****

**VISWAJYOTHI COLLEGE OF ENGINEERING AND TECHNOLOGY, VAZHAKULAM**

**VISION**

Moulding Engineers par Excellence with integrity, fairness and human values

**MISSION**

* We commit to develop the institution as a Center of Excellence   of International Standards.
* We guide our students in the attainment of intellectual and professional competence for successfully coping with the rapid advancements in technologies and the ever changing world of business, industry and services.
* We help each and every student in their personal growth into mature and responsible individuals.
* We strive to cultivate a sense of social and civic responsibility in our students, thus empowering them to serve the humanity.
* We promise to ensure a free environment where quest for the truth is encouraged.

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**VISION**

Moulding socially responsible and professionally competent Computer Engineers to adapt to the dynamic technological landscape.

**MISSION**

* Foster the principles and practices of computer science to empower life-long learning and build careers in software and hardware development.
* Impart value education to elevate students to be successful, ethical and effective problem-solvers to serve the needs of the industry, government, society and the scientific community.
* Promote industry interaction to pursue new technologies in Computer Science and provide excellent infrastructure to engage faculty and students in scholarly research activities.



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**PROGRAM EDUCATIONAL OBJECTIVES**

1. A graduate must be a proficient computer connoisseur able to solve a wide range of computing-related problems.
2. To equip graduates with integrity and ethical values so that they become responsible Engineers.
3. Apply computer science knowledge to application areas from science and industry.
4. A graduate must have the background and perspective necessary to pursue post-graduate education

**PROGRAM OUTCOMES**

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design / development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to ones own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning** : Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**PROGRAM SPECIFIC OUTCOMES**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

* Ability to integrate theory and practice to construct software systems of varying complexity.
* Able to Apply Computer Science skills, tools and mathematical techniques to analyze, design and model complex systems.
* Ability to design and manage small-scale projects to develop a career in a related industry.
  + - 1. **Syllabus**

**CS010 507 Database Lab**

**Teaching scheme**

**Credits: 2**

**3 hours practical per week**

Objectives

• To acquaint the students with the implementation and fundamental algorithms of database systems.

• To provide experience on design, querying, and processing of data in a relational database.

I. Experiments to implement the following

1. Relational algebra operations select, project and join.

2. Determination of Attribute Closure, Candidate Key, Functional Dependency.

3. Checking Serializability of a Schedule.

4. Dynamic Hashing.

II. Experiments in any relational database for the following

1. Creation, Insertion, Updation, Deletion of Tables, Indexes, Views.

2. Simple Queries, Nested Queries, Use of Arithmetic and String Functions.

3. Simple PL/SQL Programs, Use of Exceptions, Cursor, Procedure, Function, Trigger, Sequence.

4. Report Generation

5. ODBC/JDBC Interface.

Any experiment according to the syllabus of CS010 503 can be substituted.

Resources: 1 SQL,PL/SQL”Ivan Bayross”, BPB Publication 3rd Ed.

**INTRODUCTION**

Oracle has many tools such as SQL \* PLUS, Oracle Forms, Oracle Report Writer, Oracle Graphics etc.

* **SQL \* PLUS**: The SQL \* PLUS tool is made up of two distinct parts. These are
  + **Interactive SQL:** Interactive SQL is designed for create, access and manipulate data structures like tables and indexes.
  + **PL/SQL:** PL/SQL can be used to developed programs for different applications.
* **Oracle Forms:** This tool allows you to create a data entry screen along with the suitable menu objects. Thus it is the oracle forms tool that handles data gathering and data validation in a commercial application.
* **Report Writer:** Report writer allows programmers to prepare innovative reports using data from the oracle structures like tables, views etc. It is the report writer tool that handles the reporting section of commercial application.
* **Oracle Graphics:** Some of the data can be better represented in the form of pictures. The oracle graphics tool allows programmers to prepare graphs using data from oracle structures like tables, views etc.

**SQL (Structured Query Language):**

Structured Query Language is a [database](http://en.wikipedia.org/wiki/Database) [computer language](http://en.wikipedia.org/wiki/Programming_language) designed for managing [data](http://en.wikipedia.org/wiki/Data) in [relational database management systems](http://en.wikipedia.org/wiki/Relational_database_management_system)(RDBMS), and originally based upon [Relational Algebra](http://en.wikipedia.org/wiki/Relational_Algebra). Its scope includes data query and update, [schema](http://en.wikipedia.org/wiki/Database_schema) creation and modification, and data access control. SQL was one of the first languages for [Edgar F. Codd](http://en.wikipedia.org/wiki/Edgar_F._Codd)'s [relational model](http://en.wikipedia.org/wiki/Relational_model) in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks"[[3]](http://en.wikipedia.org/wiki/SQL#cite_note-codd-relational-model-2#cite_note-codd-relational-model-2) and became the most widely used language for relational databases.

* IBM developed SQL in mid of 1970’s.
* Oracle incorporated in the year 1979.
* SQL used by IBM/DB2 and DS Database Systems.
* SQL adopted as standard language for RDBS by ASNI in 1989.

**DATA TYPES:**

**CHAR (Size):** This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can hold. The maximum number of character is 255 characters.

