

LAB PRACTICAL JOURNAL OF

Database Management System

BE: III Year

**Department of Computer Science & Engineering**

**Name of the Student: Dhaval Maniyar**

**Branch & section: CS-2**

**Roll No.: 0827CS181067**

**Year: 2020-21**

### ACROPOLIS INSTITUTE OF TECHNOLOGY & RESEARCH, INDORE

**Department of Computer Science & Engineering.**

**Certificate**

## This is to certify that the experimental work entered in this journal as per the BE III year syllabus prescribed by the RGPV was done by Mr Dhaval Maniyar studying in BE III year V semester in the Laboratory of this institute during the academic year 2020-21.

Signature of Head Signature of the Faculty

### ACROPOLIS INSTITUTE OF TECHNOLOGY & RESEARCH, INDORE

**Department of Computer Science & Engineering**

**GENERAL INSTRUCTIONS FOR LABORATORY CLASSES**

**DO’S**

* Without Prior permission do not enter into the Laboratory.
* While entering into the LAB students should wear their ID cards.
* The Students should come with proper uniform.
* Students should sign in the LOGIN REGISTER before entering into the laboratory.
* Students should come with observation and record note book to the laboratory.
* Students should maintain silence inside the laboratory.
* After completing the laboratory exercise, make sure to shutdown the system properly.

**DONT’S**

* Students bringing the bags inside the laboratory.
* Students using the computers in an improper way.
* Students scribbling on the desk and mishandling the chairs.
* Students using mobile phones inside the laboratory.
* Students making noise inside the laboratory.

### ACROPOLIS INSTITUTE OF TECHNOLOGY & RESEARCH, INDORE

**Department of Computer Science & Engineering**

Syllabus

CS-502-DBMS Lab

**Unit I**

DBMS Concepts and architecture Introduction, Database approach v/s Traditional file accessing approach, Advantages, of database systems, Data models, Schemas and instances, Data independence, Data Base Language and interfaces, Overall Database Structure, Functions of DBA and designer, ER data model: Entitles and attributes, Entity types, Defining the E-R diagram, Concept of Generalization, Aggregation and Specialization. Transforming ER diagram into the tables. Various other data models object oriented data Model, Network data model, and Relational data model, Comparison between the three types of models.

**Unit II**

Relational Data models: Domains, Tuples, Attributes, Relations, Characteristics of relations, Keys, Key attributes of relation, Relational database, Schemas, Integrity constraints. Referential integrity, Intension and Extension, Relational Query languages: SQL-DDL, DML, integrity constraints, Complex queries, various joins, indexing, triggers, assertions, Relational algebra and relational calculus, Relational algebra operations like select, Project, Join, Division, outer union. Types of relational calculus i.e. Tuple oriented and domain oriented relational calculus and its operations.

**Unit III**

Data Base Design: Introduction to normalization, Normal forms, Functional dependency, Decomposition, Dependency preservation and lossless join, problems with null valued and dangling Tuples, multivalued dependencies. Query Optimization: Introduction, steps of optimization, various algorithms to implement select, project and join operations of relational algebra, optimization methods: heuristic based, cost estimation based.

**Unit IV**

Transaction Processing Concepts: - Transaction System, Testing of Serilizability, Serializability of schedules, conflict & view serializable schedule, recoverability, Recovery from transaction failures. Log based recovery. Checkpoints deadlock handling. Concurrency Control Techniques: - Concurrency Control, locking Techniques for concurrency control, time stamping protocols for concurrency control, validation based protocol, multiple granularity. Multi version schemes, Recovery with concurrent transaction. Introduction to Distributed databases, data mining, data warehousing, Object Technology and DBMS, Comparative study of OODBMS Vs DBMS. Temporal, Deductive, Multimedia, Web & Mobile database.

**Unit V**

Study of Relational Database Management Systems through Oracle/Postgres SQL/MySQL: Architecture, physical files, memory structures, background process. Concept of table spaces, segments, extents and block. Dedicated server, multi threaded server. Distributed database, database links, and snapshot. Data dictionary, dynamic performance view. Security, role management, privilege management, profiles, invoker defined security model. SQL queries, Data extraction from single, multiple tables equi-join, non equi-join, self-join, outer join. Usage of like, any, all, exists, in Special operators. Hierarchical queries, inline queries, flashback queries. Introduction of ANSI SQL, anonymous block, nested anonymous block, branching and looping constructs in ANSI SQL. Cursor management: nested and parameterized cursors, Oracle exception handling mechanism. Stored procedures, in, out, in out type parameters, usage of parameters in procedures. User defined functions their limitations. Triggers, mutating errors, instead of triggers.

