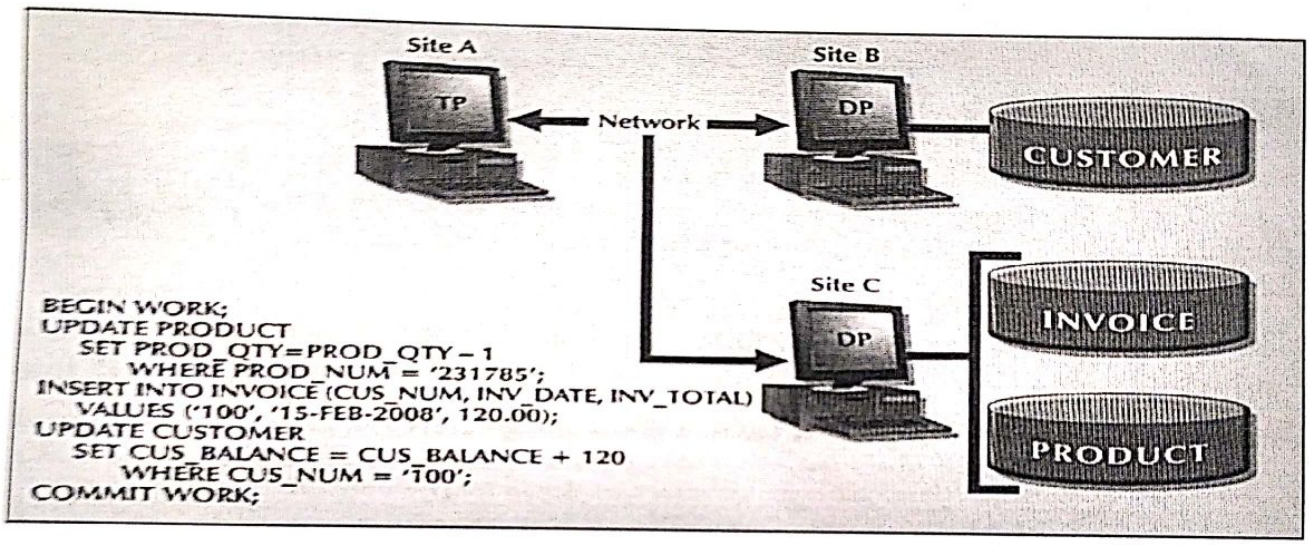
remote transaction, composed of several requests, accesses data at a single remote site.



The following remote transaction features:

- The transaction updates the PRODUCT and INVOICE tables (located at site B).
- The remote transaction is sent to and executed at the remote site B.
- The transaction can reference only one remote DP.
- Each SQL statement (or request) can reference only one (the same) remote DP at a time and the entire transaction can reference and be executed at only one remote DP.

A **distributed transaction** allows a transaction to reference several different local or remote DP sites. Although each single request can reference only one local or remote DP site, the transaction as a whole can reference DP sites because each request can reference a different site.



Note the following features:

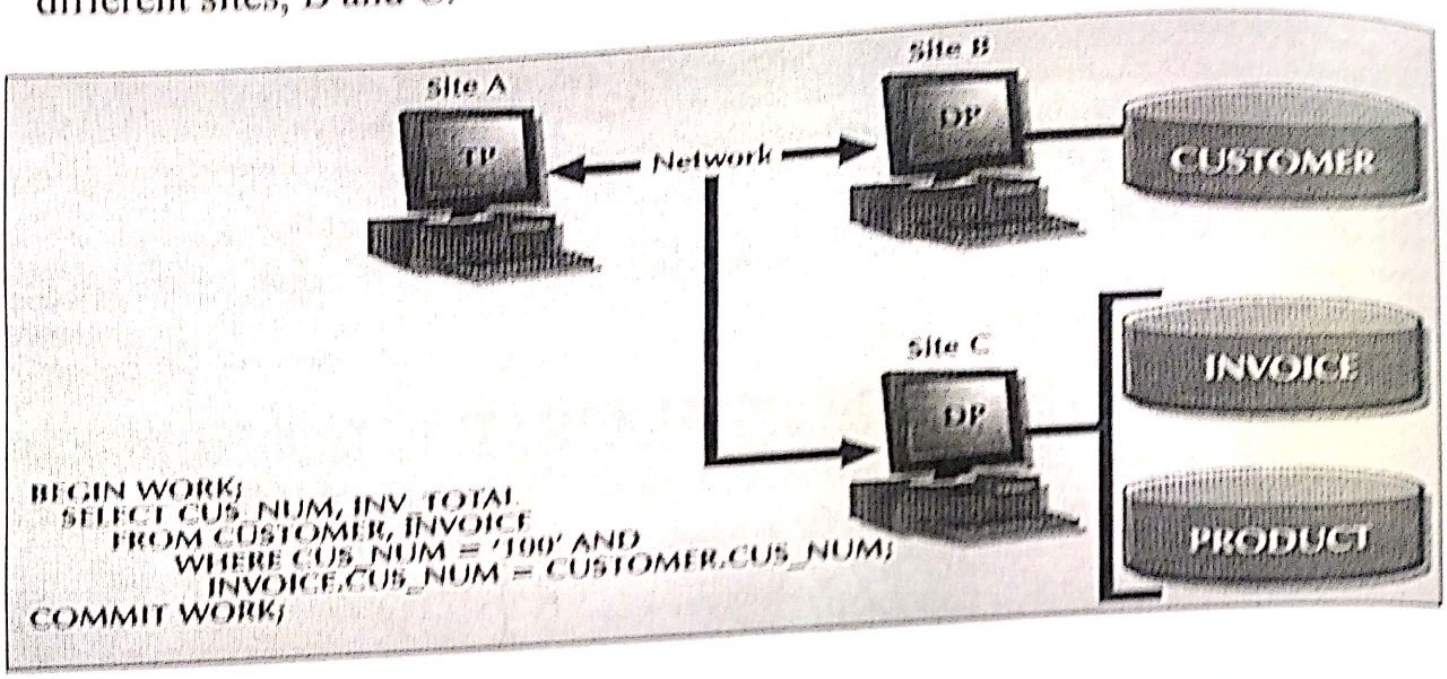
- The transaction references two remote sites (B and C).
- The first two requests (UPDATE PRODUCT and INSERT INTO INVOICE) are processed by the DP a: remote site C, and the last request (UPDATE CUSTOMER) is processed by the DP at the remote site Each request can access only one remote site at a time.

The third characteristic may create problems. For example, suppose the table PRODUCT is divided into two fragments, PRODI and PROD2, located at sites B and C, respectively.

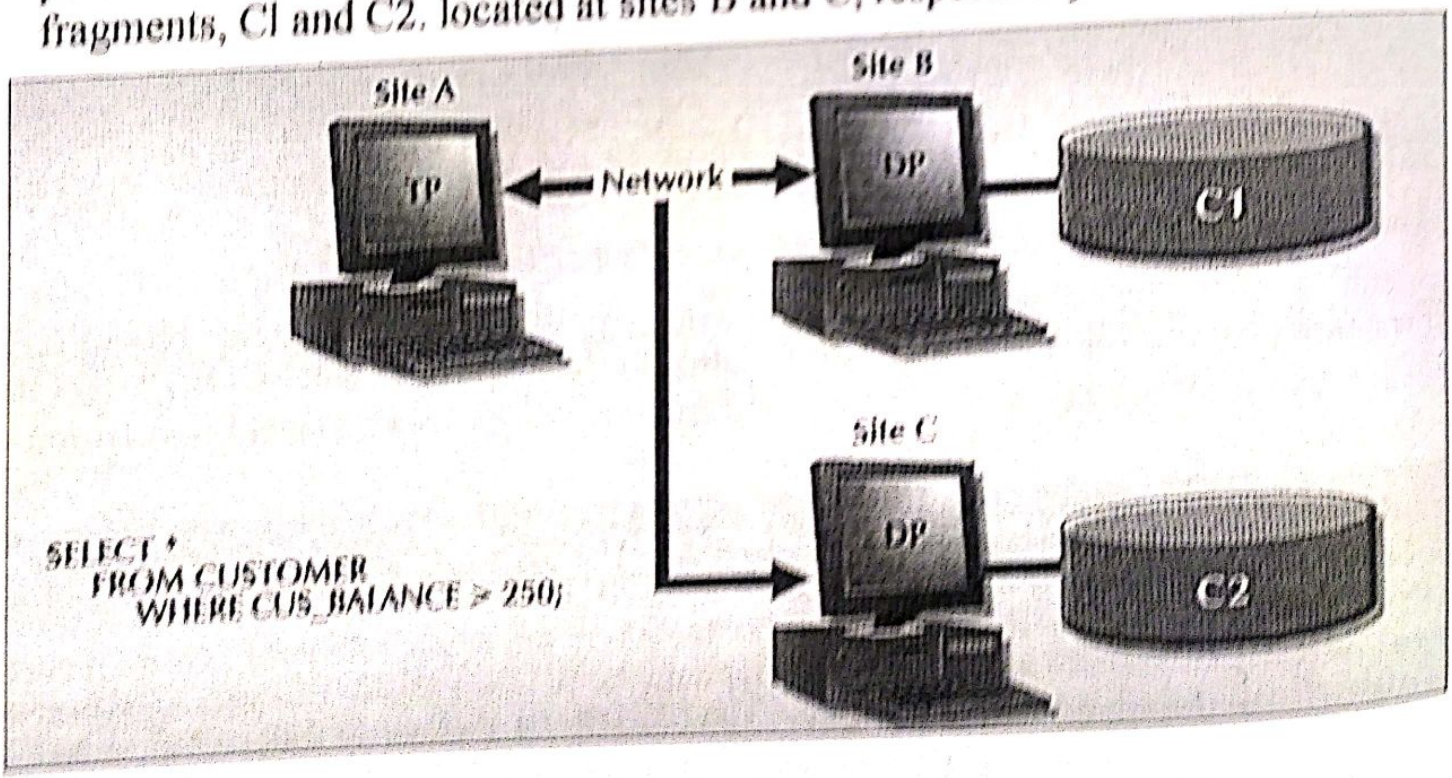
A **distributed request** lets a single SQL statement reference data located at several different local or remote DP sites. Because each request (SQL statement) can access data from more than one local or remote DP site, a transaction access several sites. The ability to execute a distributed request provides fully distributed database processing capabilities because of the ability to:

- Partition a database table into several fragments.
- Reference one or more of those fragments with only one request. In other words,

there is fragmentation transparency.
The location and partition of the data should be transparent to the end user. In the following Figure note that the transaction uses a single SELECT statement to reference two tables, CUSTOMER and INVOICE. The two tables are located at two different sites, B and C.