* 1. **VARCHAR (Size) / VERCHAR2 (Size)**: This data type is used to store variable length alphanumeric data. The maximum character can hold is 2000 character.
  2. **NUMBER (P, S):** The NUMBER data type is used to store number (fixed or floating point). Number of virtually any magnitude may be stored up to 38 digits of precision. Number as large as 9.99 \* 10 124. The precision (p) determines the number of places to the right of the decimal. If scale is omitted then the default is zero. If precision is omitted, values are stored with their original precision up to the maximum of 38 digits.
  3. **DATE:**  This data type is used to represent date and time. The standard format is dd-mm-yy as in 17-SEP-2009. To enter dates other than the standard format, use the appropriate functions. Date time stores date in the 24-Hours format. By default the time in a date field is 12:00:00 am, if no time portion is specified. The default date for a date field is the first day the current month.
  4. **LONG:** This data type is used to store variable length character strings containing up to 2GB. Long data can be used to store arrays of binary data in ASCII format. LONG values cannot be indexed, and the normal character functions such as SUBSTR cannot be applied.
  5. **RAW:** The RAW data type is used to store binary data, such as digitized picture or image. Data loaded into columns of these data types are stored without any further conversion. RAW data type can have a maximum length of 255 bytes. LONG RAW data type can contain up to 2GB.

**interactive SQL:**

***syntax :***verb(Parameter\_1,Parameter\_2,Parameter\_3,........Parameter\_n);

**SQL language is sub-divided into several language elements, including:**

* + *Clauses*, which are in some cases optional, constituent components of statements and queries.
  + *Expressions,* which can produce either [scalar](http://en.wikipedia.org/wiki/Scalar_(computing)) values or [tables](http://en.wikipedia.org/wiki/Table_(database)) consisting of [columns](http://en.wikipedia.org/wiki/Column_(database)) and [rows](http://en.wikipedia.org/wiki/Row_(database)) of data.
  + *Predicates* which specify conditions that can be evaluated to SQL [three-valued logic (3VL)](http://en.wikipedia.org/wiki/Ternary_logic) Boolean truth values and which are used to limit the effects of statements and queries, or to change program flow.
  + *Queries* which retrieve data based on specific criteria.
  + *Statements* which may have a persistent effect on schemas and data, or which may control transactions, program flow, connections, sessions, or diagnostics.
  + SQL statements also include the [semicolon](http://en.wikipedia.org/wiki/Semicolon) (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.
  + [*Insignificant white space*](http://en.wikipedia.org/wiki/Whitespace_(computer_science)) is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

There are five types of SQL statements. They are:

1. data definition LANGUAGE (ddl)

2. data manipulation language (dml)

3. DATA RETRIEVAL LANGUAGE (DRL)

4. TRANSATIONAL CONTROL LANGUAGE (TCL)

5. DATA CONTROL LANGUAGE (DCL)

**1. data definition LANGUAGE (ddl):** The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. Let's take a look at the structure and usage of four basic DDL commands:

1. CREATE 2. ALTER 3. DROP 4. RENAME

**1. CREATE:**

**(a)create table:** This is used to create a new relation and the corresponding

***Syntax:*** create table relation\_name

(field\_1 data\_type(Size),field\_2 data\_type(Size), .. . );

***Example:***

SQL>create table Student (sno NUMBER(3),sname char(10),class char(5));

**(b)create TABLE..as select....:** This is used to create the structure of a new relation from the structure of an existing relation.

***Syntax:*** create table (relation\_name\_1, field\_1,field\_2,.....field\_n) AS SELECT field\_1,field\_2,...........field\_n from relation\_name\_2;

***Example:*** SQL>create table std(rno,sname) as select sno,sname from student;

**2. ALTER:**

**(a)ALTER TABLE ...ADD...:** This is used to add some extra fields into existing relation.

***Syntax:*** ALTER TABLE relation\_name ADD(new field\_1 data\_type(size), new field\_2 data\_type(size),..);

***Example :***  SQL>ALTER TABLE std ADD(Address CHAR(10));

**(b)ALTER table...modify...:** This is used to change the width as well as data type of fields of existing relations.

***Syntax:*** alter table relation\_name modify (field\_1 newdata\_type(Size), field\_2 newdata\_type(Size),....field\_newdata\_type(Size));

***Example:***SQL>alter table student modify(sname varchar(10),class varchar(5));

**3. drop table:** This is used to delete the structure of a relation. It permanently deletes the records in the table.

***Syntax:***  drop table relation\_name;

***Example:*** SQL>drop table std;

**4. Rename:** It is used to modify the name of the existing database object.

***Syntax:***  RENAME table old\_relation\_name TO new\_relation\_name;

***Example:*** SQL>rename table std to std1;

**5. TRUNCATE:** This command will remove the data permanently. But structure will not be removed.

***Syntax:*** TRUNCATE TABLE <Table name>

***Example*** TRUNCATE TABLE student;

**Difference between Truncate & Delete:**-

* By using truncate command data will be removed permanently & will not get back where as by using delete command data will be removed temporally & get back by using roll back command.
* By using delete command data will be removed based on the condition where as by using truncate command there is no condition.
* Truncate is a DDL command & delete is a DML command.