**HARDWARE REQUIREMENTS**

* + INTEL PENTIUM 915 GV
  + 80GB HDD
  + 512MB DDR
  + Processors - 2.0 GHz or Higher
  + RAM - 256 MB or Higher
  + Hard Disk - 20 GB or Higher

**SOFTWARE REQUIREMENTS**

* + Operating System Windows XP
  + DBMS oracle 11g

**RATIONALE**

The purpose of this subject is to cover the underlying concepts and techniques used in creating a Data Base System. These techniques can be used in Software Developments.

**PREREQUISITE**

The students should have a general idea about data base concept, data models and sql statements

# INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Name of Experiment** | **Page No.** | **Remarks** |
| 1. | To Study the DBMS, RDBMS and introduction of SQL, an introduction to different data types used in SQL. |  |  |
| 2. | To exercise and implement Data Definition Language Commands and Table Creation Commands. |  |  |
| 3. | To exercise and implement Data Manipulation Language Commands (Insert, Update and Delete Commands). |  |  |
| 4. | SQL Constraints |  |  |
| 5. | To exercise and implement aggregate and scalar functions in SQL |  |  |
| 6. | To exercise and implement Wildcard characters in SQL |  |  |
| 7. | To exercise and implement Order By clause |  |  |
| 8. | To exercise and implement Group By clause |  |  |
| 9. | To exercise and implement SQL Joins |  |  |
| 10. | To exercise and implement nested Queries/ Subqueries. |  |  |
| 11. | PL/SQL |  |  |
|  | Tutorial Sheet 1 |  |  |
|  | Tutorial Sheet 2 |  |  |
|  | Tutorial Sheet 3 |  |  |

**EXPERIMENT – 1**

**OBJECTIVE:**

To Study the DBMS, RDBMS and Introduction of SQL, an exercise on different data types used in SQL.

**CONCEPT:**

**Definition of DBMS:**

* **database** is an organized collection of [data.](http://en.wikipedia.org/wiki/Data_%5C%5C(computing%5C%5C)) The data are typically organized to model relevant aspects of reality in a way that supports processes requiring this information. For example, modeling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

**Database management systems** (**DBMS**s) are specially designed software applications thatinteract with the user, other applications, and the database itself to capture and analyze data. querying,

**Feature s of DBMS:**

Formally, "database" refers to the data themselves and supporting data structures. Databases are created to operate large quantities of information by inputting, storing, retrieving and managing that information. Databases are set up so that one set of software programs provides all users with access to all the data.

A "database management system" (DBMS) is a suite of computer software providing the interface between users and a database or databases. Because they are so closely related, the term "database" when used casually often refers to both a DBMS and the data it manipulates.

Outside the world of professional [information technology,](http://en.wikipedia.org/wiki/Information_technology) the term *DATABASE* is sometimes used casually to refer to any collection of data (perhaps a [spreadsheet,](http://en.wikipedia.org/wiki/Spreadsheet) maybe even a card index). This article is concerned only with databases where the size and usage requirements necessitate use of a database management system.

The interactions catered for by most existing DBMSs fall into four main groups:

* **Data definition –** Defining new data structures for a database, removing data structuresfrom the database, modifying the structure of existing data.
* **Update –** Inserting, modifying, and deleting data.
* **Retrieval –** Obtaining information either for end-user queries and reports or for processingby applications.
* **Administration –** Registering and monitoring users, enforcing data security, monitoringperformance, maintaining data integrity, dealing with concurrency control, and recovering information if the system fails.

A DBMS is responsible for maintaining the integrity and security of stored data, and for recovering information if the system fails.

Both a database and its DBMS conform to the principles of a particular [database model.](http://en.wikipedia.org/wiki/Database_model)

"Database system" refers collectively to the database model, database management system, and database.

