KCT COLLEGE OF ENGG AND TECHNOLOGY

DEPTT. OF COMPUTER SCIENCE AND ENGG.

LAB MANUAL

Subject: **Relational database and management system-I**

Semester-5th

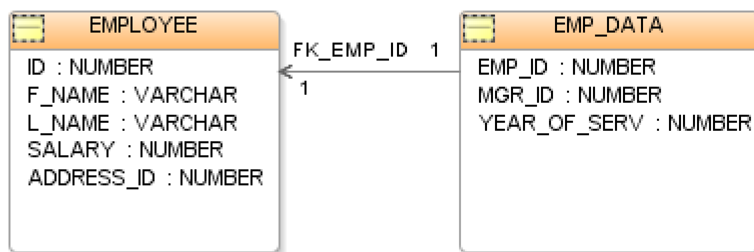
INDEX

S.NO.	EXPERIMENT NAME
1	Introduction to DBMS
2.	To Define the different type of commands in DDL, DML, DCL, and TCL.
3.	To Explain the Aggregate function, Numeric function using Master_Client table and String Function.
4.	To use constraints on the created database
5.	Explain The All Joins.
6.	To Explain The Some Addition Query.

Experiment No -1

Aim: Introduction to DBMS

Theory:-A Database Management System (DBMS) is a set of computer programs that controls the creation, maintenance, and the use of a database. It allows organizations to place control of database development in the hands of database administrators (DBAs) and other specialists. A DBMS is a system software package that helps the use of integrated collection of data records and files known as databases. It allows different user application programs to easily access the same database. DBMSs may use any of a variety of database models, such as the network model or relational model. In large systems, a DBMS allows users and other software to store and retrieve data in a structured way. Instead of having to write computer programs to extract information, user can ask simple questions in a query language. Thus, many DBMS packages provide Fourth-generation programming language (4GLs) and other application development features. It helps to specify the logical organization for a database and access and use the information within a database. It provides facilities for controlling data access, enforcing data integrity, managing concurrency, and restoring the database from backups. A DBMS also provides the ability to logically present database information to users.



Components of DBMS

- **DBMS Engine** accepts logical request from the various other DBMS subsystems, converts them into physical equivalents, and actually accesses the database and data dictionary as they exist on a storage device.
- **Data Definition Subsystem** helps user to create and maintain the data dictionary and define the structure of the files in a database.
- **Data Manipulation Subsystem** helps user to add, change, and delete information in a database and query it for valuable information. Software tools within the data manipulation subsystem are most often the primary interface between user and the information contained in a database. It allows user to specify its logical information requirements.
- **Application Generation Subsystem** contains facilities to help users to develop transaction-intensive applications. It usually requires that user perform a detailed series of tasks to process a transaction. It facilitates easy-to-use data entry screens, programming languages, and interfaces.

- **Data Administration Subsystem** helps users to manage the overall database environment by providing facilities for backup and recovery, security management, query optimization, concurrency control, and change management.

Modeling language

A data modelinglanguage to define the schema of each database hosted in the DBMS, according to the DBMS database model. The four most common types of models are the:

- Hierarchical model,
- Network Model,
- Relational Model, and
- Object Model.

Inverted lists and other methods are also used. A given database management system may provide one or more of the four models. The optimal structure depends on the natural organization of the application's data, and on the application's requirements (which include transaction rate (speed), reliability, maintainability, scalability, and cost).

The **Hierarchical Structure** was used in early mainframe DBMS. Records' relationships form a treelike model. This structure is simple but nonflexible because the relationship is confined to a one-to-many relationship. IBM's IMS system and the RDM Mobile are examples of a hierarchical database system with multiple hierarchies over the same data. RDM Mobile is a newly designed embedded database for a mobile computer system. The hierarchical structure is used primary today for storing geographic information and file systems.

The **Network Structure** consists of more complex relationships. Unlike the hierarchical structure, it can relate to many records and accesses them by following one of several paths. In other words, this structure allows for many-to-many relationships.

The **Relational Structure** is the most commonly used today. It is used by mainframe, midrange and microcomputer systems. It uses two-dimensional rows and columns to store data. The tables of records can be connected by common key values. While working for IBM, E.F. Codd designed this structure in 1970. The model is not easy for the end user to run queries with because it may require a complex combination of many tables.