**2. data manipulation language (dml):** The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

**1. INSERT 2. UPDATE 3. DELETE**

**1. insert into:**  This is used to add records into a relation. These are three type of insert into queries which are as

**a) Inserting a single record**

***Syntax:*** insert into relationname(field\_1,field\_2,.field\_n)values

(data\_1,data\_2,........data\_n);

***Example:*** SQL>insert into student(sno,sname,class,address)VALUES

(1,’Ravi’,’M.Tech’,’Palakol’);

**b) Inserting all records from another relation**

***Syntax:*** insert into relation\_name\_1 select field\_1,field\_2,field\_n

FROM relation\_name\_2 WHERE field\_x=data;

***Example:*** SQL>insert into std select sno,sname from student

where name = ‘Ramu‘;

**c) Inserting multiple records**

***Syntax:*** insert into relation\_name field\_1,field\_2,.....field\_n) values

(&data\_1,&data\_2,........&data\_n);

***Example:*** SQL>insert into student(sno,sname,class,address)

VALUES(&sno,’&sname’,’&class’,’&address’);

Enter value for sno: 101

Enter value for name: Ravi

Enter value for class: M.Tech

Enter value for name: Palakol

**2. update-set-WHERE:** This is used to update the content of a record in a relation.

***Syntax:***  SQL>update relation name set field\_name1=data,field\_name2=data,

where field\_name=data;

***Example:*** SQL>update student set sname = ‘kumar’ WHERE sno=1;

**3. delete-from**: This is used to delete all the records of a relation but it will retain the structure of that relation.

**a) delete-from**: This is used to delete all the records of relation.

***Syntax:*** SQL>delete from relation\_name;

***example:*** SQL>delete from std;

**b) delete -from-WHERE:** This is used to delete a selected record from a relation.

***Syntax:*** SQL>delete from relation\_name WHERE condition;

***Example:*** SQL>delete from student WHERE sno = 2;

**3. DRL(DATA RETRIEVAL LANGUAGE):** Retrieves data from one or more tables.

**1. select from:** To display all fields for all records.

***Syntax :*** select \* from relation\_name;

***Example :*** SQL> select \* from dept;

DEPTNO DNAME LOC

-------- ----------- ----------

10 ACCOUNTING NEW YORK

20 RESEARCH DALLAS

30 SALES CHICAGO

40 OPERATIONS BOSTON

**2. Select from:** To display a set of fields for all records of relation.

***Syntax:*** select a set of fields FROM relation\_name;

***Example:*** SQL> select deptno, dname from dept;

DEPTNO DNAME

------- ----------

10 ACCOUNTING

20 RESEARCH

30 SALES

**3. select - from -WHERE:** This query is used to display a selected set of fields for a selected set of records of a relation.

***Syntax:*** select a set of fields from relation\_name where condition;

***Example:*** SQL> select \* FROM dept WHERE deptno<=20;

DEPTNO DNAME LOC

------ ----------- ------------

10 ACCOUNTING NEW YORK

20 RESEARCH DALLAS

**4. select - from -group BY:** This query is used to group to all the records in a relation together for each and every value of a specific key(s) and then display them for a selected set of fields the relation.

***Syntax:*** select a set of fields FROM relation\_name GROUP BY field\_name;

***Example:*** SQL> SELECT EMPNO, SUM (SALARY) FROM EMP GROUP BY EMPNO;

EMPNO SUM (SALARY)

------ ----------

1 3000

2 4000

3 5000

4 6000

4 rows selected.

**5. select - from -order by:** This query is used to display a selected set of fields from a relation in an ordered manner base on some field.

***Syntax:*** select a set of fields FROM relation\_name

order by field\_name;

***Example:*** SQL> SELECT empno,ename,job FROM emp ORDER BY job;

EMPNO ENAME JOB

------ --------- --------

4 RAVI MANAGER

2 aravind Manager

1 sagar clerk

3 Laki clerk

4rows selected.

**6. join using select - from - order by:** This query is used to display a set of fields from two relations by matching a common field in them in an ordered manner based on some fields.

***Syntax:*** select a set of fields from both relations from relation\_1, relation\_2 WHERE relation\_1.field\_x = relation\_2.field\_y order by field\_z;

***Example:*** SQL>SELECT empno,ename,job,dname FROM emp,dept

WHERE emp.deptno = 20 ORDER BY job;

EMPNO ENAME JOB DNAME

------ ------ ------- ----------

7788 SCOTT ANALYST ACCOUNTING

7902 FORD ANALYST ACCOUNTING

------

7566 JONES MANAGER OPERATIONS

7566 JONES MANAGER SALES

20 rows selected.

**7. join using select - from - group by:** This query is used to display a set of fields from two relations by matching a common field in them and also group the corresponding records for each and every value of a specified key(s) while displaying.

***Syntax:*** select a set of fields from both relations FROM relation\_1,relation\_2 WHERE relation\_1.field-x=relation\_2.field-y group by field-z;

***Example:*** SQL> SELECT empno,SUM(SALARY) FROM emp,dept

WHERE emp.deptno =20 GROUP BY empno;

EMPNO SUM (SALARY)

------- --------

7369 3200

7566 11900

7788 12000

7876 4400

**8. union:** This query is used to display the combined rows of two different queries, which are having the same structure, without duplicate rows.

***Syntax:*** SELECT field\_1,field\_2,....... FROM relation\_1 WHERE (Condition) UNION SELECT field\_1,field\_2,....... FROM relation\_2 WHERE (Condition);

***Example:***

SQL> SELECT \* FROM STUDENT;

SNO SNAME

----- -------

1 kumar

2 ravi

3 ramu

SQL> SELECT \* FROM STD;

SNO SNAME

----- -------

3 ramu

5 lalitha

9 devi

1 kumar

SQL> SELECT \* FROM student UNION SELECT \* FROM std;

SNO SNAME

---- ------

1 kumar

2 ravi

3 ramu

5 lalitha

9 devi

**9. interset:** This query is used to display the common rows of two different queries, which are having the same structure, and to display a selected set of fields out of them.