Physically, database [servers](http://en.wikipedia.org/wiki/Server_%5C%5C(computing%5C%5C)) are dedicated computers that hold the actual databases and run only the DBMS and related software. Database servers are usually multiprocessor computers, with generous memory and [RAID](http://en.wikipedia.org/wiki/Redundant_array_of_independent_disks) disk arrays used for stable storage. RAID is used for recovery of data if any of the disks fail. Hardware database accelerators, connected to one or more servers via a high-speed channel, are also used in large volume transaction processing environments. DBMSs are found at the heart of most [database applications.](http://en.wikipedia.org/wiki/Database_application) DBMSs may be built around a custom [multitasking](http://en.wikipedia.org/wiki/Computer_multitasking) [kernel](http://en.wikipedia.org/wiki/Kernel_%5C%5C(computing%5C%5C)) with built-in [networking](http://en.wikipedia.org/wiki/Computer_network) support, but modern DBMSs typically rely on a standard [operating system](http://en.wikipedia.org/wiki/Operating_system) to provide these functions.

Databases and DBMSs can be categorized according to the database model(s) that they support (such as relational or XML), the type(s) of computer they run on (from a server cluster to a mobile phone), the [query language(s)](http://en.wikipedia.org/wiki/Query_language) used to access the database (such as SQL or [XQuery),](http://en.wikipedia.org/wiki/XQuery) and their internal engineering, which affects performance, [scalability,](http://en.wikipedia.org/wiki/Scalability) resilience, and security.

**Advantages of the DBMS:**

The DBMS serves as the intermediary between the user and the database. The database structure itself is stored as a collection of files, and the only way to access the data in those files is through the DBMS. The DBMS receives all application requests and translates them into the complex **operations required to fulfill those requests. The DBMS hides much of the database’s internal** complexity from the application programs and users.

***The different advantages of DBMS are as follows.***

1. **Improved data sharing**

The DBMS helps create an environment in which end users have better access to more and better-managed data. Such access makes it possible for end users to respond quickly to changes in their environment.

**2. Improved data security**

The more users access the data, the greater the risks of data security breaches. Corporations invest considerable amounts of time, effort, and money to ensure that corporate data are used properly. A DBMS provides a framework for better enforcement of data privacy and security policies.

**3. Better data integration**

Wider access to well-**managed data promotes an integrated view of the organization’s operations** and a clearer view of the big picture. It becomes much easier to see how actions in one segment of the company affect other segments.

**4. Minimized data inconsistency.**

Data inconsistency exists when different versions of the same data appear in different places. For example, data inconsistency exists when a company’s sales department stores a sales representative’s name as “Bill Brown” and the company’s personnel department stores that same person’s name as “William G. Brown,” or when the company’s regional sales office shows the price of a product as $45.95 and its national sales office shows the same product’s price as $43.95. The probability of data inconsistency is greatly reduced in a properly designed database.

**5. Improved data access.**

The DBMS makes it possible to produce quick answers to ad hoc queries. From a database perspective, a query is a specific request issued to the DBMS for data manipulation**—**for example, to read or update the data. Simply put, a query is a question, and an ad hoc query is a spur-of-the-moment question. The DBMS sends back an answer (called the query result set) to the application. For example, end users, when dealing with large amounts of sales data, might want quick answers to questions (ad hoc queries) such as:

* What was the dollar volume of sales by product during the past six months?
* What is the sales bonus figure for each of our salespeople during the past three months?
* How many of our customers have credit balances of $3,000 or more?

**6. Improved decision making**

Better-managed data and improved data access make it possible to generate better-quality information, on which better decisions are based. The quality of the information generated depends on the quality of the underlying data. Data quality is a comprehensive approach to promoting the accuracy, validity, and timeliness of the data. While the DBMS does not guarantee data quality, it provides a framework to facilitate data quality initiatives.

**7. Increased end-user productivity**

The availability of data, combined with the tools that transform data into usable information, empowers end users to make quick, informed decisions that can make the difference between success and failure in the global economy.

**Disadvantages of Database:**

Although the database system yields considerable advantages over previous data management approaches, database systems do carry significant disadvantages. For example:

**1. Increased costs.**

Database systems require sophisticated hardware and software and highly skilled personnel. The cost of maintaining the hardware, software, and personnel required to operate and manage a database system can be substantial. Training, licensing, and regulation compliance costs are often overlooked when database systems are implemented.

**2. Management complexity.**

Database systems interface with many different technologies and have a significant impact on a Company’s resources and culture. The changes introduced by the adoption of a database system must be properly managed to ensure that they help advance the company’s objectives. Given the fact that database systems hold crucial company data that are accessed from multiple sources, security issues must be assessed constantly.