The multidimensional structure is similar to the relational model. The dimensions of the cube looking model have data relating to elements in each cell. This structure gives a spreadsheet like view of data. This structure is easy to maintain because records are stored as fundamental attributes, the same way they're viewed and the structure is easy to understand. Its high performance has made it the most popular database structure when it comes to enabling online analytical processing (OLAP).

The **Object Oriented Structure** has the ability to handle graphics, pictures, voice and text, types of data, without difficultly unlike the other database structures. This structure is popular for

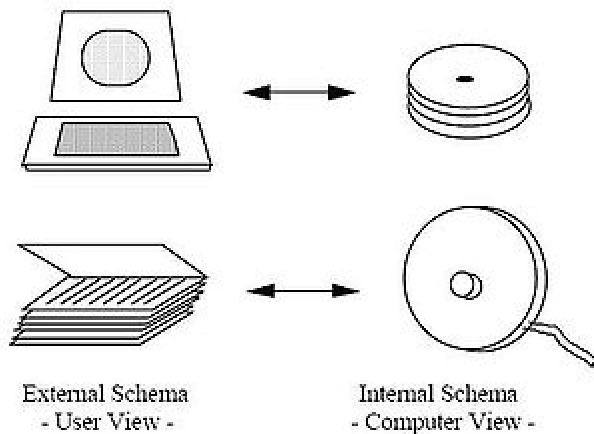
multimedia Web-based applications. It was designed to work with object-oriented programming languages such as Java.

Database query language

A database query language and report writer allows users to interactively interrogate the database, analyze its data and update it according to the user's privileges on data. It also controls the security of the database. Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called ***subschemas*.** For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data.

If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases. However, it may not leave an audit trail of actions or provide the kinds of controls necessary in a multi-user organization. These controls are only available when a set of application programs are customized for each data entry and updating function.

External, Logical and Internal view



Traditional View of Data

A database management system provides the ability for many different users to share data and process resources. But as there can be many different users, there are many different database needs. The question now is: How can a single, unified database meet the differing requirement of so many users?

A DBMS minimizes these problems by providing two views of the database data: an external view (or User view), logical view (or conceptual view)and physical(or internal) view. The user's view, of a database program represents data in a format that is meaningful to a user and to the software programs that process those data.

One strength of a DBMS is that while there is typically only one conceptual (or logical) and physical (or internal) view of the data, there can be an endless number of different External views. This feature allows users to see database information in a more business-related way rather than from a technical, processing viewpoint. Thus the logical view refers to the way user views data, and the physical view to the way the data are physically stored and processed.

Experiment No-2

Aim: To Define the different type of commands in DDL, DML, DCL, and TCL.

Theory: The Different types of command are in the following:

1. Create table:

This command is create a row and column of the table uniquely, Each column has three attributes, a name, a data type and a size (column width).

Rules:

1. A name can have maximum up to 30 characters.
2. Alphabets from A-Z, a-z, and number from 0-9 are allowed.
3. A name should begin with an alphabet
4. The use of the special character like _ is allowed and also recommended.
5. SQL reserved words not allowed. For example: create, select and so on.

Syntax:

Create table <table name>(<column name><data type>(data size),<column name><data type>(data size),<column name><data type>(data size));

2. Insert:

Once a table is created, the most natural thing to do is load this table with data to be manipulated later

When inserting a single row of data into table, the insert operation:

- Create a new row (empty) in the data base table.
- Loads the values passed (by the SQL insert) into column specified.

Syntax:

Insert into <table name> values (<expression1>, <expression2>, <expression3>);

3. Viewing data in the table

Once data has been inserted into a table, the next most logical operation would be to view what has been inserted. The SELECT SQL verb is used to achieve this. The SELECT command is used to retrieve row selected from one or more table.

The entire row and the entire column are display at a time

Syntax:
Select * from <tablename>;

To specific column from a table can be done
Syntax: Select <column1><column2> from <table name>

4. **Using Where Command:**

We might want to conditionally select the data from a table. For example, we may want to only retrieve stores with sales above $1,000. To do this, we use the **WHERE** keyword. The syntax is as follows:

SELECT "column_name"FROM "table_name" WHERE "condition"

5. **Updating the contents of a table:**

The update command is used to change or modify data values in a table.