The distributed request feature also allows a single request to reference a physically partitioned table. For example, Suppose a CUSTOMER table is divided into two fragments, C1 and C2, located at sites B and C, respectively.

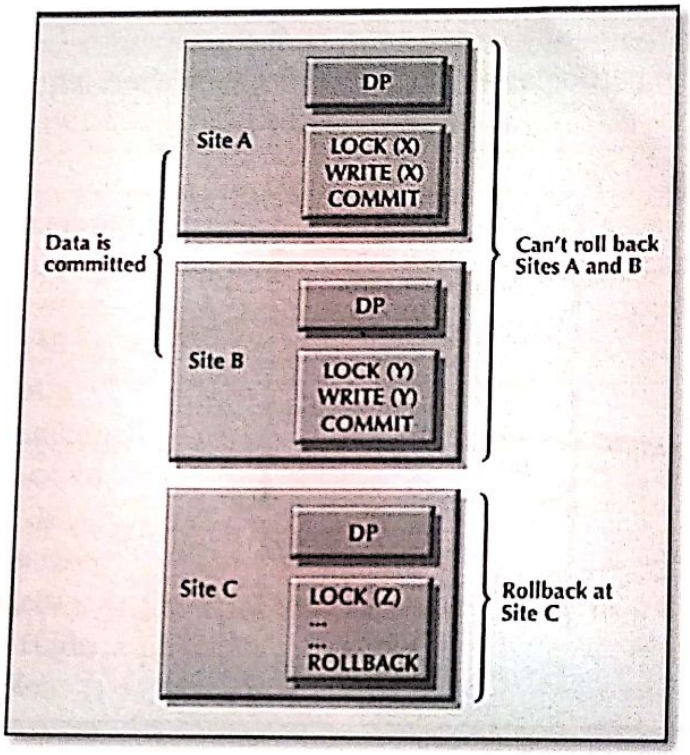


6.2 Distributed Concurrency Control:

Concurrency controlling techniques ensure that multiple transactions are executed simultaneously while maintaining the ACID properties of the transactions and serializability in the schedules. It is important because multiple process operations are

more likely to create data inconsistencies and deadlocked transactions than single site systems.

For example, a transaction updates data at three DP sites. The first two sites complete the transaction and commit the data at each local DP, while the third DP site can not commit the transaction. It will yield to an inconsistent database.



The solution of this problem is Two-phase commit protocol.

3.6.3 Two-Phase Commit Protocol:

The two-phase commit protocol requires DO-UNDO-REDO protocol and a write-ahead protocol.

The DO-UNDO-REDO protocol defines three types of operations.

- DO perform the operation and records values in the transaction log.
- UNDO undoes the operation.
- REDO redoes the operation using the log entries.

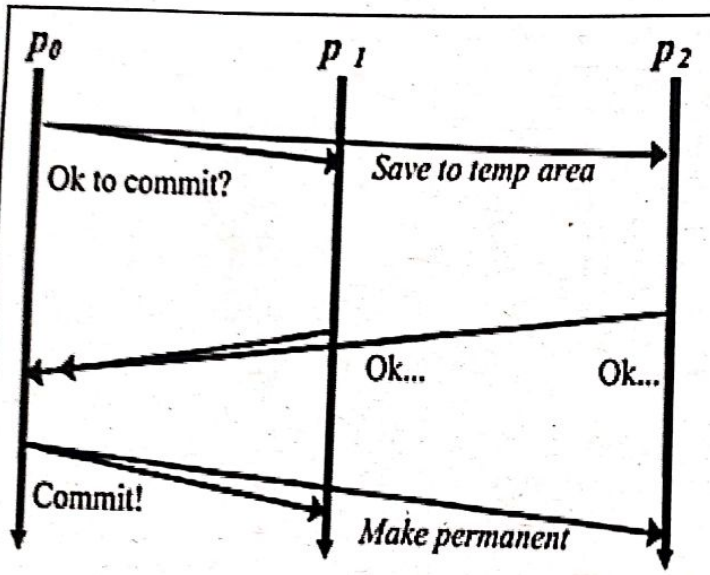
The write-ahead protocol performs the log entry to be written to permanent storage before the actual operation takes place.

The functionality of Two-phase commit protocol is as follows. In the Two-phase commit protocol, there are two phases (that's the reason it is called Two-phase commit protocol) involved. The first phase is called **prepare**. In this phase, a coordinator (either a separate node or the node initiating the transaction) makes a

request to all the participating nodes, asking them whether they are able to commit the transaction. They either return yes or no in the response. They return yes if they can successfully commit the transaction or no if they unable to do so.

In the second phase is called **final commit** in which, coordinator decides based on the votes whether to send the commit or abort request to the participating nodes. This phase is called the commit phase.

- If all the participating nodes said yes, then commit request is sent to all the nodes
- If any of the participating nodes said no, then abort request is sent to all the nodes

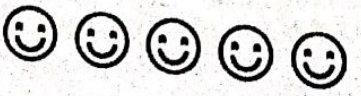


Coordinator :
multicast: *ok to commit?*
collect replies
all *ok* => *send commit*
else => *send abort*
Participant:
ok to commit =>
save to temp area, reply *ok*
commit =>
make change permanent
abort =>
delete temp area

3.7 Performance Transparency and Query Optimization:

Performance transparency requires that the performance of systems should degrade gracefully as the load on the system increases. Consistency of performance is a significant aspect of the user experience and can be more important than absolute performance. A system that has consistent good performance is better received than one in which the performance is outstanding some of the time but can degrade rapidly and unpredictably, which leads to user frustration. Ultimately, this is a measure of usability.

The main goal of query optimization is to minimize the total cost associated with the execution of a request.



Exercises

Answer the following:

1. What is DDBMS? Explain evolution of it in detail.
2. Give difference between distributed processing and distributed database.
3. List and explain levels of data and process distribution.
4. Explain distributed database transparency features.
5. Give difference between distributed request and distributed transaction.
6. Explain two-phase commit protocol in brief.

CC-212 SQL Practical (UNIT – 3)

Aggregate Functions:

SQL> **SELECT * FROM TEAMSTATS;**
OUTPUT:

```
NAME       POS  AB HITS WALKS SINGLES DOUBLES TRIPLES HR SO
---------- --- --- ---- ----- ------- ------- ------- -- --
JONES      1B 145   45 34    31      8       1       5 10
DONKNOW    3B 175   65 23    50      10      1       4 15
WORLEY     LF 157   49 15    35      8       3       3 16
DAVID      OF 187   70 24    48      4       0       17 42
HAMHOCKER  3B  50   12 10    10      2       0       0 13
CASEY      DH  1    0 0     0       0       0       0 1
```

6 rows selected.

COUNT:

COUNT the number of rows that satisfy the condition in the WHERE clause. how. many ball players were hitting under 350.
INPUT/OUTPUT:
SQL> **SELECT COUNT(*) FROM TEAMSTATS WHERE HITS/AB < .35;**

```
COUNT(*)
--------
       4
```

SUM:

SUM does just that. It returns the sum of all values in a column. To find out how many singles have been hit, type

```
SQL> SELECT SUM(SINGLES) TOTAL_SINGLES FROM TEAMSTATS;

TOTAL_SINGLES
-------------
          174

SQL> SELECT SUM(SINGLES) TOTAL_SINGLES, SUM(DOUBLES)
TOTAL_DOUBLES,
SUM(TRIPLES) TOTAL_TRIPLES, SUM(HR) TOTAL_HR
  2  FROM TEAMSTATS;

TOTAL_SINGLES TOTAL_DOUBLES TOTAL_TRIPLES TOTAL_HR
------------- ------------- ------------- --------
          174            32             5       29
```

To collect similar information on all 300 or better players, type

```
SQL> SELECT SUM(SINGLES) TOTAL_SINGLES, SUM(DOUBLES)
TOTAL_DOUBLES,
SUM(TRIPLES) TOTAL_TRIPLES, SUM(HR) TOTAL_HR
  2  FROM TEAMSTATS
  3  WHERE HITS/AB >= .300;

TOTAL_SINGLES TOTAL_DOUBLES TOTAL_TRIPLES TOTAL_HR
------------- ------------- ------------- --------
          164            30             5       29
```

To compute a team batting average, type

```
SQL> SELECT SUM(HITS)/SUM(AB) TEAM_AVERAGE
2  FROM TEAMSTATS;
TEAM_AVERAGE
------------
  .33706294
```

AVG:

The AVG function computes the average of a column. To find the average number of strike outs

```
SQL> SELECT AVG(SO) AVE_STRIKE_OUTS FROM TEAMSTATS;

AVE_STRIKE_OUTS
---------------
      16.166667
```

The following example illustrates the difference between SUM and AVG:

INPUT/OUTPUT:

```
SQL> SELECT AVG(HITS/AB) TEAM_AVERAGE
  2  FROM TEAMSTATS;

TEAM_AVERAGE
------------
  .26803448
```

MAX:

what is the highest number of hits?

```
SQL> SELECT MAX(HITS) FROM TEAMSTATS;

MAX(HITS)
---------
       70
```

MIN:

MIN does the expected thing and works like MAX except it returns the lowest member of a column. To find out the fewest at bats, type

```
SQL> SELECT MIN(AB) FROM TEAMSTATS;

MIN(AB)
---------
        1
```

INPUT/OUTPUT:

```
SQL> SELECT MIN(AB), MAX(AB) FROM TEAMSTATS;

 MIN(AB)  MAX(AB)
--------- --------
       1      187
```

INPUT/OUTPUT:

```
SQL> SELECT COUNT(AB), AVG(AB),MIN(AB), MAX(AB), STDDEV(AB),
VARIANCE(AB), SUM(AB)  FROM TEAMSTATS;

COUNT(AB) AVG(AB) MIN(AB) MAX(AB) STDDEV(AB) VARIANCE(AB)
SUM(AB)
--------- ------- ------- ------- ---------- ----------- ------
        6 119.167       1     187     75.589     5712.97    715
```

UNION and UNION ALL:

UNION returns the results of two queries minus the duplicate rows. The following two tables represent the teams:

INPUT:
SQL> **SELECT * FROM FOOTBALL;**
OUTPUT:

```
NAME
--------------------
ABLE
BRAVO
CHARLIE
DECON
EXITOR
FUBAR
GOOBER
7 rows selected.
```

INPUT:
SQL> **SELECT * FROM SOFTBALL;**
OUTPUT:

```
NAME
--------------------
ABLE
BAKER
CHARLIE
DEAN
EXITOR
FALCONER
GOOBER
7 rows selected.
```

How many different people play on one team or another?
INPUT/OUTPUT:

```
SQL> SELECT NAME FROM SOFTBALL
 2  UNION
 3  SELECT NAME FROM FOOTBALL;

NAME
--------------------
ABLE
BAKER
BRAVO
```

CHARLIE
DEAN
DECON
EXITOR
FALCONER
FUBAR
GOOBER
10 rows selected.
UNION returns 10 distinct names from the two lists.