**3. Maintaining currency.**

To maximize the efficiency of the database system, you must keep your system current. Therefore, you must perform frequent updates and apply the latest patches and security measures to all components. Because database technology advances rapidly, personnel training costs tend to be significant. Given the heavy investment in technology and personnel training, companies might be reluctant to change database vendors. As a consequence, vendors are less likely to offer pricing point advantages to existing customers, and those customers might be limited in their choice of database system components.

**4. Frequent upgrade/replacement cycles.**

DBMS vendors frequently upgrade their products by adding new functionality. Such new features often come bundled in new upgrade versions of the software. Some of these versions require hardware upgrades. Not only do the upgrades themselves cost money, but it also costs money to train database users and administrators to properly use and manage the new features.

**Definition of RDBMS**

A **relational database management system** (**RDBMS**) is a [database management](http://en.wikipedia.org/wiki/Database_management_system) [system](http://en.wikipedia.org/wiki/Database_management_system) (DBMS) that is based on the [relational model](http://en.wikipedia.org/wiki/Relational_model) as introduced by [E. F. Codd,](http://en.wikipedia.org/wiki/Edgar_F._Codd) of IBM's [San](http://en.wikipedia.org/wiki/IBM_Almaden_Research_Center) [Jose Research Laboratory.](http://en.wikipedia.org/wiki/IBM_Almaden_Research_Center) Many popular databases currently in use are based on the [relational](http://en.wikipedia.org/wiki/Relational_database) [database](http://en.wikipedia.org/wiki/Relational_database) model.

RDBMSs have become a predominant choice for the storage of information in new databases used for financial records, manufacturing and logistical information, personnel data, and much more since the 1980s. Relational databases have often replaced legacy [hierarchical](http://en.wikipedia.org/wiki/Hierarchical_database) [databases](http://en.wikipedia.org/wiki/Hierarchical_database) and [network databases](http://en.wikipedia.org/wiki/Network_database) because they are easier to understand and use. However, relational databases have been challenged by [object databases,](http://en.wikipedia.org/wiki/Object_database) which were introduced in an attempt to address the [object-relational impedance mismatch](http://en.wikipedia.org/wiki/Object-relational_impedance_mismatch) in relational database, and [XML](http://en.wikipedia.org/wiki/XML_database) [databases](http://en.wikipedia.org/wiki/XML_database)

RDBMS stands for Relational Database Management System. RDBMS data is structured in database tables, fields and records. Each RDBMS table consists of database table rows. Each database table row consists of one or more database table fields.

RDBMS store the data into collection of tables, which might be related by common fields (database table columns). RDBMS also provide relational operators to manipulate the data stored into the database tables. Most RDBMS use [SQL](http://www.sql-tutorial.net/) as database query language.

Edgar Codd introduced the relational database model. Many modern DBMS do not conform to the **Codd’s definition of a R**DBMS, but nonetheless they are still considered to be RDBMS.

The most popular RDBMS are MS SQL Server, DB2, Oracle and MySQL.

**Advantages of Relational approach**

The popularity of the relational database approach has been apart from access of availability of a large variety of products also because it has certain inherent advantages.

1. Ease of use: The revision of any information as tables consisting 0f rows and columns is quite natural and therefore even first time users find it attractive.
2. Flexibility: Different tables from which information has to be linked and extracted can be easily manipulated by operators such as project and join to give information in the form in which it is desired.
3. Precision: The usage of relational algebra and relational calculus in the manipulation of he relations between the tables ensures that there is no ambiguity, which may otherwise arise in establishing the linkages in a complicated network type database.
4. Security: Security control and authorization can also be implemented more easily by moving sensitive attributes in a given table into a separate relation with its own authorization controls. If authorization requirement permits, a particular attribute could be joined back with others to enable full information retrieval.
5. Data Independence: Data independence is achieved more easily with normalization structure used in a relational database than in the more complicated tree or network structure.
6. Data Manipulation Language: The possibility of responding to ad-hoc query by means of a language based on relational algebra and relational calculus is easy in the relational database approach. For data organized in other structure the query language either becomes complex or extremely limited in its capabilities.