The verb update in SQL is used to either update:

- All the rows from a table

  Or

- A select set of rows from a table

The update statement updates columns in the existing table rows with new values; the SET clause indicates which column data should be modified and the new values that they should hold. The WHERE clause, if given specifies which rows should be updated, otherwise all table rows are updated.

Syntax

Update tablename set ColumnName=expression where columnName=expression;

6. **MODIFYING THE STRUCTURE OF TABLES**

The structure of a table can be modified by using the ALTER table command. Alter table allows3 changing the structure of an existing table, with ALTER table, it is possible to add or delete columns, create or destroy indexes, change the data type of existing column is increased.

(a) Adding a new columns

Syntax:

Alter table tablenameadd(newcolumndatatype(size));

(b) Dropping a column from a table

Alter table tablename drop column columnname;

(c) Modifying existing columns

Alter table tablenameModify(columnnamenewdatatype(newsize));

Restrictions on the ALTER table

- Change the name of the table
- Change the name of the column
- Decrease the size of a column if table data exists

7. Logical Operators:

(a) AND Operator: The AND operator allows creating as SQL statement based on two or more conditions being met, it can be used in any valid SQL statement such as Select ,insert, update, or delete. The AND operator requires that each condition must be met for the record to be included in the result set.

(b) OR Operator : The OR operator allows creating as SQL statement where records are returned when any one of the conditions being met, it can be used in any valid SQL statement such as Select ,insert, update, or delete. The OR operator requires that each condition must be met for the record to be included in the result set.

(c) NOT Operator : The oracle engine will process all rows in a table and display only those records that do not satisfy the condition specified.

8. RENAMING TABLES

Oracle allows renaming of tables; the rename operation is done atomically, which means that no other thread can access any of the tables while the rename process is running.

Rename <oddtableName> to <newtablename>

9. ELIMINATING DUPLICATE ROWS WHEN USING A SELECT STATEMENT

A table could hold duplicate rows. In such a case, to view only unique rows the distinct clause can be used.

The DISTINCT clause allows removing duplicates from the result set, the Distinct clause can only be used with select statements.

The Distinct clause scans through the values of the column specified and displays only unique values from amongst them.

Syntax

SELECT DISTINCT "column_name" FROM "table_name";

10. EXAMINING OBJECTS CREATED BY A USER

Finding out the tables created by a user

The command shown below is used to determine the tables to which a user has access. The created under the currently selected table space are display.

(a) Display the table Structure: To Display information about the columns defined in a table use the Following

Syntax:

Describe <tablename>;

Or

Desc<tablename>

11. Sorting data in a table:

Oracle allows data from a table to be viewed in sorted order. The rows retrieved from the table will be sorted in either ascending or descending order depending on

the condition specified in the SELECT sentence. The syntax for viewing data in a sorted order is as follows:

(a) Ascending order: -

Syntax: select * from <tablename> order by <columnname1>;

(b) Descending order: -

Syntax: select * from <tablename> order by <columnname1>desc;

12. Range Searching:

In order to select data that is within a range of values, the **BETWEEN** operator allows the selection of rows that contain values within a specified lower and upper limit, the range coded after the word between is inclusive.

The two values in **BETWEEN** the range must be linked with the keyword AND. The between operator can be used with both character and numeric data types. However, the data types cannot be mixed.

Syntax:

Select * from<table name>**where** <column name>**between** min **AND** max;

13. IN operator:

The arithmetic operator (=) compares a single values to another single value. In case a value needs to be compared to a list of values then the IN predicates is used. The IN predicates helps reduce the need to use multiple or conditions

Syntax: select * from <Table name> where <column name> in (101,105);

14. Pattern Matching

The use of the like predicate

The comparison operators discussed so far have compared one value, exactly to one other value, such precision may not always be desired or necessary, for this purpose oracle provides the like predicates.

The like predicates allows comparison of one string value with other string values, which is not identical. This is achieved by using wildcard characters. Two wildcard characters that are available are:

For character data types:

- % allows to match any string of any length(including Zero length)
- _ allows to match on a single character

Syntax:

Select * from tablename where columnname like '_a%';

Experiment No-3

Aim: To Explain the Aggregate function, Numeric function using Master_Client table and String Function.