INTERSECT:

INTERSECT returns only the rows found by both queries.
SQL> **SELECT * FROM FOOTBALL**
INTERSECT
SELECT * FROM SOFTBALL;
In this example INTERSECT finds the short list of players who are on both teams by combining the results of the two SELECT statements.

MINUS (Difference):

Minus returns the rows from the first query that were not present in the second.
SQL> **SELECT * FROM FOOTBALL**
MINUS
SELECT * FROM SOFTBALL;

NAME

BRAVO
DECON
FUBAR

JOINS:

SELECT * FROM CUSTOMER
OUTPUT:

NAME	ADDRESS	STATE	ZIP	PHONE	REMARKS
TRUE WHEEL	55O HUSKER	NE	58702	555-4545	NONE
BIKE SPEC	CPT SHRIVE	LA	45678	555-1234	NONE
LE SHOPPE	HOMETOWN	KS	54678	555-1278	NONE
AAA BIKE	10 OLDTOWN	NE	56784	555-3421	JOHN-MGR
JACKS BIKE	24 EGLIN	FL	34567	555-2314	NONE

SELECT * FROM PART

OUTPUT:

PARTNUM	DESCRIPTION	PRICE
54	PEDALS	54.25
42	SEATS	24.50
46	TIRES	15.25
23	MOUNTAIN BIKE	350.45
76	ROAD BIKE	530.00
10	TANDEM	1200.00

INPUT:

SELECT * FROM ORDERS

OUTPUT:

ORDEREDON	NAME	PARTNUM	QUANTITY	REMARKS
15-MAY-1996	TRUE WHEEL	23	6	PAID
19-MAY-1996	TRUE WHEEL	76	3	PAID
2-SEP-1996	TRUE WHEEL	10	1	PAID
30-JUN-1996	TRUE WHEEL	42	8	PAID
30-JUN-1996	BIKE SPEC	54	10	PAID
30-MAY-1996	BIKE SPEC	10	2	PAID
30-MAY-1996	BIKE SPEC	23	8	PAID
17-JAN-1996	BIKE SPEC	76	11	PAID
17-JAN-1996	LE SHOPPE	76	5	PAID
1-JUN-1996	LE SHOPPE	10	3	PAID
1-JUN-1996	AAA BIKE	10	1	PAID
1-JUL-1996	AAA BIKE	76	4	PAID
1-JUL-1996	AAA BIKE	46	14	PAID
11-JUL-1996	JACKS BIKE	76	14	PAID

Join PARTS and ORDERS:

Equi-Joins:

INPUT:

SELECT O.ORDEREDON, O.NAME, O.PARTNUM,
P.PARTNUM, P.DESCRIPTION
FROM ORDERS O, PART P
WHERE O.PARTNUM = P.PARTNUM

Or can be with a where condition also..

SELECT O.ORDEREDON, O.NAME, O.PARTNUM,
P.PARTNUM, P.DESCRIPTION

```
FROM ORDERS O, PART P
WHERE O.PARTNUM = P.PARTNUM
AND O.PARTNUM = 76
```

ORDEREDON	NAME	PARTNUM	PARTNUM	DESCRIPTION
1-JUL-1996	AAA BIKE	76	76	ROAD BIKE
17-JAN-1996	BIKE SPEC	76	76	ROAD BIKE
19-MAY-1996	TRUE WHEEL	76	76	ROAD BIKE
11-JUL-1996	JACKS BIKE	76	76	ROAD BIKE
17-JAN-1996	LE SHOPPE	76	76	ROAD BIKE

INPUT/OUTPUT:

```
SELECT  O.ORDEREDON, O.NAME, O.PARTNUM,
P.PARTNUM, P.DESCRIPTION
FROM ORDERS O, PART P
WHERE O.PARTNUM = P.PARTNUM
AND P.DESCRIPTION = 'ROAD BIKE'
```

INPUT/OUTPUT:

```
SELECT SUM(O.QUANTITY * P.PRICE) TOTAL
FROM ORDERS O, PART P
WHERE O.PARTNUM = P.PARTNUM
AND P.DESCRIPTION = 'ROAD BIKE'
```

INPUT/OUTPUT:

```
SELECT C.NAME, C.ADDRESS, (O.QUANTITY * P.PRICE) TOTAL
FROM ORDER O, PART P, CUSTOMER C
WHERE O.PARTNUM = P.PARTNUM
AND O.NAME = C.NAME
```

You could make the output more readable by writing the statement like this:

INPUT/OUTPUT:

```
SELECT C.NAME, C.ADDRESS, O.QUANTITY * P.PRICE TOTAL
FROM ORDERS O, PART P, CUSTOMER C WHERE O.PARTNUM =
P.PARTNUM
AND O.NAME = C.NAME ORDER BY C.NAME
```

```
SELECT C.NAME, C.ADDRESS, O.QUANTITY * P.PRICE TOTAL,
P.DESCRIPTION FROM ORDERS O, PART P, CUSTOMER C
WHERE O.PARTNUM = P.PARTNUM AND O.NAME = C.NAME
ORDER BY C.NAME
```

Non-Equi-Joins:

INPUT:

```
SELECT O.NAME, O.PARTNUM, P.PARTNUM,
O.QUANTITY * P.PRICE TOTAL FROM ORDERS O, PART P
WHERE O.PARTNUM > P.PARTNUM
```

Outer Joins versus Inner Joins:

INPUT:

```
SELECT P.PARTNUM, P.DESCRIPTION,P.PRICE,
O.NAME, O.PARTNUM FROM PART P JOIN ORDERS O ON
ORDERS.PARTNUM = 54
```

INPUT/OUTPUT:

```
SELECT P.PARTNUM, P.DESCRIPTION,P.PRICE,
O.NAME, O.PARTNUM FROM PART P RIGHT OUTER JOIN ORDERS O
ON ORDERS.PARTNUM = 54
```

```
SELECT P.PARTNUM, P.DESCRIPTION,P.PRICE,
O.NAME, O.PARTNUM FROM PART P LEFT OUTER JOIN ORDERS O ON
ORDERS.PARTNUM = 54
```

Joining a Table to Itself:

INPUT/OUTPUT:

```
SELECT * FROM PART
```

PARTNUM	DESCRIPTION	PRICE
54	PEDALS	54.25
42	SEATS	24.50
46	TIRES	15.25
23	MOUNTAIN BIKE	350.45
76	ROAD BIKE	530.00
10	TANDEM	1200.00
76	CLIPPLESS SHOE	65.00 <-NOTE SAME #

INPUT/OUTPUT:

```
SELECT F.PARTNUM, F.DESCRIPTION, S.PARTNUM,S.DESCRIPTION
FROM PART F, PART S WHERE F.PARTNUM = S.PARTNUM
AND F.DESCRIPTION <> S.DESCRIPTION
```

Unit -4

Advanced SQL

- Set Operators
- SQL Join
- SQL Functions
- Sub Queries
- Sequence

Unit-4 Advanced SQL

4.1 Set Operators:

The set operators are availed to combine information of similar type from one or more than one table. SQL set operators combine rows from distinct queries. There are distinct types of set operators in SQL like Union, Union All, Intersect, Minus with Examples. Let's take the sample tables to understand Set operators:

DEPT TABLE		
DEPTNO	DEPTNAME	LOCATION
10	MKTG	AHMEDABAD
20	SALES	VADODARA
30	PURCHASE	SURAT
40	MANUFACTURING	AHMEDABAD

EMP TABLE			
EMPNO	ENAME	SALARY	DEPTNO
1	NARENDRA	21000	10
2	MAHENDRA	23000	20
3	JITENDRA	18000	10
4	BHUPENDRA	35000	30
5	AMIT	45000	10
6	VIJAY	34000	20
7	NITIN	27000	40
8	NARESH	24000	10
9	RAMESH	23000	30
10	MAHESH	20000	20

1. **Union:** This set operator is used to combine the outputs of two or more queries into a single set of rows and columns having different records. It will display all non-duplicate rows retrieved from both of the table.

   Example: select deptno from emp union select deptno from dept;

   Output:

```
DEPTNO
---------
10
20
30
40
```

2. **Union All:** This set operator is used to join the outputs of two or more queries into a single set of rows and columns without the removal of any duplicates. It will display all rows including duplicate rows retrieved from both of the table.

   Example: select deptno from emp union all select deptno from dept;

   Output: It will display all the Deptno from emp and dept table.
   DEPTNO: 10 20 10 30 10 20 40 10 30 20 10 20 30 40

3. **Intersect:** This set operator is availed to retrieve the information which is common in both the tables.

   Example: select deptno from emp intersect select deptno from dept;

   Output:

   DEPTNO

   10
   20
   30
   40

   Example: select deptno from emp where empno <=5 intersect select deptno from dept;

   Output:

   DEPTNO

   10

4. **Minus:** This set operator is availed to retrieve the information of one table which is not available in another table. It returns the remaining rows when the rows retrieved by the second query are subtracted from the rows retrieved by the first query.

   Example:select deptno from emp minus select deptno from dept;

   Output: no rows selected.

   Example: select deptno from dept minus select deptno from emp where empno <=5;
   Output:

   DEPTNO

   40

4.2 SQL Joins:

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Let's take the sample tables to understand joins:

CUSTOMER TABLE

CUST_ID	CUST_NAME	CUST_BDATE	CUST_CITY
C001	RAM	10-JAN-99	AHMEDABAD
C002	RAHIM	14-APR-96	VADODARA
C003	MOHAN	05-SEP-98	SURAT
C004	SOHAN	16-MAR-00	RAJKOT

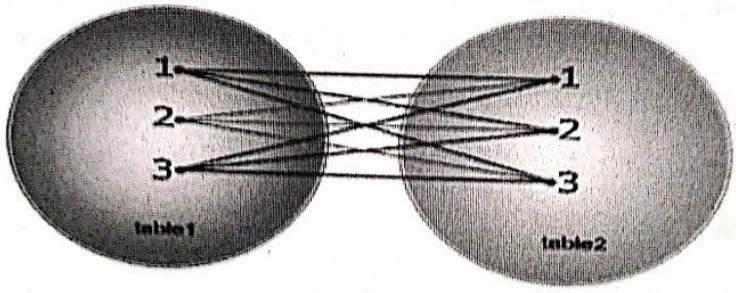
CUST_ORDER TABLE

ORDER_ID	CUST_ID	ORDER_DATE	AMOUNT
X001	C002	10-NOV-19	10000
X002	C003	10-DEC-19	14000
X003	C001	20-DEC-19	21000
X004	C003	21-DEC-19	9000
X005	C004	23-DEC-19	9000
X006	C001	25-DEC-19	11250

1. Cross Join (Cartesian Join):

The CROSS JOIN specifies that all rows from first table join with all of the rows of second table. If there are "x" rows in table1 and "y" rows in table2 then the cross join result set have x*y rows. It normally happens when no matching join columns are specified.