***Syntax:*** select field\_1,field\_2,.. FROM relation\_1 WHERE

(Condition) INTERSECT SELECT field\_1,field\_2,.. FROM relation\_2 WHERE(Condition);

***Example :*** SQL> SELECT \* FROM student INTERSECT SELECT \* FROM std;

SNO SNAME

---- -------

1 Kumar

**10. minus:** This query is used to display all the rows in relation\_1,which are not having in the relation\_2.

***Syntax:*** select field\_1,field\_2,......FROM relation\_1

WHERE(Condition) MINUS SELECT field\_1,field\_2,.....

FROM relation\_2 WHERE(Conditon);

**SQL>** SELECT \* FROM student MINUS SELECT \* FROM std;

SNO SNAME

---- -------

2 RAVI

3 RAMU

**3. TRANSATIONAL CONTROL LANGUAGE (T.C.L):**

A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be mode permanent to the database only if they are committed a transaction begins with an executable SQL statement & ends explicitly with either role back or commit statement.

**1. COMMIT:** This command is used to end a transaction only with the help of the commit command transaction changes can be made permanent to the database.

***Syntax:*** SQL>COMMIT;

***Example:*** SQL>COMMIT;

**2. SAVE POINT**: Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter role back. Thus, save point is used in conjunction with role back.

***Syntax:*** SQL>SAVE POINT ID;

***Example:*** SQL>SAVE POINT xyz;

**3. ROLE BACK:** A role back command is used to undo the current transactions. We can role back the entire transaction so that all changes made by SQL statements are undo (or) role back a transaction to a save point so that the SQL statements after the save point are role back.

***Syntax:***  ROLE BACK( current transaction can be role back)

ROLE BACK to save point ID;

***Example:*** SQL>ROLE BACK;

SQL>ROLE BACK TO SAVE POINT xyz;

**4. DATA CONTROL LANGUAGE (D.C.L)**:

DCL provides uses with privilege commands the owner of database objects (tables), has the soul authority ollas them. The owner (data base administrators) can allow other data base uses to access the objects as per their requirement

**1. GRANT:** The GRANT command allows granting various privileges to other users and allowing them to perform operations with in their privileges

***For Example***, if a uses is granted as ‘SELECT’ privilege then he/she can only view data but cannot perform any other DML operations on the data base object GRANTED privileges can also be withdrawn by the DBA at any time

***Syntax:*** SQL>GRANT PRIVILEGES on object\_name To user\_name;

***Example***: SQL>GRANT SELECT, UPDATE on emp To hemanth;

**2. REVOKE:** To with draw the privileges that has been GRANTED to a uses, we use the REVOKE command

***Syntax:*** SQL>REVOKE PRIVILEGES ON object-name FROM user\_name;

***Example:*** SQL>REVOKE SELECT, UPDATE ON emp FROM ravi;

**1. Creation, altering and dropping of tables and inserting rows into a table (use constraints while creating tables) examples using SELECT command.**

**1. CREATE:**

**(a)create table:** This is used to create a new relation

***Syntax:*** create table relation\_name

(field\_1 data\_type(Size),field\_2 data\_type(Size), .. . );

***Example:***

SQL>create table Student (sno NUMBER(3) **PRIMARY KEY** ,sname

char(10),class char(5));

**2. ALTER:**

**(a)ALTER TABLE ...ADD...:** This is used to add some extra fields into existing relation.

***Syntax:*** ALTER TABLE relation\_name ADD(new field\_1 data\_type(size), new field\_2 data\_type(size),..);

***Example :***  SQL>ALTER TABLE std ADD(Address CHAR(10));

**(b)ALTER table...modify...:** This is used to change the width as well as data type of fields of existing relations.

***Syntax:*** alter table relation\_name modify (field\_1 newdata\_type(Size), field\_2 newdata\_type(Size),....field\_newdata\_type(Size));

***Example:*** SQL>alter table student modify(sname varchar(10),class

varchar(5));

**3. drop table:** This is used to delete the structure of a relation. It permanently deletes the records in the table.

***Syntax:***  drop table relation\_name;

***Example:*** SQL>drop table std;

**4. INSERT:**

***Syntax:*** insert into relation\_name field\_1,field\_2,.....field\_n) values

(&data\_1,&data\_2,........&data\_n);

***Example:*** SQL>insert into student(sno,sname,class,address)

VALUES(&sno,’&sname’,’&class’,’&address’);

Enter value for sno: 101

Enter value for name: SIRISHA

Enter value for class: CSE

Enter value for address: Palakol

**5. select from:** To display all fields for all records.

***Syntax :*** select \* from relation\_name;

***Example :*** SQL> select \* from student;

SNO SNAME CLASS ADDRESS

---- -------- ------ -----

101 SIRISHA CSE PALAKOL

102 DEVAKI CSE NARSAPUR

103 KUMAR CAD BHIMAVARAM

104 RAVI VLSI PALAKOL

**2. Select from:** To display a set of fields for all records of relation.

***Syntax:*** select a set of fields FROM relation\_name;

***Example:*** SQL> select sno, sname from student;

SNO SNAME

---- --------

101 SIRISHA

102 DEVAKI

103 KUMAR

104 RAVI

**3. select - from -WHERE:** This query is used to display a selected set of fields for a selected set of records of a relation.