Theory:

SQL> select * from master_client;

CLIENT	NAME	CITY	PINCODE	STATE	BAL_DUE	TELEPHONE
C00001	WAN BEGAN	MUMBAI	400054	MAHARASHTRA	1000	786547433
C00003	CHHAYA BANKER	MUMBAI	400057	MAHARASHTRA	5000	426456566
C00004	ASHWANI JOSHI	BANGALORE	560001	KARNATKA	0	453433
C00005	HANSEL CALACO	MUMBAI	400060	MAHARASHTRA	2000	45564332
C00006	DEEPAK SHARMA	MANGALORE	560050	KARNATKA	0	7656443

(a) Aggregate Function

- **Avg :-** returns an average value of 'n', ignoring null values in a column.
  Syntax: Avg ([<distinct>|<all>]<n>)

Example:

SQL> select avg (baldue) from master;

AVG(BALDUE)

1333.33333

- **Count:-** returns the numbers of rows where expr is not null.
  Syntax: : Count ([<distinct>|<all>]<expr>)

SQL> select count (city) from master;

COUNT (CITY)

6

- **Sum:** returns the sum of the value of ‘n’.
  Syntax: Sum ([<distinct>|<all>]<n>)

SQL> select sum (baldue) from master;

SUM(BALDUE)

8000

- **Min:-** returns a minimum value of expr.
  Syntax: : Min ([<distinct>|<all>]<expr>)
  SQL> select min (baldue) from master;

MIN(BALDUE)

0

- **Max:-** returns a minimum value of expr.
  Syntax: : Max ([<distinct>|<all>]<expr>)

SQL> select max(baldue) from master;

MAX(BALDUE)

5000

(b) Numeric Function

- **Exp:-** returns e raised to the **n^{th}** power, where **e=2.71828183**
  Syntax: EXP(n)
  SQL> select exp(5) from dual;

EXP(5)

148.413159

- **ABS**:- returns the absolute value of ‘n’;
  Syntax: abs(n)

SQL> select abs(-56) from dual;

ABS(-56)

56

- **Power :-** returns m raised to the **n**thpower. **N** must be an integer, else an error is returned.
  Syntax: power(m,n)

SQL> select power(3,2) from dual;

POWER(3,2)

9

- **Round:** returns **n**, rounded to m places to the right of a decimal point, if m is omitted, **n** is rounded to 0 places, m can be negative to round off digits to the left of the decimal point. **M** must be an integer.
  Syntax: Round(n[m])

SQL> select round(15.32569754,2) from dual;

ROUND(15.32569754,2)

15.33

- **Sqrt:-** returns square root of n. if **n<0**,Null sqrt returns a real result
  Syntax:-Sqrt (n)
  SQL> select sqrt(98) from dual;

SQRT(98)

9.89949494

- **Greatest**: returns the greatest value in a list of expressions.
  Syntax:- greatest(exp1,exp2,-----------------,exp_n)
  Where, exp1, exp2, ------, exp_n are expressions that are evaluated by the greatest function.
  SQL> select greatest (5, 6, 7) from dual;

GREATEST(5,6,7)

7

- **Least:-** returns the least value in a list of expressions
  Syntax: least (exp1,exp2,-----------------,exp_n)
  Where, exp1, exp2, ------,exp_n are expressions that are evaluated by the least function.

SQL> select least (4, 56, 9) from dual;

LEAST(4,56,9)

4

- **Mod:** returns the remainder of a first number divided by second number passed a parameter, if the second number is zero; the result is the same as the first number.
  Syntax: MOD (m, n)

SQL> select mod (2, 2) from dual;

MOD(7,2)

1

- **Trunc:**-returns a number truncated to a certain number of a decimal places. The decimal place value must be an integer. If this parameter is omitted, the TRUNC function will truncate the number to a 0 decimal places.
  Syntax: **Trunc (number,[decimal_places])**
  SQL> select trunc(128.357,2) from dual;

TRUNC(128.357,2)

128.35

- **Floor:-** returns the largest integer value that is equal to or less than a number.
  Syntax:- floor(n)
  SQL> select floor (24.67) from dual;

FLOOR (24.67)

24

- **Ceil:** - returns the smallest integer value that is greater than or equal to a number.
  Syntax:- CEIL (n)
  SQL> select ceil (24.89) from dual;

CEIL (24.89)

25

(c) String Function

- **Lower:** Returns char, with all letters in lowercase.