In simple words, if two tables in a join query have no join condition, then the Oracle returns their Cartesian product.



Example: SQL> select * from CUSTOMER, CUST_ORDER;

Output: It will display 24 Rows(6*4). Following is the output:

CUST_ID	CUST_NAME	CUST_BDATE	CUST_CITY	ORDER_ID	CUST_ID	ORDER_DATE	AMOUNT
C001	RAM	10-JAN-99	AHMEDABAD	X001	C002	10-NOV-19	10000
C001	RAM	10-JAN-99	AHMEDABAD	X002	C003	10-DEC-19	14000
C001	RAM	10-JAN-99	AHMEDABAD	X003	C001	20-DEC-19	21000

C001	RAM	10-JAN-99	AHMEDABAD	X004	C003	21-DEC-19	9000
C001	RAM	10-JAN-99	AHMEDABAD	X005	C004	23-DEC-19	9000
C001	RAM	10-JAN-99	AHMEDABAD	X006	C001	25-DEC-19	11250
SAME WAY FOR THE NEXT CUSTOMER							

2. **Natural Join:**

A NATURAL JOIN is a JOIN operation that creates an implicit join clause for you based on the common columns in the two tables being joined. Common columns are columns that have the same name in both tables.

Example: select * from CUST_ORDER natural join CUSTOMER;

CUST_ID	ORDER_ID	ORDER_DATE	AMOUNT	CUST_NAME	CUST_BDATE	CUST_CITY
C002	X001	10-NOV-19	10000	RAHIM	14-APR-96	VADODARA
C003	X002	10-DEC-19	14000	MOHAN	05-SEP-98	SURAT
C001	X003	20-DEC-19	21000	RAM	10-JAN-99	AHMEDABAD
C003	X004	21-DEC-19	9000	MOHAN	05-SEP-98	SURAT
C004	X005	23-DEC-19	9000	SOHAN	16-MAR-00	RAJKOT
C001	X006	25-DEC-19	11250	RAM	10-JAN-99	AHMEDABAD

A NATURAL JOIN can be an INNER join or LEFT OUTER join or RIGHT OUTER join. The default is INNER join.

3. **Join Using Clause:**

- Use the USING clause to specify the columns for the equijoin where several columns have the same names but not same data types.
- Use the USING clause to match only one column when more than one column matches.
- The NATURAL JOIN and USING clauses are mutually exclusive. Here, join keyword is used instead of natural join with using keyword.

Syntax:

SELECT table1.column, table2.column FROM table1 JOIN table2 USING (join_column1, join_column2...);

Example: select * from CUST_ORDER join CUSTOMER using (cust_id);

Using Table Aliases with the USING Clause:

When we use the USING clause in a join statement, the join column is not qualified with table aliases. Do not alias it even if the same column is used elsewhere in the SQL statement.

Example: Select c.cust_name, c.cust_city, o.order_id from CUST_ORDER o join CUSTOMER c using (cust_id) where cust_id<>'C001';

Or

select cust_name, cust_city, order_id from CUST_ORDER o join CUSTOMER c using (cust_id) where cust_id<>'C001';

CUST_NAME	CUST_CITY	ORDER_ID
RAHIM	VADODARA	X001
MOHAN	SURAT	X002
MOHAN	SURAT	X004
SOHAN	RAJKOT	X005

4. **Join On Clause:**

- The join condition for the natural join is basically an equijoin of identical column names.
- ON clause can be used to join columns that have different names.
- Use the ON clause to specify conditions or specify columns to join.
- The join condition is separated from other search conditions.
- This is the easiest and widely used form of the join clauses.

Syntax:

SELECT table1.column, table2.column FROM table1 JOIN table2 ON (table1.column_name = table2.column_name);

Let's INSERT 2 DUMMY RECORDS IN CUSTOMER TABLE to understand Join On Clause:

CUSTOMER TABLE (after insert 2 records)

CUST_ID	CUST_NAME	CUST_BDATE	CUST_CITY
C001	RAM	10-JAN-99	AHMEDABAD
C002	RAHIM	14-APR-96	VADODARA
C003	MOHAN	05-SEP-98	SURAT
C004	SOHAN	16-MAR-00	RAJKOT
X002	**DUMMY**	**10-OCT-01**	**AHMEDABAD**
X005	**DUMMY2**	**15-OCT-01**	**VADODARA**

Example:

Select CUSTOMER.cust_id, CUSTOMER.cust_name, CUST_ORDER.order_id from CUSTOMER join CUST_ORDER ON (CUSTOMER.cust_id=CUST_ORDER.order_id);

CUST_ID	CUST_NAME	ORDER_ID
X002	DUMMY	X002
X005	DUMMY2	X005

Example:

```
Select CUSTOMER.cust_id,CUSTOMER.cust_name,CUST_ORDER.order_id from
CUSTOMER join CUST_ORDER ON
(CUSTOMER.cust_id=CUST_ORDER.order_id) and
CUSTOMER.cust_city='AHMEDABAD';
```

CUST_ID	CUST_NAME	ORDER_ID
X002	DUMMY	X002

5. Equi Join:

Equi join returns the matching column values of the associated tables. It uses a comparison operator (=) in the WHERE clause to refer equality. It is very easy to understand.

Syntax

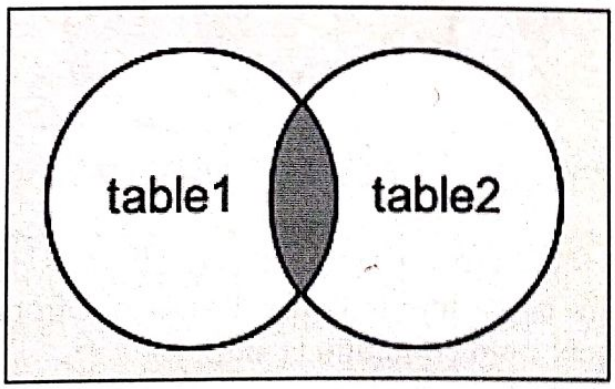
```
SELECT column_list FROM table1, table2
WHERE table1.column_name = table2.column_name;
```

Example: SELECT C.CUST_ID,C.CUST_NAME,O.ORDER_ID,O.AMOUNT FROM CUSTOMER C,CUST_ORDER O WHERE C.CUST_ID=O.CUST_ID;

CUST_ID	CUST_NAME	ORDER_ID	AMOUNT
C002	RAHIM	X001	10000
C003	MOHAN	X002	14000
C001	RAM	X003	21000
C003	MOHAN	X004	9000
C004	SOHAN	X005	9000
C001	RAM	X006	11250

6. Inner Join (Simple Join):

INNER JOINS return all rows from two or multiple tables where the join condition is met.



Example: SELECT C.CUST_ID,C.CUST_NAME,O.ORDER_ID,O.AMOUNT FROM CUSTOMER C inner join CUST_ORDER O on c.cust_id=o.cust_id;

CUST_ID	CUST_NAME	ORDER_ID	AMOUNT
C002	RAHIM	X001	10000
C003	MOHAN	X002	14000
C001	RAM	X003	21000
C003	MOHAN	X004	9000
C004	SOHAN	X005	9000
C001	RAM	X006	11250

Example:

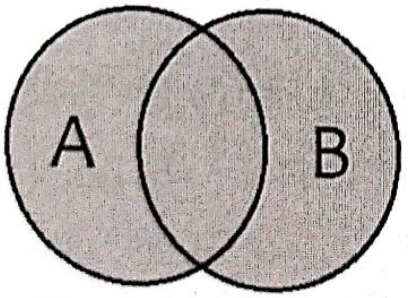
Difference between Inner Join and Equi Join:

Equi join is a special type of join in which we use only an equality operator. Hence, when you make a query for join using equality operator then that join query comes under Equi join. Inner join returns only those records/rows that match/exists in both the database tables. Following are the main differences between them.

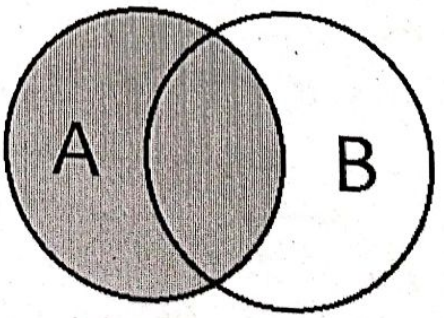
1. Inner join can have equality (=) and other operators (like <,>,<>) in the join condition.
2. Equi join only have equality (=) operator in the join condition.
3. Equi join can be an Inner join, Left Outer join, Right Outer join.

7. Outer Join:

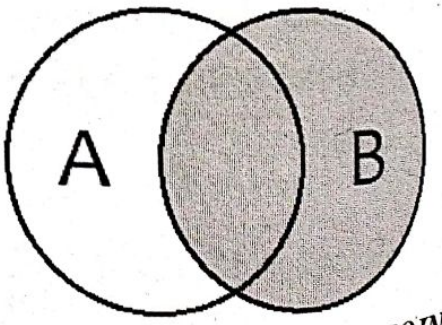
An outer join is similar to equijoin but it gets also the non-matched rows from the table. It is categorized in Full Outer Join, Left Outer Join and Right Outer Join.