***Syntax:*** select a set of fields from relation\_name where condition;

***Example:*** SQL> select \* FROM student WHERE class=’CSE’;

SNO SNAME CLASS ADDRESS

---- -------- ------ -------

101 SIRISHA CSE PALAKOL

102 DEVAKI CSE NARSAPUR

**There are 5 constraints available in ORACLE:**

**1. NOT NULL:** When a column is defined as NOTNULL, then that column becomes a mandatory column. It implies that a value must be entered into the column if the record is to be accepted for storage in the table.

***Syntax:***

**CREATE TABLE** Table\_Name(column\_name data\_type(*size*) **NOT NULL,** );

***Example:***

**CREATE TABLE** student (sno **NUMBER(3)NOT NULL,** name **CHAR**(**10**));

**2. UNIQUE:** The purpose of a unique key is to ensure that information in the column(s) is unique i.e. a value entered in column(s) defined in the unique constraint must not be repeated across the column(s). A table may have many unique keys.

***Syntax:***

**CREATE TABLE** Table\_Name(column\_name data\_type(*size*) **UNIQUE, ….**);

***Example:***

**CREATE TABLE** student (sno **NUMBER(3) UNIQUE,** name **CHAR**(**10**));

**3. CHECK:** Specifies a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null).

***Syntax:***

**CREATE TABLE** Table\_Name(column\_name data\_type(*size*) **CHECK(*logical expression*), ….**);

***Example:* CREATE TABLE** student (sno **NUMBER (3),** name **CHAR**(**10**),class **CHAR(5),CHECK**(class **IN**(‘CSE’,’CAD’,’VLSI’));

**4. PRIMARY KEY:** A field which is used to identify a record uniquely. A column or combination of columns can be created as primary key, which can be used as a reference from other tables. A table contains primary key is known as Master Table.

* It must uniquely identify each record in a table.
* It must contain unique values.
* It cannot be a null field.
* It cannot be multi port field.
* It should contain a minimum no. of fields necessary to be called unique.

***Syntax:***

**CREATE TABLE** Table\_Name(column\_name data\_type(*size*) **PRIMARY KEY, ….**);

***Example:***

**CREATE TABLE** faculty (fcode **NUMBER(3) PRIMARY KEY,** fname **CHAR**(**10**));

**5. FOREIGN KEY:** It is a table level constraint. We cannot add this at column level. To reference any primary key column from other table this constraint can be used. The table in which the foreign key is defined is called a **detail table**. The table that defines the primary key and is referenced by the foreign key is called the **master table**.

***Syntax:* CREATE TABLE** Table\_Name(column\_name data\_type(*size*)

**FOREIGN KEY**(column\_name) **REFERENCES** table\_name);

***Example:***

**CREATE TABLE** subject (scode **NUMBER (3) PRIMARY KEY,**

subname **CHAR**(**10**),fcode **NUMBER(3),**

**FOREIGN KEY**(fcode) **REFERENCE** faculty );

**Defining integrity constraints in the alter table command:**

***Syntax:* ALTER TABLE** Table\_Name **ADD** **PRIMARY KEY** (column\_name);

***Example:* ALTER TABLE** student **ADD** **PRIMARY KEY** (sno);

(Or)

***Syntax:* ALTER TABLE** table\_name **ADD CONSTRAINT** constraint\_name

**PRIMARY KEY**(colname)

***Example:* ALTER TABLE** student **ADD CONSTRAINT** SN **PRIMARY KEY(**SNO**)**

**Dropping integrity constraints in the alter table command:**

***Syntax:* ALTER TABLE** Table\_Name **DROP** constraint\_name;

***Example:* ALTER TABLE** student **DROP** **PRIMARY KEY**;

(or)

***Syntax:* ALTER TABLE** student **DROP CONSTRAINT** constraint\_name**;**

***Example:* ALTER TABLE** student **DROP CONSTRAINT** SN**;**

**3) Queries using Aggregate functions (COUNT, SUM, AVG, MAX and MIN), GROUP BY, HAVING and Creation and dropping of Views.**

**Aggregative operators:** In addition to simply retrieving data, we often want to perform some computation or summarization. SQL allows the use of arithmetic expressions. We now consider a powerful class of constructs for computing aggregate values such as MIN and SUM.

**1. Count:** COUNT following by a column name returns the count of tuple in that column. If DISTINCT keyword is used then it will return only the count of unique tuple in the column. Otherwise, it will return count of all the tuples (including duplicates) count (\*) indicates all the tuples of the column.

***Syntax:*** COUNT (Column name)

***Example:*** SELECT COUNT (Sal) FROM emp;

**2. SUM:** SUM followed by a column name returns the sum of all the values in that column.

***Syntax:*** SUM (Column name)

***Example:*** SELECT SUM (Sal) From emp;

**3. AVG:** AVG followed by a column name returns the average value of that column values.

***Syntax:*** AVG (n1,n2..)

***Example:*** Select AVG(10, 15, 30) FROM DUAL;

**4. MAX:** MAX followed by a column name returns the maximum value of that column.

***Syntax:*** MAX (Column name)

***Example:*** SELECT MAX (Sal) FROM emp;

SQL> select deptno,max(sal) from emp group by deptno;

DEPTNO MAX(SAL)

------ --------

10 5000

20 3000

30 2850

SQL> select deptno,max(sal) from emp group by deptno having max(sal)<3000;

DEPTNO MAX(SAL)

----- --------

30 2850

**5. MIN:** MIN followed by column name returns the minimum value of that column.

***Syntax:*** MIN (Column name)

***Example:*** SELECT MIN (Sal) FROM emp;

SQL>select deptno,min(sal) from emp group by deptno having min(sal)>1000;

DEPTNO MIN(SAL)

----- --------

10 1300

**VIEW:** In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

A view is a virtual table, which consists of a set of columns from one or more tables. It is similar to a table but it doest not store in the database. View is a query stored as an object.