Syntax: lower (char)

Example: SQL> select lower ('BFCET') “LOWER” from dual;

LOWER

-----

bfcet

- **Initcap**: Returns a string with the first letter of each word in upper case.

Syntax: initcap(char)

Example SQL> select initcap('nitin') from dual;

INITC

Nitin

- **Upper**: Returns char, with all letters forced to uppercase.

Syntax: upper(char)

Example SQL> select upper('bfcet') from dual;

UPPER

BFCET

- **Substr:** Returns a portion of characters , beginning at character m,and going upto character n. if n is omitted ,the result returned is upto the last character in the string. The first position of char is 1.

Syntax: substr(<string>, <start_position>,[<length>])

Example SQL> select substr('nitin',2,3) from dual;

SUB

---

iti

- **ASCII:** Returns the number code that represents the specified character, if more than one character is entered , the function will return the value for the first character and ignore all of the characters after the first.

Syntax: ascii(<single_character>)

Example SQL> select ascii('A') from dual;

ASCII('A')

65

- **Compose:** Returns a Unicode string .it can be a char, varchar2, nchar, nvarcar2, clob, or nclob.

Syntax : compose(<single>)

Below is a listing of unstring values that can combined with other characters in the compose function.

Unistring value	Resulting character
Unistr('\0300')	Grave accent(`)
Unistr('\0301')	Acute accent(')
Unistr('\0302')	Circumflex(^)
Unistr('\0303')	Tidle(~)
Unistr('\0308')	Umlaut(")

Example SQL> SELECT COMPOSE ('a'||unistr('\301)) from dual;

Compose

----

a`

- **Decompose**: Accepts a string and unicode string.

Syntax: decompose(<single>)

Example SQL> SELECT DECOMPOSE(COMPOSE ('a'||unistr('\301))) from dual;

DECompose

----

a`

- **Instr:** Returns the locaton of a substring in a string.

Syntax: instr(<string1>,<string2>,[start_postion], [nth_ postion])

Where string 1 is the string is serach.

String2 is the sbstrin g to search for in string1.

Start postion is the postion in string1 where the serach will start.

Example SQL> select instr('nitin is bfcet student','i',1) from dual;

INSTR('NITINISBFCETSTUDENT','I',1)

2

- **Translate**: Replace a sequence of characters in a string with another set of characters. How ever it replaces a single character at a time.
  Syntax:
  Translate (<string1>, <string_to_replace>,<replacement_string>)
  Example SQL> SELECT TRANSLATE (‘1SCT523’,’123’,’7A9’) FROM DUAL;
  TRANSLATE

  7SCT5A9
- **Length**: Returns the length of a word.
  Syntax: length (word)
  Example SQL> select length('nitin') from dual;

  LENGTH ('NITIN')

  5
- **LTRIM:** Removes characters from the left of char with initial characters removed upto the first character not in set.
  Syntax: **LTRIM (char(set))**
  Example SQL> select ltrim ('nitin','n') from dual;

  LTRI
  ----
  itin
- **RTRIM:** Returns char, with final characters removed after the last character not in the set, set is optional, it defaults to spaces.

Syntax: RTRIM (char, [set])
Example SQL> select rtrim('nitin','n') from dual;

RTRI
----
niti

- **TRIM:** Removes all specified characters either from the beginning or the ending of a strin.
  Syntax: Trim ([leading | Trailing | both[<trim_character>from]] <string1>)
  Leading removes trim_ string from the front of string1.
  Trailing removes trim_ string from the end of string1.
  Both remove trim_ string from the front and end of string1.
  Example SQL> select trim(leading 'x' from 'xxxnitinxx') from dual;

  TRIM(LE
  ----
  nitinxx

  Example SQL> select trim(trailing 'x' from 'xxxnitinxxx') from dual;