FULL OUTER JOIN



LEFT OUTER JOIN



RIGHT OUTER JOIN

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;

Example:

SELECT CUSTOMER.CUST_ID,CUST_NAME,ORDER_ID,AMOUNT FROM CUSTOMER left outer join CUST_ORDER on CUSTOMER.CUST_ID = CUST_ORDER.CUST_ID;

CUST_ID	CUST_NAME	ORDER_ID	AMOUNT
C002	RAHIM	X001	10000
C003	MOHAN	X002	14000
C001	RAM	X003	21000
C003	MOHAN	X004	9000
C004	SOHAN	X005	9000
C001	RAM	X006	11250
X002	DUMMY		
X005	DUMMY2		

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;

Example:

SELECT CUSTOMER.CUST_ID, CUST_NAME, ORDER_ID, AMOUNT FROM CUSTOMER RIGHT OUTER JOIN CUST_ORDER ON CUSTOMER.CUST_ID=CUST_ORDER.CUST_ID;

CUST_ID	CUST_NAME	ORDER_ID	AMOUNT
C001	RAM	X006	11250
C001	RAM	X003	21000
C002	RAHIM	X001	10000
C003	MOHAN	X004	9000
C003	MOHAN	X002	14000
C004	SOHAN	X005	9000

Full Outer Join: The Full Outer Join returns all rows from the left hand table and right hand table. It places NULL where the join condition is not met.

Syntax:

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

Example:

SELECT CUSTOMER.CUST_ID, CUST_NAME, ORDER_ID, AMOUNT FROM CUSTOMER FULL OUTER JOIN CUST_ORDER ON CUSTOMER.CUST_ID=CUST_ORDER.CUST_ID;

CUST_ID	CUST_NAME	ORDER_ID	AMOUNT
C002	RAHIM	X001	10000
C003	MOHAN	X002	14000
C001	RAM	X003	21000
C003	MOHAN	X004	9000
C004	SOHAN	X005	9000
C001	RAM	X006	11250
X002	DUMMY		
X005	DUMMY2		

4.3 SQL FUNCTIONS:

➢ Date and Time Functions:

Function	Example	Result	Description
CURRENT_DATE	select current_date from dual	20-Dec-19	Return the current date and time in the session time zone
SYSDATE	select sysdate from dual;	20-Dec-19	Return the current date and time of the operating system where the Oracle Database resides.
EXTRACT	select extract(year from sysdate) from dual;	2019	Extract a value of a date time field e.g., YEAR, MONTH, DAY, … from a date time value.
ADD_MONTHS	select add_months(sysdate,3)	20-Mar-20	Add a number of months (n) to a

	from dual;		date and return the same day which is n of months away.
LAST_DAY	select last_day('20-dec-2019') from dual;	31-DEC-19	Gets the last day of the month of a specified date.
MONTHS_BETWEEN	SELECT MONTHS_BETWEEN(DATE '2019-07-01', DATE '2019-01-01') FROM DUAL;	6	Return the number of months between two dates.
NEXT_DAY	SELECT NEXT_DAY(DATE '2019-01-20', 'SUNDAY') FROM DUAL;	27-JAN-19	Get the first weekday that is later than a specified date.
TO_DATE	TO_DATE('01 Jan 2020', 'DD MON YYYY')	01-JAN-20	Convert a date which is in the character string to a DATE value.
TRUNC	SELECT TRUNC(DATE '2020-09-25', 'MM') FROM DUAL;	01-SEP-20	Return a date truncated to a specific unit of measure.

➢ **Numeric Functions:**

The Oracle numeric functions take a numeric input as an expression and return numeric values. The return type for most of the numeric functions is NUMBER.

Function	Example	Result	Description
ABS	SELECT ABS(5) FROM dual;	5	Calculates the absolute value of an expression
	SELECT ABS(-5) FROM dual;	5	
CEIL	SELECT 561.75 AS "Number",CEIL(561.75) FROM dual;	562	Returns the smallest whole number greater than or equal to a specified number.
	SELECT -561.75 AS "Number",CEIL(-561.75) FROM dual	561	
FLOOR	SELECT FLOOR(4.99) FROM dual;	4	Returns the largest whole number equal to or less than a specified number.
	SELECT FLOOR(-4.99) FROM dual;	-5	
MOD	SELECT MOD(7,2) FROM dual;	1	Returns the modulus of a

	SELECT MOD(-7, 2);	-1	number.
POWER	SELECT POWER(3, 2) FROM dual;	9	Returns m_value raised to the n_value power
ROUND	SELECT ROUND(4.43) FROM dual;	4	Returns the number rounded to the nearest multiple of a second number you specify or to the number of decimal places indicated by the second number.
	SELECT ROUND(-4.535,2) FROM dual;	-4.54	
	SELECT ROUND(34.4158,-1) FROM dual;	30	
SIGN	SELECT SIGN(-145), SIGN(0), SIGN(145) FROM dual;	-1 0 1	Returns a value that indicates if a specified number is less than, equal to, or greater than 0 (zero).
SQRT	SELECT SQRT(36) FROM dual	6	Computes the square root of an expression.
TRUNC	SELECT TRUNC(4.465,1) FROM dual;	4.4	Truncates a number to a specified number of decimal places.

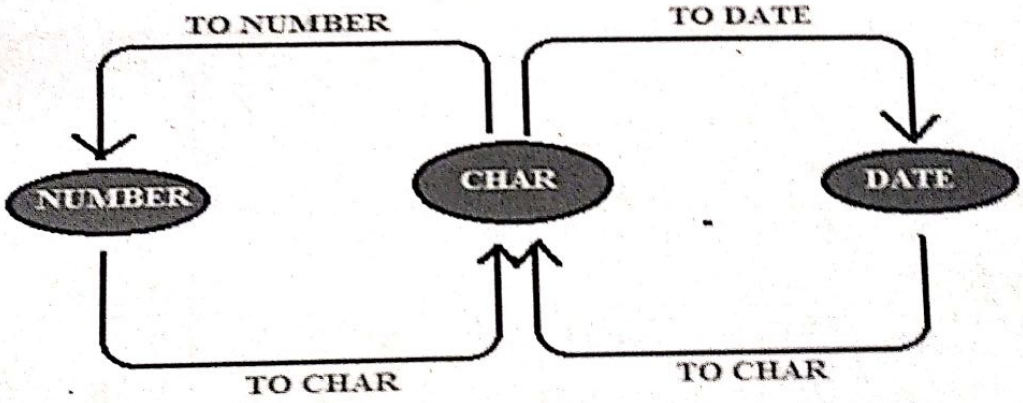
➢ **String Functions:**

Function	Example	Result	Purpose
ASCII	select ASCII('A') from dual;	65	Returns an ASCII code value of a character.
CHR	select CHR('65') from dual;	'A'	Converts a numeric value to its corresponding ASCII character.
CONCAT	select CONCAT('X','YZ') from dual;	XYZ	Concatenate two strings and return the combined string.
INITCAP	SELECT INITCAP('HELLO AMDAVAD') FROM DUAL;	Hello Amdavad	Converts the first character in each word in a specified string to uppercase and the rest to lowercase.
INSTR	SELECT INSTR('Johny Johny Yes Papa', 'Yes') FROM DUAL;	13	Search for a substring and return the location of the substring in a string
LENGTH	select LENGTH('hello') from dual;	5	Return the number of characters (or length) of a specified string
LOWER	select lower('ORACLE') from dual;	Oracle	Return a string with all characters converted to lowercase.
LPAD	select LPAD('ABC',5,'*')	'**ABC'	Return a string that is left-padded

Function	Example	Result	Purpose
	from dual;		with the specified characters to a certain length.
LTRIM	select LTRIM(' hello ') from dual;	hello	Remove spaces or other specified characters in a set from the left end of a string.
REPLACE	select REPLACE('Bhupendra and Bhupesh','Bhup','Mah') from dual	Mahendra and Mahesh	Replace all occurrences of a substring by another substring in a string.
RPAD	select RPAD('ABC',5,'*') from dual;	'ABC**'	Return a string that is right-padded with the specified characters to a certain length.
RTRIM	select RTRIM(' hello ') from dual;	hello	Remove all spaces or specified character in a set from the right end of a string.
SUBSTR	select SUBSTR('Hello How R U', 7, 7) from dual;	How R U	Extract a substring from a string.
TRIM	select TRIM(' hello ') from dual;	hello	Remove the space character or other specified characters either from the start or end of a string.
UPPER	select lower('ORacle') from dual;	oracle	Convert all characters in a specified string to uppercase.

➢ **Conversion Functions:**

Conversions functions are used to convert one data type to another type. In some cases oracle automatically converts the data to the required type. This is called implicit conversion. Explicit conversions are done by using the conversion functions. Oracle provides three functions to covert from one data type to another. one data type to another.



Explicit Data Type Converson

Function	Example	Result	Purpose
TO_CHAR(number | date, format)	select to_char (sysdate,'HH:MI:SS') from dual;	03:41:22	Convert a DATE or an INTERVAL value to a character string in a specified format.
	select to_char(sysdate,'YYYY/MM/DD') from dual;	2019/12/21	
	select to_char(sysdate,'Year/Month/DD/DY') from dual;	Twenty Nineteen/December/21/SAT	
	select to_char(10000,'$99,999') from dual;	$10,000	
TO_NUMBER(char, format)	SELECT TO_NUMBER('12,345','99,999') FROM DUAL;	12345	The TO_NUMBER function converts the characters to a number format.
TO_DATE(char,format)	SELECT TO_DATE('JAN-2020-01','MON-YYYY-DD') FROM DUAL;	01-JAN-20	The TO_DATE function converts the characters to a date data type.

4.4 Sub Queries:

Subquery means a SELECT statement which retrieves the values from the table and passes these values as argument to the main or calling SELECT statement. In simple words, nested queries are subqueries. Subqueries can be of different types:

1) Single-row subquery
2) Multiple-row subquery
3) Multiple-column subquery
4) Correlated query.

Let's take some sample table to execute subqueries.

T_NO	F_NAME	L_NAME	SALARY	SUPERVISOR	JDATE	BDATE	TITLE
T01	DR. RAMESH	PATEL	180000		10-JAN-90	10-APR-67	PRINCIPAL
T02	KAUSHAL	SHAH	120000	T01	10-APR-96	15-MAR-73	ASST. PROF
T03	VIBHA	PATEL	110000	T01	21-SEP-99	13-SEP-	ASST.