***Syntax:*** create view view\_name AS SELECT set of fields FROM relation\_name WHERE (Condition)

***1. Example:***

SQL>create view employee as select empno,ename,job from emp

where job = ‘clerk’;

view created.

sql> select \* from employee;

empno ename job

---- ------ -------

7369 smith clerk

7876 adams clerk

7900 james clerk

7934 miller clerk

***2.Example:***

CREATE VIEW [Current Product List] AS  
SELECT ProductID,ProductName  
FROM Products  
WHERE Discontinued=No

**drop view**:This query is used to delete a view , which has been already created.

***Syntax:***  drop VIEW view\_name;

***Example :*** SQL> DROP VIEW EMPLOYEE;

View dropped

**4. Queries using Conversion functions (to\_char, to\_number and to\_date), string functions (Concatenation, lpad, rpad, ltrim, rtrim, lower, upper, initcap, length, substr and instr), date functions (Sysdate, next\_day, add\_months, last\_day, months\_between, least, greatest, trunc, round, to\_char, to\_date)**

**1. Conversion functions:**

**To\_char:** TO\_CHAR (number) converts n to a value of VARCHAR2 data type, using the optional number format fmt. The value n can be of type NUMBER, BINARY\_FLOAT, or BINARY\_DOUBLE.

SQL>select to\_char(65,'RN')from dual;

LXV

**To\_number :** TO\_NUMBER converts expr to a value of NUMBER data type.

SQL> Select to\_number('1234.64') from Dual;  
1234.64

**To\_date:** TO\_DATE converts char of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of DATE data type.

SQL>SELECT TO\_DATE('January 15, 1989, 11:00 A.M.')FROM DUAL;

TO\_DATE

---------

15-JAN-89

**2. String functions:**

**Concat:** CONCAT returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes

SQL>SELECT CONCAT(‘ORACLE’,’CORPORATION’)FROM DUAL;

ORACLECORPORATION

**Lpad:** LPAD returns expr1, left-padded to length n characters with the sequence of characters in expr2.

SQL>SELECT LPAD(‘ORACLE’,15,’\*’)FROM DUAL;

\*\*\*\*\*\*\*\*\*ORACLE

**Rpad:** RPAD returns expr1, right-padded to length n characters with expr2, replicated as many times as necessary.

SQL>SELECT RPAD (‘ORACLE’,15,’\*’)FROM DUAL;

ORACLE\*\*\*\*\*\*\*\*\*

**Ltrim:** Returns a character expression after removing leading blanks.

SQL>SELECT LTRIM(‘SSMITHSS’,’S’)FROM DUAL;

MITHSS

**Rtrim:** Returns a character string after truncating all trailing blanks

SQL>SELECT RTRIM(‘SSMITHSS’,’S’)FROM DUAL;

SSMITH

**Lower:** Returns a character expression after converting uppercase character data to lowercase.

SQL>SELECT LOWER(‘DBMS’)FROM DUAL;

dbms

**Upper:** Returns a character expression with lowercase character data converted to uppercase

SQL>SELECT UPPER(‘dbms’)FROM DUAL;

DBMS

**Length:** Returns the number of characters, rather than the number of bytes, of the given string expression, excluding trailing blanks.

SQL>SELECT LENGTH(‘DATABASE’)FROM DUAL;

8

**Substr:** Returns part of a character, binary, text, or image expression.

SQL>SELECT SUBSTR(‘ABCDEFGHIJ’3,4)FROM DUAL;

CDEF

**Instr:** The INSTR functions search string for substring. The function returns an integer indicating the position of the character in string that is the first character of this occurrence.

SQL>SELECT INSTR('CORPORATE FLOOR','OR',3,2)FROM DUAL;

14

**3. Date functions:**

**Sysdate:**

SQL>SELECT SYSDATE FROM DUAL;

29-DEC-08

**next\_day:**

SQL>SELECT NEXT\_DAY(SYSDATE,’WED’)FROM DUAL;

05-JAN-09

**add\_months:**

SQL>SELECT ADD\_MONTHS(SYSDATE,2)FROM DUAL;

28-FEB-09

**last\_day:**

SQL>SELECT LAST\_DAY(SYSDATE)FROM DUAL;

31-DEC-08

**months\_between:**

SQL>SELECT MONTHS\_BETWEEN(SYSDATE,HIREDATE)FROM EMP;

4

**Least:**

SQL>SELECT LEAST('10-JAN-07','12-OCT-07')FROM DUAL;

10-JAN-07

**Greatest:**

SQL>SELECT GREATEST('10-JAN-07','12-OCT-07')FROM DUAL;

10-JAN-07

**Trunc:**

SQL>SELECT TRUNC(SYSDATE,'DAY')FROM DUAL;

28-DEC-08

**Round:**

SQL>SELECT ROUND(SYSDATE,'DAY')FROM DUAL;

28-DEC-08

**to\_char:**

SQL> select to\_char(sysdate, "dd\mm\yy") from dual;

24-mar-05.

**to\_date:**

SQL> select to\_date(sysdate, "dd\mm\yy") from dual;

24-mar-05.