  TRIM(TRA
  ----
  xxxnitin

  Example SQL> select trim(both 'x' from 'xxxnitinxxx') from dual;

  TRIM(

  Nitin

- **LPAD:** Returns char1,left-padded to length n with the sequence of characters specified in char2.if char2 is not specified oracle uses blanks by details.
  Syntax: LPAD (char1,n[,char2])
  Example SQL> select LPAD ('nitin', 10,'*') from dual;

  RPAD ('NITIN')
  ----

nitin*****

- **RPAD**: Returns char1, right -padded to length n with the sequence of characters specified in char2.if char2 is not specified oracle uses blanks by details.
  Syntax: RPAD (char1,n[,char2])
  Example SQL>selectrpad('nitin',10,'*') from dual;

  RPAD ('NITIN')

  *****nitin
- **VSIZE:** Returns the number of bytes in the internal representation of an expression.
  Syntax:
  VSize(<expression>)
  Example SQL> select vsize('hello world') from dual;

  VSIZE('HELLOWORLD')

  11

Experiment No-4

Aim: - To use constraints on the created database

Theory:

Data Constraints table attached to table columns SQL syntax that checks data for integrity prior storage. Once data constraints are part of table column construct, the oracle database engine checks the data being entered into a table column against the data constraints. If the data passes this check, it is stored in the table column, else the data is rejected. Even if a single column of the record being entered into the table fails a constraint, the entire record is rejected and not stored in the column.

Types of data constraints:-

1. Not Null
2. Primary Key
3. Foreign Key
4. Check
5. Unique

1. Not Null

By default all columns in a table can contain null values. If you want to ensure that a column must always have a value, i.e. it should not be left blank, then define a NOT NULL constraint on it.

Always be careful in defining NOT NULL constraint on columns,

For Example in employee table some employees might have commission and some employees might not have any commission. If you put NOT NULL constraint on COMM column then you will not be able insert rows for those employees whose commission is null. Only put NOT NULL constraint on those column which are essential for example in EMP table ENAME column is a good candidate for NOT NULL constraint.

2. Primary Key

Each table can have one primary key, which uniquely identifies each row in a table and ensures that no duplicate rows exist. Use the following guidelines when selecting a primary key:

- Whenever practical, use a column containing a sequence number. It is a simple way to satisfy all the other guidelines.
- Minimize your use of composite primary keys. Although composite primary keys are allowed, they do not satisfy all of the other recommendations. For example, composite primary key values are long and cannot be assigned by sequence numbers.
- Choose a column whose data values are unique, because the purpose of a primary key is to uniquely identify each row of the table.

- Choose a column whose data values are never changed. A primary key value is only used to identify a row in the table, and its data should never be used for any other purpose. Therefore, primary key values should rarely or never be changed.
- Choose a column that does not contain any nulls. A PRIMARYKEY constraint, by definition, does not allow any row to contain a null in any column that is part of the primary key.
- Choose a column that is short and numeric. Short primary keys are easy to type. You can use sequence numbers to easily generate numeric primary keys.

Syntax:

<column name>datatype<datasize> Primary Key;

For example in EMP table EMPNO column is a PRIMARY KEY.

Create table EMP (empnovarchar2(10) PRIMARY KEY, ename varchar2(10), city varchar2(10));