						77	PROF
T04	NILAY	JOSHI	90000	T01	20-APR-09	18-APR-84	ASST. PROF
T05	MAHESH	PATEL	60000	T01	20-MAR-06	19-APR-80	LAB OP.

Single-Row Subquery :

Single-row subquery is the query that returns only one value or row to the calling SELECT statement (from inner query). For example:

Query 1: Display the name of the customer who is oldest among all customers.
Explanation: To know the name of the customer who is oldest, you need to first find the minimum birth date and then corresponding to that date display the name of the customer.

```
select cust_id, cust_name, cust_bdate from customer where cust_bdate = (select
min(cust_bdate) from customer);
```

CUST_ID	CUST_NAME	CUST_BDATE
C002	RAHIM	14-APR-96

Query 2: Display Second Highest Order value from customer order.
Explanation: To find the second highest order value, you need to find first the maximum order value.

```
select max(amount) as "SECOND HIGHEST AMOUNT" from cust_order where
amount < (select max(amount) from cust_order);
```

SECOND HIGHEST AMOUNT
14000

Query 3: Display customer details who is older than Sohan.
Explanation: To find the customer details, you need to find first the birth date of sohan and then compare.

```
SELECT * FROM CUSTOMER WHERE CUST_BDATE < (SELECT
CUST_BDATE FROM CUSTOMER WHERE CUST_NAME='SOHAN');
```

CUST_ID	CUST_NAME	CUST_BDATE	CUST_CITY
C001	RAM	10-JAN-99	AHMEDABAD
C002	RAHIM	14-APR-96	VADODARA
C003	MOHAN	05-SEP-98	SURAT

Multiple-Row Subquery

Multiple-row subqueries are the queries that return more than one value or rows to the calling SELECT statement (inner sub query).
For example:

Query 4: Display the list of all teachers who are earning equal to any teacher who have joined before '31-dec-2000'

Explanation: First you need to know the salaries of all those who have joined before '31-dec-94' and then any teacher whose salary matches any of these returned salaries. IN operator looks for this existence into the set. You can also use Distinct to avoid duplicate salary tuples.

SELECT T_NO, F_NAME, L_NAME FROM TEACHER WHERE SALARY IN (SELECT SALARY FROM TEACHER WHERE JDATE < '31-DEC-2000');

T_NO	F_NAME	L_NAME
T01	DR. RAMESH	PATEL
T02	KAUSHAL	SHAH
T03	VIBHA	PATEL

Query 5: Display the list of all those teachers whose salary is greater than any other teacher with job title 'ASST.PROF.'.

Explanation: First you need to know the salaries of all those who are 'PRT' and then any teacher whose salary is greater than any of these returned salaries. ANY operator looks for inequality.

SELECT T_NO, F_NAME, L_NAME, SALARY FROM TEACHER WHERE SALARY > ANY (SELECT SALARY FROM TEACHER WHERE UPPER (TITLE) = 'ASST.PROF');

T_NO	F_NAME	L_NAME	SALARY
T01	DR. RAMESH	PATEL	180000
T02	KAUSHAL	SHAH	120000
T03	VIBHA	PATEL	110000

Query 6: Display the list of all those teachers whose salary is greater than all the teachers with job title as 'ASST.PROF'.

Explanation: First you need to know the salaries of all those who are 'PRT' and then any teacher whose salary is greater than all of these returned salaries. ALL operator looks for inequality.

SELECT T_NO, F_NAME, L_NAME, SALARY FROM TEACHER WHERE SALARY > ALL (SELECT SALARY FROM TEACHER WHERE UPPER (TITLE) = 'ASST.PROF');

Single-row Subquery		Multiple-row subquery	
Operator	**Description**	**Operator**	**Description**
=	Equal to	IN	Returns true if any of the values in the list match i.e. equality check

>	Greater than		
<	Less than	ALL	Returns true if all the values returned by the subquery match the condition
>=	Greater than equal to		
<=	Less than or equal to	ANY	Returns true if any of the values returned by subquery match the condition.
<>	Not equal to		

Multiple-Column subquery

Multiple-Column subquery is the subquery that returns more than one column to the calling or main SELECT statement. For example:

Query 6: Display the list of all teachers whose job title and supervisor is same as that of the employee whose first name is 'NILAY'.

Explanation: Firstly you need to find the job title and supervisor of 'NILAY' and then you need to find all other teachers whose job title and supervisor exactly matches NILAY's job title and salary.

SELECT T_NO, F_NAME,L_NAME, TITLE, SALARY FROM TEACHER WHERE (TITLE, SUPERVISOR) = (SELECT TITLE, SUPERVISOR FROM TEACHER WHERE F_NAME = 'NILAY');

T_NO	F_NAME	L_NAME	TITLE	SALARY
T02	KAUSHAL	SHAH	ASST.PROF	120000
T03	VIBHA	PATEL	ASST.PROF	110000
T04	NILAY	JOSHI	ASST.PROF	90000

Correlated Subqueries

This is another type of subquery where the subquery is executed for each and every record retrieved by the calling or main SELECT statement. A correlated subquery returns only a Boolean value (true/false). A true is returned if the subquery returns one or more records otherwise a false is returned. The operator EXISTS can be used for these types of subqueries. For example:

Query 7: Display the records in the format given below for all customer:

RAM IS A REGULAR CUSTOMER

Explanation: The main query uses the EXISTS operator to find whether the customer is a regular customer or not. If the correlated subquery returns a true value, then the record retrieved by main SELECT statement is accepted otherwise it is rejected.

Solution 1:

```
SELECT CUST_NAME, 'IS A REGULAR CUSTOMER' FROM CUSTOMER
WHERE EXISTS (SELECT * FROM CUST_ORDER WHERE
CUSTOMER.CUST_ID = CUST_ORDER.CUST_ID);
```

Output:

```
RAM        IS A REGULAR CUSTOMER
RAHIM      IS A REGULAR CUSTOMER
MOHAN      IS A REGULAR CUSTOMER
SOHAN      IS A REGULAR CUSTOMER
```

Solution 2: Using Group by and Having.

```
SELECT CUST_NAME, 'IS A REGULAR CUSTOMER' FROM CUSTOMER
WHERE EXISTS (SELECT * FROM CUST_ORDER WHERE
CUSTOMER.CUST_ID = CUST_ORDER.CUST_ID GROUP BY
CUSTOMER.CUST_ID HAVING COUNT(CUST_ORDER.CUST_ID)>=2);
```

Output:

```
CUST_NAME      'ISAREGULARCUSTOMER'

-------------- --------------------
RAM            is a Regular Customer
MOHAN          is a Regular Customer
```

4.5 Sequence:

A sequence is an object in Oracle that is used to generate a number sequence. This can be useful when you need to create a unique number to act as a primary key.

Syntax:

```
CREATE SEQUENCE sequence_name

MINVALUE value

MAXVALUE value

START WITH value

INCREMENT BY value

CACHE value;
```

If you specify none of the above clauses, then you create an ascending sequence that starts with 1 and increases by 1 with no upper limit.

Specifying only INCREMENT BY -1 creates a descending sequence that starts with -1 and decreases with no lower limit.

- To create a sequence that increments without bound, for ascending sequences, omit the MAXVALUE parameter or specify NOMAXVALUE. For descending sequences, omit the MINVALUE parameter or specify the NOMINVALUE.
- To create a sequence that stops at a predefined limit, for an ascending sequence, specify a value for the MAXVALUE parameter. For a descending sequence, specify a value for the MINVALUE parameter. Also specify NOCYCLE. Any attempt to generate a sequence number once the sequence has reached its limit results in an error.
- To create a sequence that restarts after reaching a predefined limit, specify values for both the MAXVALUE and MINVALUE parameters. Also specify CYCLE. If you do not specify MINVALUE, then it defaults to NOMINVALUE, which is the value 1.

INCREMENT BY: It specify the interval between sequence numbers. This integer value can be any positive or negative integer, but it cannot be 0. The absolute of this value must be less than the difference of MAXVALUE and MINVALUE. If this value is negative, then the sequence descends. If the value is positive, then the sequence ascends. If you omit this clause, then the interval defaults to 1.

START WITH: It specify the first sequence number to be generated. Use this clause to start an ascending sequence at a value greater than its minimum or to start a descending sequence at a value less than its maximum.

MAXVALUE: It specify the maximum value the sequence can generate. MAXVALUE must be equal to or greater than START WITH and must be greater than MINVALUE.

NOMAXVALUE: It specify NOMAXVALUE to indicate a maximum value of 10^{27} for an ascending sequence or -1 for a descending sequence. This is the default.

MINVALUE: It specify the minimum value of the sequence. MINVALUE must be less than or equal to START WITH and must be less than MAXVALUE.

NOMINVALUE: It specify NOMINVALUE to indicate a minimum value of 1 for an ascending sequence or -10^{26} for a descending sequence. This is the default.

CYCLE: It specify CYCLE to indicate that the sequence continues to generate values after reaching either its maximum or minimum value. After an ascending sequence

reaches its maximum value, it generates its minimum value. After a descending sequence reaches its minimum, it generates its maximum value.

NOCYCLE: It specify NOCYCLE to indicate that the sequence cannot generate more values after reaching its maximum or minimum value. This is the default.

CACHE: It specify how many values of the sequence the database pre- allocates and keeps in memory for faster access. The minimum value for this parameter is 2. For sequences that cycle, this value must be less than the number of values in the cycle. Maximum value allowed for CACHE must be less than the value determined by the following formula:

(CEIL (MAXVALUE - MINVALUE)) / ABS (INCREMENT)

If a system failure occurs, all cached sequence values that have not been used in committed DML statements are lost. The potential number of lost values is equal to the value of the CACHE parameter.

NOCACHE: It specify NOCACHE to indicate that values of the sequence are not pre-allocated. If you omit both CACHE and NOCACHE, the database caches 20 sequence numbers by default.

Example:

```
CREATE SEQUENCE my_seq
MINVALUE 1
MAXVALUE 100
START WITH 1
INCREMENT BY 1
CACHE 20;
```

This would create a sequence object called *my seq*. The first sequence number that it would use is 1 and each subsequent number would increment by 1 (ie: 2,3,4,...). It will cache up to 20 values for performance.

nextval statement: It is used to retrieve the next value in the sequence order.

```
SELECT MY_SEQ.NEXTVAL FROM DUAL;
```

Output: 1

If user execute the same statement again then it increments by 1 and display 2.

How to bind the sequence with Insert Statement:

```
INSERT INTO student (roll_no, stud_name) VALUES(my_seq.NEXTVAL, 'Harshi Shah');
```

Exercises

Fill in the Blanks:

1. _____Union_____ set operator is used to display all non-duplicate rows retrieved from multiple table.
2. _____Intersect_____operator is used to retrieve the information which is common in multiple tables.
3. The _____Cross join_____specifies that all rows from first table join with all of the rows of second table.
4. _____Equi join_____ Join returns the matching column values of the associated tables.
5. Inner join or simple_____ Join return all rows from two or multiple tables where the join condition is met.
6. _____Current_date_____ Function return the current date and time in the session time zone.
7. _____SYSDATE_____ Function return the current date and time of the operating system where the Oracle Database resides.
8. _____MOD_____ returns the modulus of a number.
9. The _____TO_NUMBER_____ function converts the characters to a number format.
10. The _____TO_DATE_____ function converts the characters to a date data type.
11. A _____Sequence_____ is an object which is used to generate a number sequence.