# LIST OF EXPERIMENTS

1. Creation of Database and Familiarization of DML commands
2. Creating Relationship between Databases and Implementation of ORDERBY, GROUPBY and HAVING clauses.
3. Program to find sum of ‘n’ numbers
4. Built in functions in SQL
5. Program to check a String is Palindrome.
6. PL/SQL program to display the salary of all Employees and find the total salary paid to employee using cursor loop.
7. Program to create table pass and fail.
8. Program to find factorial using recursive function.
9. Program to display basic incremented Salary.
10. Program using Trigger.

**Experiment No:01**

Creation of Database and Familiarization of DML commands

**Aim**

To create a database and familiarize the DML commands.

Create the following table and insert the data and find the result for question given below using SQL.

EMPLOYEE

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| EID | LNAME | FNAME | MNAME | JOBID | DOJ | SALARY | DEPID |
| 1000 | John | Smith | b | 111 | 12-jun-12 | 25000 | 1 |
| 1001 | Larry | Becker | T | 222 | 12-jun-12 | 30000 | 2 |
| 1002 | Paul | Mathew | p | 333 | 10-mar-12 | 20000 | 3 |
| 1003 | George | Thomas | v | 222 | 13-jun-11 | 35000 | 1 |
| 1004 | Jacob | Mathew | v | 111 | 10-jun-11 | 40000 | 2 |

**Algorithm**

Step 1: Create table Employee with given fields.

CREATE TABLE Tablename

(Columnname 1 datatype Constraints,

Columnname2datatype Constraints,

……………………………………………………….);

Step 2: Insert given data into the table.

INSERT INTO Tablename

VALUES (……………..);

Step 3: List all the employees details.

Select \* FROM Tablename;

Step 4: Alter the structure of the table by adding a field called commission and insert values into the added field.

ALTER TABLE Tablename

ADD Newfieldnamedatatype;

Step 5: Give 10% increase in the salary to the employee of department 1.

UPDATE Tablename

SET SALARY = SALARY + (10/100)\*SALARY

WHERE condition;

Step 6: Delete the details of Employee working in department 3.

DELETE FROM Tablename

WHERE condition;

Step 7: CREATE a view of Employee table with attributes EID,LNAME,FNAME,JOBID and DEPID

CREATE VIEW Viewname

SELECT Fieldname FROM TAblename;

Step 8: List out the department wise maximum salary, minimum salary, average salary of Employees.

SELECT DEPID,MAX(SALARY),MIN(SALARY),AVG(SALARY) FROM Tablename

GROUP BY Fieldname;

Step 9: List out the employee id ,Lname, salary in descending order based on salary.

SELECT Fieldname

FROM Tablename

ORDERBY DESC;

STEP 10: How many employee are working in each department in the organization.

SELECT DEPID,count(\*)

FROM Tablename

GROUP BY Fieldname;

**Experiment No: 2**

Creating Relationship Between Databases and Implementation of orderby,groupby and having clauses

**Aim**

To create a database, insert the data and find the result for the questions.

Create the following tables and insert the data and find the result for the questions given below using SQL.

CUSTOMER

cid- primary key, should start with letter c

name- not null

userid- unique

password- not null

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CID | NAME | DATEOFREG | USERID | PASSWORD |
| C1000 | Madhav | 22-FEB-12 | mmmmm | mpmpm |
| C1001 | Bobby | 13-MAR-12 | bbbbb | bpbpb |
| C1002 | Catherine | 16-APR-12 | ccccc | cpcpc |
| C1003 | Kale | 21-MAY-12 | kkkkk | kpkpk |
| C1004 | Vijay | 10-JUN-12 | vvvvv | vpvpv |

ITEM

Itemcode – primary key

|  |  |  |  |
| --- | --- | --- | --- |
| ITEMCODE | ITEMNAME | UNITPRICE | QUANTITY |
| I100 | pen | 10 |  |
| I101 | pencil | 6 |  |
| I102 | eraser | 4 |  |
| I103 | pen | 17 |  |
| I104 | book | 20 |  |
| I105 | book | 40 |  |

CUSTOMERPURCHASE

custid should refer cid in CUSTOMER table

Itemid should refer itemcode in ITEM table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CUSTID | ITEMID | QTY | BILLNO | NETPRICE |
| C1000 | I100 | 10 | b1 | 100 |
| C1001 | I101 | 5 | b2 | 30 |
| C1002 | I102 | 20 | b3 | 80 |
| C1003 | I103 | 10 | b4 | 170 |
| C1004 | I104 | 20 | b5 | 400 |
| C1003 | I105 | 10 | b6 | 50 |
| C1001 | I101 | 20 | b7 | 120 |
| C1000 | I100 | 30 | b8 | 300 |

Step 1: List the number of items in the item table based on the item name.

SELECT count(\*) FROM fieldname

GROUP BY fieldname;

Step 2: Get the customerid and name of the customers who registerdbefre 01-JUNE-2012.

SELECT fieldname

FROM tablename

WHERE condition;

Step 3 : List the details of customers where name start with ‘c’ and has ‘t’ as the third character.

SELECT \* FROM Tablename

WHERE condition;

Step 4: List the different items in the item table.

SELECT Fieldname

FROM Tablename;

Step 5: Display the customerid,name and itemid of the customer who have purchased same items.

SELECT fieldname

FROM tablename1 JOIN tablename2

WHERE condition;

Step 6: Get the customerid and name of the customer who have purchased more than one items.