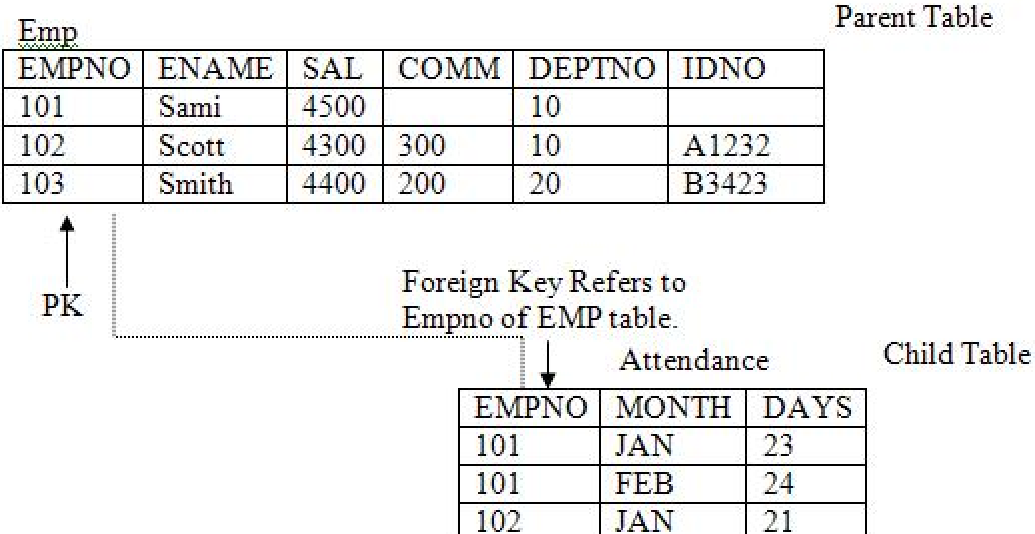
3. FOREIGN KEY

On whichever column you put FOREIGN KEY constraint then the values in that column must refer to existing values in the other table. A foreign key column can refer to primary key or unique key column of other tables. This Primary key and foreign key relationship is also known as PARENT-CHILD relationship i.e. the table which has Primary Key is known as PARENT table and the table which has Foreign key is known as CHILD table. This relationship is also known as REFERENTIAL INTEGRITY.

Syntax:-

<column name><data types><size> REFERENCES <table name > [<column name >]

The following shows an example of parent child relationship.



Emp

EMPNO	ENAME	SAL	COMM	DEPTNO	IDNO
101	Sami	4500		10	
102	Scott	4300	300	10	A1232
103	Smith	4400	200	20	B3423

Attendance

EMPNO	MONTH	DAYS
101	JAN	23
101	FEB	24
102	JAN	21

EXAMPLE:

First table is parent table you have to create a simple way. After than you have to create a table child then you write this:

Create table child (empnonumber(5) Primary Key, Month char(10),days number(6) REFERENCES Parent (empno));

Some points to remember for referential integrity

You cannot delete a parent record if any existing child record is there. If you have to first delete the child record before deleting the parent record. In the above example you cannot delete row of employee no. 101 since its child exist in the ATTENDANCE table. However, you can delete the row of employee no. 103 since no child record exists for this employee in ATTENDANCE table. If you define the FOREIGN KEY with ON DELETE CASCADE option then you can delete the parent record and if any child records exist it will be automatically deleted.

4. **Check:-**

A check constraints takes substantially longer to execute as compared to **NOT NULL, PRIMARY KEY, FOREIGN KEY OR UNIQUE**. Thus check constraints must be avoided if the constraint can be using the not null, primary key or foreign key constraint.

Syntax:

<column name><data type><size> CHECK (<logical expression>)

Example:

Create table emp (empnamevarchar2(10) CHECK (empname LIKE ‘A%’));

5. **UNIQUE KEY**

Unique Key constraint is same as primary key i.e. it does not accept duplicate values, except the following differences

- There can be only on Primary key per table. Whereas, you can have as many Unique Keys per table as you want.

- Primary key does not accept NULL values whereas; unique key columns can be left blank.

You can also refer to Unique key from Foreign key of other tables.

On which columns you should put Unique Key Constraint

It depends on situations, first situation is suppose you have already defined a Primary key constraint on one column and now you have another column which also should not contain any duplicate values, Since a table can have only one primary key, you can define Unique Key constraint on these columns. Second situation is when a column should not contain any duplicate value but it should also be left blank. For example in the EMP table IDNO is a good candidate for Unique Key because all the IDNO's are unique but some employees might not have ID Card so you want to leave this column blank.

To define a UNIQUE KEY constraint on an existing table give the following command.

Alter table emp add constraint id_unique unique (idno);

Again the above command will execute successfully if IDNO column contains complying values otherwise you have to remove non complying values and then add the constraint

Experiment No-5

Aim:-Explain The All Joins.

Theory: Joins

The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables.

Tables in a database are often related to each other with keys.

A primary key is a column (or a combination of columns) with a unique value for each row. Each primary key value must be unique within the table. The purpose is to bind data together, across tables, without repeating all of the data in every table.