Answers:

1. Union **2.** Intersect **3.** Cross Join or Cartesian Join
4. Equi Join **5.** Inner Join or Simple Join **6.** CURRENT_DATE
7. SYSDATE **8.** MOD **9.** TO_NUMBER
10. TO_DATE **11.** Sequence

Descriptive Questions

1. Explain the difference between Union and Union all?
2. Explain Intersect and Minus with example.
3. What is the use of Join Using clause? Explain with example.
4. What is the use of Join On clause? Explain with example.
5. What is the difference between equi join and inner join?
6. Explain the types of Outer join with example.
7. Explain any three date and time function with example.
8. Explain any three numeric function with example.
9. Explain any three string function with example.
10. Explain to_char function with example.
11. What are the different types of subqueries? Explain with example.
12. What is Sequence? Explain with example.
13. What is the difference between cycle and no cycle in Sequence?
14. What is the difference between cache and no cache in sequence?

CC-212 SQL Practical (UNIT – 4)

❖ Date and Time Functions:

Function	Example	Result	Description
CURRENT_DATE	select current_date from dual	20-Dec-19	Return the current date and time in the session time zone
SYSDATE	select sysdate from dual;	20-Dec-19	Return the current date and time of the operating system where the Oracle Database resides.
EXTRACT	select extract(year from sysdate) from dual;	2019	Extract a value of a date time field e.g., YEAR, MONTH, and DAY … from a date time value.
ADD_MONTHS	select add_months(sysdate,3) from dual;	20-Mar-20	Add a number of months (n) to a date and return the same day which is n of months away.
LAST_DAY	select last_day('20-dec-2019') from dual;	31-DEC-19	Gets the last day of the month of a specified date.
MONTHS_BETWEEN	SELECT MONTHS_BETWEEN(DATE '2019-07-01', DATE '2019-01-01') FROM DUAL;	6	Return the number of months between two dates.
NEXT_DAY	SELECT NEXT_DAY(DATE '2019-01-20', 'SUNDAY') FROM DUAL;	27-JAN-19	Get the first weekday that is later than a specified date.
TO_DATE	TO_DATE('01 Jan 2020', 'DD MON YYYY')	01-JAN-20	Convert a date which is in the character string to a DATE value.

TRUNC	SELECT TRUNC(DATE '2020-09-25', 'MM') FROM DUAL;	01-SEP-20	Return a date truncated to a specific unit of measure.

❖ Numeric Functions:

Function	Example	Result	Description
ABS	SELECT ABS(5) FROM dual;	5	Calculates the absolute value of an expression
	SELECT ABS(-5) FROM dual;	5	
CEIL	SELECT 561.75 AS "Number",CEIL(561.75) FROM dual;	562	Returns the smallest whole number greater than or equal to a specified number.
	SELECT -561.75 AS "Number",CEIL(-561.75) FROM dual;	561	
FLOOR	SELECT FLOOR(4.99) FROM dual;	4	Returns the largest whole number equal to or less than a specified number.
	SELECT FLOOR(-4.99) FROM dual;	-5	
MOD	SELECT MOD(7,2) FROM dual;	1	Returns the modulus of a number.
	SELECT MOD(-7, 2);	-1	
POWER	SELECT POWER(3, 2) FROM dual;	9	Returns m_value raised to the n_value power
ROUND	SELECT ROUND(4.43) FROM dual;	4	Returns the number rounded to the nearest multiple of a second number you specify or to the number of decimal places indicated by the second number.
	SELECT ROUND(-4.535,2) FROM dual;	-4.54	
	SELECT ROUND(34.4158,-1) FROM dual;	30	
SIGN	SELECT SIGN(-145), SIGN(0), SIGN(145) FROM dual;	-1 0 1	Returns a value that indicates if a specified number is less than, equal to, or greater than 0 (zero).

SQRT	SELECT SQRT(36) FROM dual	6	Computes the square root of an expression.
TRUNC	SELECT TRUNC(4.465,1) FROM dual;	4.4	Truncates a number to a specified number of decimal places.

❖ String Functions:

Function	Example	Result	Purpose
ASCII	select ASCII('A') from dual;	65	Returns an ASCII code value of a character.
CHR	select CHR('65') from dual;	'A'	Converts a numeric value to its corresponding ASCII character.
CONCAT	select CONCAT('X','YZ') from dual;	XYZ	Concatenate two strings and return the combined string.
INITCAP	SELECT INITCAP('HELLO AMDAVAD') FROM DUAL;	Hello Amdavad	Converts the first character in each word in a specified string to uppercase and the rest to lowercase.
INSTR	SELECT INSTR('Johny Johny Yes Papa', 'Yes') FROM DUAL;	13	Search for a substring and return the location of the substring in a string
LENGTH	select LENGTH('hello') from dual;	5	Return the number of characters (or length) of a specified string
LOWER	select lower('ORACLE') from dual;	Oracle	Return a string with all characters converted to lowercase.
LPAD	select LPAD('ABC',5,'*') from dual;	'**ABC'	Return a string that is left-padded with the specified characters to a certain length.
LTRIM	select LTRIM(' hello ') from dual;	hello	Remove spaces or other specified characters in a set from the left end of a string.
REPLACE	select REPLACE('Bhupendra and Bhupesh','Bhup','Mah') from dual	Mahendra and Mahesh	Replace all occurrences of a substring by another substring in a string.
RPAD	select RPAD('ABC',5,'*') from dual;	'ABC**'	Return a string that is right-padded with the specified characters to a certain length.
RTRIM	select RTRIM(' hello ') from dual;	hello	Remove all spaces or specified character in a set from the right

Function	Example	Result	Purpose
			end of a string.
SUBSTR	select SUBSTR('Hello How R U', 7, 7) from dual;	How R U	Extract a substring from a string.
TRIM	select TRIM(' hello ') from dual;	hello	Remove the space character or other specified characters either from the start or end of a string.
UPPER	select lower('ORacle') from dual;	oracle	Convert all characters in a specified string to uppercase.

❖ Conversion Functions:

Function	Example	Result	Purpose
TO_CHAR(number | date, format)	select to_char (sysdate,'HH:MI:SS') from dual;	03:41:22	Convert a DATE or an INTERVAL value to a character string in a specified format.
	select to_char(sysdate,'YYYY/MM/DD') from dual;	2019/12/21	
	select to_char(sysdate,'Year/Month/DD/DY') from dual;	Twenty Nineteen/December/21/SAT	
	select to_char(10000,'$99,999') from dual;	$10,000	
TO_NUMBER(char, format)	SELECT TO_NUMBER('12,345','99,999') FROM DUAL;	12345	The TO_NUMBER function converts the characters to a number format.
TO_DATE(char,format)	SELECT TO_DATE('JAN-2020-01','MON-YYYY-DD') FROM DUAL;	01-JAN-20	The TO_DATE function converts the characters to a date data type.

❖ Subqueries:

1. Single Row Subquery:

Single-row subquery is the query that returns only one value or row to the calling SELECT statement (from inner query).

Let's take some sample table to execute subqueries.

T_NO	F_NAME	L_NAME	SALARY	SUPERVISOR	JDATE	BDATE	TITLE
T01	DR. RAMESH	PATEL	180000		10-JAN-90	10-APR-67	PRINCIPAL

T02	KAUSHAL	SHAH	120000	T01	10-APR-96	15-MAR-73	ASST.PROF
T03	VIBHA	PATEL	110000	T01	21-SEP-99	13-SEP-77	ASST.PROF
T04	NILAY	JOSHI	90000	T01	20-APR-09	18-APR-84	ASST.PROF
T05	MAHESH	PATEL	60000	T01	20-MAR-06	19-APR-80	LAB OP.

For example:

Query 1: Display the name of the customer who is oldest among all customers.
Explanation: To know the name of the customer who is oldest, you need to first find the minimum birth date and then corresponding to that date display the name of the customer.

select cust_id, cust_name, cust_bdate from customer where cust_bdate = (select min (cust_bdate) from customer);

CUST_ID	CUST_NAME	CUST_BDATE
C002	RAHIM	14-APR-96

Query 2: Display Second Highest Order value from customer order.
Explanation: To find the second highest order value, you need to find first the maximum order value.

select max(amount) as "SECOND HIGHEST AMOUNT" from cust_order where amount < (select max(amount) from cust_order);

SECOND HIGHEST AMOUNT
14000

Query 3: Display customer details who is older than Sohan.
Explanation: To find the customer details, you need to find first the birth date of sohan and then compare.

SELECT * FROM CUSTOMER WHERE CUST_BDATE < (SELECT CUST_BDATE FROM CUSTOMER WHERE CUST_NAME='SOHAN');

CUST_ID	CUST_NAME	CUST_BDATE	CUST_CITY
C001	RAM	10-JAN-99	AHMEDABAD
C002	RAHIM	14-APR-96	VADODARA
C003	MOHAN	05-SEP-98	SURAT

2. Multiple-Row Subquery:

Multiple-row subqueries are the queries that return more than one value or rows to the calling SELECT statement (inner sub query).

For example:

Query 4: Display the list of all teachers who are earning equal to any teacher who have joined before '31-dec-2000'

Explanation: First you need to know the salaries of all those who have joined before '31-dec-94' and then any teacher whose salary matches any of these returned salaries. IN operator looks for this existence into the set. You can also use Distinct to avoid duplicate salary tuples.

SELECT T_NO, F_NAME, L_NAME FROM TEACHER WHERE SALARY IN (SELECT SALARY FROM TEACHER WHERE JDATE < '31-DEC-2000');

T_NO	F_NAME	L_NAME
T01	DR. RAMESH	PATEL
T02	KAUSHAL	SHAH
T03	VIBHA	PATEL

Query 5: Display the list of all those teachers whose salary is greater than any other teacher with job title 'ASST.PROF.'.

Explanation: First you need to know the salaries of all those who are 'PRT' and then any teacher whose salary is greater than any of these returned salaries. ANY operator looks for inequality.

SELECT T_NO, F_NAME, L_NAME, SALARY FROM TEACHER WHERE SALARY > ANY (SELECT SALARY FROM TEACHER WHERE UPPER (TITLE) = 'ASST.PROF');

T_NO	F_NAME	L_NAME	SALARY
T01	DR. RAMESH	PATEL	180000
T02	KAUSHAL	SHAH	120000
T03	VIBHA	PATEL	110000

Query 6: Display the list of all those teachers whose salary is greater than all the teachers with job title as 'ASST.PROF'.