**Disadvantages of Relational Approach**

One should not get carried away into believing that there can be no alternative to the RDBMS. This is not so. A major constraint and therefore disadvantage in the use of relational database system is machine performance. If the number of tables between which relationships to be established are large and the tables themselves are voluminous, the performance in responding to queries is definitely degraded. It must be appreciated that the simplicity in the relational database approach arises in the logical view. With an interactive system, for example an operation like join would depend upon the physical storage also. It is, therefore common in relational databases to tune the databases and in such a case the physical data layout would be chosen so as to give good performance in the most frequently run operations. It therefore would naturally result in the fact that the lays frequently run operations would tend to become even more shared.

**SQL: Structured Query Language.**

SQL (Structured Query Language) is a database computer language designed for the retrieval and management of data in relational database management systems (RDBMS), database schema creation and modification, and database object access control management.SQL is a standard supported by all the popular relational database management systems in the market place. The basis data structure in RDBMS is a table. SQL provides you the features to define tables, define constraints on tables, query for data in table, and change the data in table by adding, modifying, and removing data. SQL also supports grouping of data in multiple rows, combining tables and other features. All these put together, SQL is a high-level query language standard to access and alter data in RDBMS.

**History of SQL:**

The first version of SQL was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s. This version, initially called SEQUEL, was designed to manipulate and retrieve

data stored in IBM's original relational database product, System R. IBM patented their version of SQL in 1985, while the SQL language was not formally standardized until 1986, by the American National Standards Institute (ANSI) as SQL-86. Subsequent versions of the SQL standard have been released by ANSI and as International Organization for Standardization (ISO) standards.

Originally designed as a declarative query and data manipulation language, variations of SQL have been created by SQL database management system (DBMS) vendors that add procedural constructs, Control-of-flow statements, user-defined data types, and various other language extensions. With the release of the SQL: 1999 standard, many such extensions were formally adopted as part of the SQL language via the SQL Persistent Stored Modules (SQL/PSM) portion of the standard.

Querying can be used to: To retrieve existing data from database.

* Get all data from the table
* Get selected columns from the table.
* Get selected rows from the table.
* Get selected columns of selected rows from the table.
* Get computed columns using char, number, data functions, general functions, and aggregating functions.
* Get data in multiple rows grouped on an aggregating function applied on one or more columns.
* Select specific aggregating data on multiple rows using having clause. Apply set operations like Union and Intersection on data sets of the same cardinality and type.
* Get data from multiple tables using Cartesian product, equality join, un-equal join, and outer join?
* Create views on physical data.

**Various Data Types:**

**1. Character Data types:**

* Char **–** fixed length character string that can varies between 1-2000 bytes

Varchar / Varchar2 **–** variable length character string, size ranges from 1-4000 bytes. it saves the disk space(only length of the entered value will be assigned as the size of column)

* Long - variable length character string, maximum size is 2 GB

1. **Number Data types: Can store +ve,-ve,zero,fixed point, floating point with 38 precision.**

* Number **–** {p=38, s=0}
* Number(p) - fixed point
* Number(p,s) **–**floating point (p=1 to 38,s= -84 to 127)

**3. Date Data type: used to store date and time in the table.**

* DB uses its own format of storing in fixed length of 7 bytes for century, date, month, year, hour, minutes, and seconds.
* **Default data type is “dd**- mon-**yy”.**

1. **Raw Data type: used to store byte oriented data like binary data and byte string.**
2. **Other :**
   * CLOB **–** stores character object with single byte character.
   * BLOB **–** stores large binary objects such as graphics, video, sounds.
   * BFILE **– stores file pointers to the LOB’s.**

**EXPERIMENT-2**

**OBJECTIVE:**

**To exercise and implements Data Definition Language Commands and Table Creation Commands.**

**CONCEPT:**

* DDL is used for specifying the database schema. Let’s take SQL for instance to categorize the statements that comes under DDL.
* To create the database instance – [**CREATE**](https://beginnersbook.com/2014/05/sql-create-database-statement/)
* To alter the structure of database – **ALTER**
* To drop database instances – [**DROP**](https://beginnersbook.com/2014/05/sql-drop-database-statement/)
* To delete tables in a database instance – **TRUNCATE**
* To rename database instances – **RENAME**

**ORACLE CREATE TABLE**

In Oracle, CREATE TABLE statement is used to create a new table in the database.

To create a table, you have to name that table and define its columns and datatype for each column.