SELECT count(Column name),columnname1,…

FROM table\_name 1,….WHERE condition

GROUP BY column name having count(columnname);

Step 7: Display the customerid,name and total purchase amount of customers who have purchase items.

SELECT column\_name 1,….sum(column\_name)

FROM tablename 1,…WHERE condition

GROUP BY column\_name;

Step 8: List the name and the unit price of the items purchased by customers.

SELECT fieldnames

GROUP BY column\_name;

Step 9: Get the details of item which has minimum unit price.

SELECT \* from tablename WHERE fieldname==(SELECT min(column\_name) from Tablename);

Step 10: Display the id of customer whose total purchase amount is more than other customer.

SELECT fieldnames , sum(columnname) from Tablenames

GROUP BY columnname having SUM(column\_name)=(SELECT MAX(SUM(columnname))

FROM Tablename GROUP BY Column\_name;

**Experiment No: 3**

PROGRAM TO FIND SUM OF ‘N’ NUMBERS

**Aim:**

Write a PL/SQL program to find sum of ‘n’ numbers.

**ALGORITHM:**

1. Start
2. Declare the variables.
3. Initialize the result, res =0.
4. Read the limit, n.
5. If n>0 , repeat steps from 6 to 7.
6. res= res + n
7. n=n-1
8. Print result, res.

**Experiment No: 4**

BUILT IN FUNCTIONS IN SQL

**Aim:**

Write the suitable SQL queries for the following:

1. Get the system date.
2. Get the absolute value of -5.
3. Round 9.5 upwards.
4. Round 10.6 downwards.
5. Find the remainder when 15/6.
6. Find the value of 6^4.
7. Find the square root of 4096.
8. Convert the string ‘INDIA’ to lowercase.
9. Convert the string ‘kerala’ to uppercase.
10. Remove the leading string ‘K’ from string ‘KKochi’.
11. Remove the trailing m from ‘Malayalam’.
12. Remove the leading and trailing m from ‘Malayalam’.
13. Convert the string ‘abcdeabcde’ to ‘1234512345’.
14. Display ‘is in’ from string ‘Doctor is in’.
15. Find the length of string ‘Welcome to Kerala’.
16. Join the words ‘Happy ‘ and ‘Birthday’ into a single string.
17. Use the appropriate function to limit the number & decimal places to 1 when the input is 198.2345
18. Convert the string “good morning” to “Good Morning”.
19. Given current date and day as input, find the date in next week on same day .

**Experiment No: 05**

**Aim:**

Program to check a String is Palindrome.

**Algorithm:**

Step1: Start

Step2: Declare the variables and assign the input string to one of the variables.

Step3: Find the length of the string

Step4: Using the for loop reverse the string

Step5: Check whether the reverse string is equal to input string

Step6: If yes, print Palindrome. Else, not Palindrome.

Step7: Stop.

**Experiment No:06**

**Aim:**

PL/SQL program to display the salary of all Employees and find the total salary paid to employee using cursor loop.

**Algorithm:**

Step1:Start

Step2: Create cursor with salary of employees, from employees table.

Step3: Open cursor

Step4: Declare variables – total\_sal

Step5: Loop

* 1. Fetch a row from cursor
  2. total\_sal = total\_sal + sal

Step6: End Loop

Step7: Output total\_sal

Step8: Stop

**EXPERIMENT NO:07**

**AIM**

Program to create table pass and fail.

**ALGORITHM**

Step1: Start the program.

Step 2: Create the table student tablewith the entity and insert data in to it.

Step 3: Create two other table pass fail.

Step 4: If total > = 80 and m1, m2 > = 35 then go to step 4.1 else go to step 5.

Step 4.1: Insert details of student in to pass.

Step 5: Stop.

**EXPERIMENT NO:08**

**AIM**

Program to find factorial using recursive function.

**ALGORITHM**

**Algorithm for recursive function:**Step 1: Accept the value of n in the function FACT through parameters.  
Step 2: If n=1 return 1otherwise return n fact(n-1).  
Step 3: Stop the program.

**Algorithm for main function :**Step 1: Get the value of n, factorial to be found.  
Step 2: Call the function fact with parameters.  
Step 3: Display the factorial of a given number.  
Step 4: Stop the program.

**EXPERIMENT NO:09**

**AIM**

Program to display basic incremented Salary.

**ALGORITHM**

Step 1: Start

Step 2: Procedure creation.

Step 3: Create tables.

Step 4: Select salary of employee from employe table and store in basic salary.

Step 5: If table found,insert emp\_id,basic\_sal,date and inc\_salary to the table else print not found.

**Main program**

Step 1: Input emp\_id, inc\_salary, abcd=0.

Step 2: Call procedure with arguments emp\_id, inc\_salary.

Step 3: Stop.

**EXPERIMENT NO :10**

**AIM**

Program using Trigger.

**ALGORITHM**

Step 1: Start the program.

Step 2: Create the table with its essential attributes.

Step 3: Insert attribute values into the table.

Step 4: Create the trigger for a particular table .

Step 5: Specify when the trigger is to be fired - before or after .

Step 6: Specify DML statement that invokes the trigger - UPDATE, DELETE, or INSERT .

Step 7: Specify the type of triger whether row-level trigger or not.

Step 8: Condition to filter rows.

Step 9: PL/SQL block that is to be executed when trigger is fired.

Step 10: Modify the specified table to fire the trigger.

Step 11: Display the trigger message for the particular kind of modification.

Step 12: Stop the program