Types of joins:

- Inner
- Outer(LEFT, RIGHT, FULL)
- Cross
- **Inner join:** inner join is also known as Equi joins, there are the most common joins used in it. They are known as equi joins because the where statement generally compares two columns from two table with the equivalence operator =. this type of the join is by far the most commonly
  Used. In fact many systems use this type as the default join. There are two types of style:
  (1) ANSI Style
  (2) Theta style
  (1) ANSI Style:
  Syntax:
  Select <column1><column2>from <table1>**inner join**<table2> on
  <table1>.<column1>= <table2>.<column1>;
  (2) Theta style:
  Select <column1><column2><column>from <table1>,<table2>
  Where <table1>.<column1>= <table2>.<column1>;

- **Outer join:** outer joins are similar to inner joins, but give a bit more flexibility when selecting data from related tables. This types of join can be used in situation where it is desired, to select all row from the table on the **left(or right or full)** regardless of whether the other table has values in common and usually enter NULL where data is missing.
  Left join:
  Select <col1>, <col2>, <coln> from <table1>**LEFT JOIN**<table2> on
  <table1>.<column1>= <table2>.<column1>;

Right join:
Select <col1>,<col2>,<coln> from <table1>**RIGHT JOIN**<table2> on <table1>.<column1>= <table2>.<column1>;

FULL join:
Select <col1>,<col2>,<coln> from <table1>**FULL JOIN**<table2> on <table1>.<column1>= <table2>.<column1>;

- **Cross join :**A cross join returns what's known as a Cartesian product. This means that the join combines every row from the left table with every row in the right table. As can be imagined, sometimes this join produces a mess, but under the right circumstances, it can be very useful. This type of join can be used in situation where it is desired.

Syntax:

Select <col1>,<col2>,<coln> from <table1> CROSS JOIN <table2>;

Experiment-6

Aim: To Explain The Some Addition Query.

Theory:

(1) Creating a table from a table

The source table is the table identified in the SELECT section of the SQL sentence. The TARGET table is one identified in the CREATE section of this SQL sentence. This SQL sentence populates the TARGET table with data from the Source table.

Syntax:

CREATE table <new table name>(<col1,col2,-----, col n>) as select col1, col2,coln from <old table name>;

(2) Inserting data into table from another table

In addition data one row at a time into a table, it is quite possible to populate a table with data that already exists in another table.

Syntax:

Insert into <new table name> select <col1,col2—coln> from <old table name>;

(3) Defining integrity constraints via the alter table command

Integrity constraints can be defined using the constraints clause, in the ALTER TABLE command.

Primary key

Alter table <table name> add Primary key(col1);

(4) Dropping integrity constraints via the alter table command

Integrity constraints can be dropped if the rule that it enforces is no longer true or if the constraints is no longer needed Drop the constraints using the ALTER table command with the drop clause.

Alter table <table name> drop Primary key;

(5) Default value concept:

At the time of table creation a default value can be assigned to a column. When a record is loaded into the table ,and the column is left empty, the oracle engine will automatically load this

column with the default value specified. The data type of the default value should match the data type of the column. The DEFAULT clause can be used to specify a default value for a column.

Create table <table name>(column name <data type><size> default <value>);

(6) UNION

Multiple queries can be put together and their output can be combined using the union clause.

The union clause merges the output of two or more queries into a single set of rows and column.

While working with the union clause the following pointers should be considered:

- The number of column and the data types of the columns being selected must be identical in all the select statement used in query. The names of the columns need not be identical.
- UNION operates over all of the columns being selected.
- NULL values are not ignored during duplicate checking.
  Syntax:
  Select * from <table name1> UNION Select * from <table name2>;

(7) Intersect clause

Multiple queries can be put together and their output combined using the intersect clause. The intersect clause outputs only rows produced by both the queries intersected i.e. the output in an intersect clauses will include only those rows that retrieved common to both the queries.

While working with the intersect clause the following pointers should be considered:

- The number of columns and the data types of the column s being selected by the SELECT statement in the queries must be identical in all the SELECT statement used in the query. The names of the columns need not be identical.
- Reversing the order of the intersected tables does not alter the result.
- INTERSECT does not ignore NULL values.

  Syntax:
  Select * from <table name1> intersect Select * from <table name2>;