Explanation: First you need to know the salaries of all those who are 'PRT' and then any teacher whose salary is greater than all of these returned salaries. ALL operator looks for inequality.

SELECT T_NO, F_NAME, L_NAME, SALARY FROM TEACHER WHERE SALARY > ALL (SELECT SALARY FROM TEACHER WHERE UPPER (TITLE) = 'ASST.PROF');

3. Multiple-Column subquery:

Multiple-Column subquery is the subquery that returns more than one column to the calling or main SELECT statement. For example:

Query 7: Display the list of all teachers whose job title and supervisor is same as that of the employee whose first name is 'NILAY'.

Explanation: Firstly you need to find the job title and supervisor of 'NILAY' and then you need to find all other teachers whose job title and supervisor exactly matches NILAY's job title and salary.

SELECT T_NO, F_NAME,L_NAME, TITLE, SALARY FROM TEACHER WHERE (TITLE, SUPERVISOR) = (SELECT TITLE, SUPERVISOR FROM TEACHER WHERE F_NAME = 'NILAY');

T_NO	F_NAME	L_NAME	TITLE	SALARY
T02	KAUSHAL	SHAH	ASST.PROF	120000
T03	VIBHA	PATEL	ASST.PROF	110000
T04	NILAY	JOSHI	ASST.PROF	90000

4. Correlated Subqueries:

This is another type of subquery where the subquery is executed for each and every record retrieved by the calling or main SELECT statement. A correlated subquery returns only a Boolean value (true/false). A true is returned if the subquery returns one or more records otherwise a false is returned. The operator EXISTS can be used for these types of subqueries. For example:

Query 8: Display the records in the format given below for all customer:
RAM IS A REGULAR CUSTOMER

Explanation: The main query uses the EXISTS operator to find whether the customer is a regular customer or not. If the correlated subquery returns a true value, then the record retrieved by main SELECT statement is accepted otherwise it is rejected.

Solution 1:

```
SELECT CUST_NAME, 'IS A REGULAR CUSTOMER' FROM CUSTOMER
WHERE EXISTS (SELECT * FROM CUST_ORDER WHERE
CUSTOMER.CUST_ID = CUST_ORDER.CUST_ID);
```

Output:

```
RAM        IS A REGULAR CUSTOMER
RAHIM      IS A REGULAR CUSTOMER
MOHAN      IS A REGULAR CUSTOMER
SOHAN      IS A REGULAR CUSTOMER
```

Solution 2: Using Group by and Having.

```
SELECT CUST_NAME, 'IS A REGULAR CUSTOMER' FROM CUSTOMER
WHERE EXISTS (SELECT * FROM CUST_ORDER WHERE
CUSTOMER.CUST_ID = CUST_ORDER.CUST_ID GROUP BY
CUSTOMER.CUST_ID HAVING COUNT(CUST_ORDER.CUST_ID)>=2);
```

Output:

```
CUST_NAME       'ISAREGULARCUSTOMER'
--------------- --------------------
RAM             is a Regular Customer
MOHAN           is a Regular Customer
```

Sequence: A sequence is an object in Oracle that is used to generate a number sequence. This can be useful when you need to create a unique number to act as a primary key.

Example:

```
CREATE SEQUENCE my_seq
MINVALUE 1
MAXVALUE 100
START WITH 1
INCREMENT BY 1
CACHE 20;
```

This would create a sequence object called *my seq*. The first sequence number that it would use is 1 and each subsequent number would increment by 1 (ie: 2,3,4,...}. It will cache up to 20 values for performance.

nextval statement: It is used to retrieve the next value in the sequence order.

```
SELECT MY_SEQ.NEXTVAL FROM DUAL;
```

Output: 1

If user execute the same statement again then it increments by 1 and display 2.

How to bind the sequence with Insert Statement:

```
INSERT INTO student (roll_no, stud_name) VALUES(my_seq.NEXTVAL, 'Harshi Shah');
```

MT-101

March-2019

B.C.A., Sem-IV

CC-208: Database Management System-II

(New Course)

Time : 2:30 Hours] [Max. Marks : 70

1. **(A)** (1) Write a short-note on Special Operators used in SQL. 7

(2) How can you ADD and DROP the column in existing table using SQL query? Explain with example. 7

OR

(1) Write a short-note on SQL data types.

(2) Write a short-note on SQL data Constraint.

(B) Answer the following. (any **Four**) 4

(1) DDL is stands for Database Definition Language (True/False).

(2) _____is used to delete a table structure.

(3) The default order in ORDER BY clause is ascending. (True/False)

(4) AND/OR/NOT are comparison operators. (True/False)

(5) UPDATE command is used to modify data in a table. (True/False)

(6) The_____ command permanently saves all changes.

2. **(A)** (1) Explain Binary Locks and Shared Locks in detail. 7

(2) Write a short note on Transaction Properties. 7

OR

(1) Write a short note on Lock Granularity.

(2) Write a short-note on WAIT/DIE and WOUND/WAIT concurrency control scheme

(B) Answer the following. (Any **Four**) 4

(1) A transaction is any action that reads from and/or writes to a database. (True/False)

(2) A DBMS uses a transaction log to keep track of all transactions that update the database. (True/False)

(3) Inconsistent retrievals occur when two transactions are updating the same data elements and one of the updates is lost. (True/False)

(4) _____ occurs when two transaction wait indefinitely for each other to unlock data.

(5) The time stamps value produces an_____ order in which transactions are submitted to the DBMS.

(6) Database recovery restores a database from a given state to a previously consistent state. (True/False)

3. **(A)** (1) Write a short note on Two-Phase Commit Protocol 7

(2) Write a short note on Distributed Database Transperancy Features. 7

OR

(1) Write a short note on Distributed Processing and Distributed Database.

(2) Write a short note on Levels of Data and Process Distribution.

(B) Answer the following (Any **three**) 3

(1) DDBMS stands for_____.

(2) A DDBMS database transaction can update data stored in many different computers connected in a network. (True/False)

(3) _____is the hoghest level of transparency.

(4) A_____transparency allows a transaction to update the data at more than one network sites.

(a) transaction (b) failure

(c) performance (d) heterogeneity.

(5) SPSD is stands for single site processing, single site data. (True/False)

4. **(A)** (1) Explain "JOIN ON" clause and "JOIN USING" clause with example. 7

(2) Explain Interesect and Union Operator with example. 7

OR

(1) Explain TO_CHAR and TO_NUMBER conversion function with example.

(2) Write a short note on Oracle Sequences.

(B) Answer the following. (Any **three**) 3

(1) Cross join is also known as_____.

(2) Minus yields only the rows that appear in both the tables. (True/False)

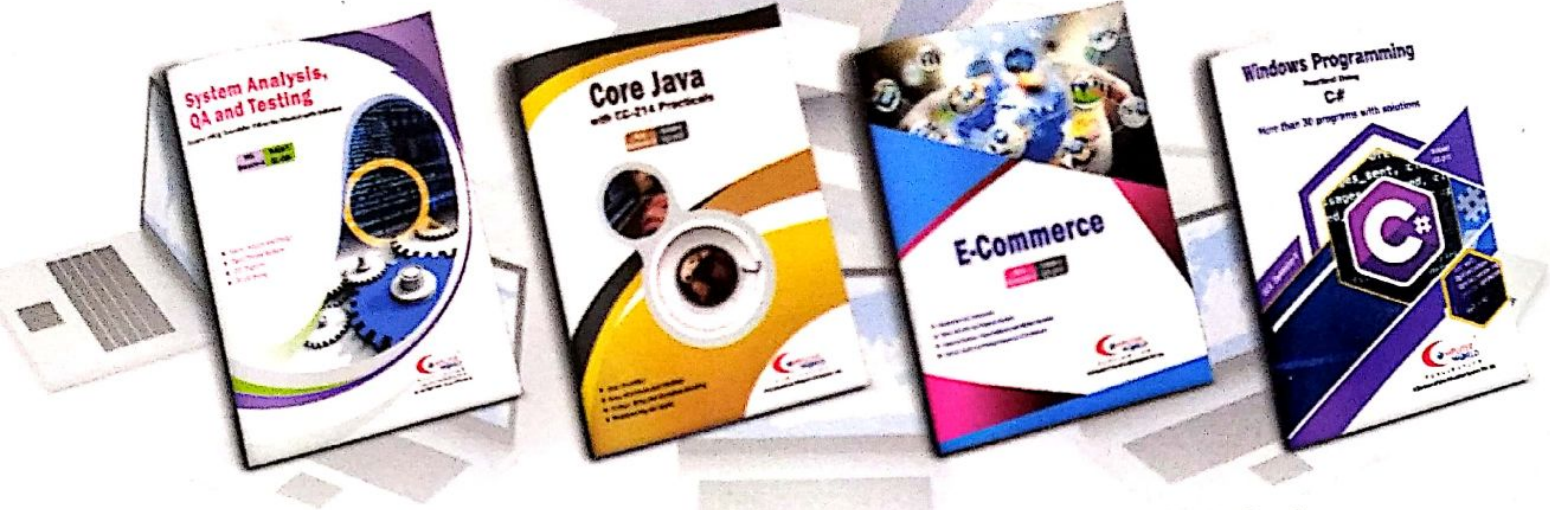
(3) _____ function returns today's date.

(4) TO_DATE returns a date value using character string and a format mask. (True/False)

(5) A sub query is a query inside a query. (True/False)

According to the New Syllabus o the Gujarat University

Study Material for Bachelor of Computer Application (BCA)



Exercise (MCQ, True-False, Fill in the Blanks) with Solution

More Books for BCA

Semester-1, Semester-2, Semester-3, Semester-4, Semester-5 and Semester-6

Shop Online @ www.computerworld.ind.in

COMPUTER WORLD PUBLICATION
A Division of Live Education System Pvt. Ltd.
An ISO 9001:2008 Certified Company

COMPUTER WORLD PUBLICATION
43, 5th Floor, Sanidhya Complex, Opp. Sanyas Ashram,
Nr. M. J. Library,Ashram Road, Ahmedabad-380009.
Ph. : (079) 26580723 / 823, 97240 11150, 97250 22917
URL: www.computerworld.ind.in, www.cworld.co.in
e-Mail : info@computerworld.ind.in

978-93-88092-20-3
Computer World

Book Code : CEBCA114 | Price ₹ 100/-