**Syntax:**

**CREATE TABLE table\_name**

**(    column1 datatype [ NULL | NOT NULL ],**

**column2 datatype [ NULL | NOT NULL ],**

**...**

**column\_n datatype [ NULL | NOT NULL ]**

**);    
CREATE** **TABLE** customers

( customer\_id number(10) NOT NULL,

  customer\_name varchar2(50) NOT NULL,

  city varchar2(50)

);

**CREATE TABLE AS STATEMENT**  
The CREATE TABLE AS statement is used to create a table from an existing table by copying the columns of existing table.

**Syntax:**

CREATE TABLE new\_table  AS (SELECT \* FROM old\_table);

**Example:  
CREATE** **TABLE** newcustomers  **AS** (**SELECT** \*   **FROM** customers  **WHERE** customer\_id < 5000);

**CREATE TABLE EXAMPLE: COPYING SELECTED COLUMNS OF ANOTHER TABLE  
Syntax:**

**CREATE TABLE new\_table  AS (SELECT column\_1, column2, ... column\_n  FROM old\_table);**

Let's take an example:

**CREATE TABLE newcustomers2 AS (SELECT customer\_id, customer\_name  FROM customers WHERE customer\_id < 5000);**

**ORACLE ALTER TABLE STATEMENT**In Oracle, ALTER TABLE statement specifies how to add, modify, drop or delete columns in a table. It is also used to rename a table.

**Syntax:**

**ALTER** **TABLE** table\_name   **ADD** column\_name **column**-definition;

**Example:**

Consider that already existing table customers. Now, add a new column customer\_age into the table customers.

**ALTER** **TABLE** customers  **ADD** customer\_age varchar2(50);

**Syntax:**

**ALTER TABLE table\_name   MODIFY column\_name column\_type;**

**Example:**

**ALTER TABLE customers  MODIFY customer\_name varchar2(100) not null;   
  
Syntax:**

**ALTER TABLE table\_name   DROP COLUMN column\_name;**

**Example:**

**ALTER TABLE customers   DROP COLUMN customer\_name;**

**Syntax:**

**ALTER** **TABLE** table\_name  RENAME **COLUMN** old\_name **to** new\_name;

**Example:**

**ALTER** **TABLE** customers   RENAME **COLUMN** customer\_name **to** cname;

**Syntax:**

**ALTER** **TABLE** table\_name   RENAME **TO** new\_table\_name;

**Example:**

**ALTER** **TABLE** customers  RENAME **TO** retailers;

**ORACLE DROP TABLE STATEMENT**

Oracle DROP TABLE statement is used to remove or delete a table from the Oracle database.

**Syntax**

DROP TABLE customers;

**EXPERIMENT-3**

**OBJECTIVE:**

**To exercise and implements Data Manipulation Language Commands.**

**CONCEPT:**

**ORACLE SELECT STATEMENT  
  
Select all rows & all columns**

Select \* from tablename;

**Select all rows & specific columns**

Select column\_names from tablename;

**Select specific rows & all columns**

Select \* from tablename where condition;

**Select specific rows & specific columns**

Select column\_names from tablename where condition ; **ORACLE INSERT STATEMENT**  
In Oracle, INSERT statement is used to add a single record or multiple records into the table.

**Syntax: (Inserting a single record using the Values keyword):**

***INSERT******INTO******table****(column1, column2, ... column\_n )****VALUES****(expression1, expression2, ... expression\_n );*

***INSERT******INTO****suppliers  (supplier\_id, supplier\_name)****VALUES****(50, 'Flipkart');*

The Oracle INSERT ALL statement is used to insert multiple rows with a single INSERT statement. You can insert the rows into one table or multiple tables by using only one SQL command.

**Syntax:**

***INSERT ALL***

***INTO****table\_name (column1, column2, column\_n)****VALUES****(expr1, expr2, expr\_n)*

***INTO****table\_name(column1, column2, column\_n)****VALUES****(expr1, expr2, expr\_n)*

***INTO****table\_name (column1, column2, column\_n)****VALUES****(expr1, expr2, expr\_n)*

***SELECT****\*****FROM****dual;*

***INSERT****ALL*

***INTO****suppliers (supplier\_id, supplier\_name)****VALUES****(20, 'Google')*

***INTO****suppliers (supplier\_id, supplier\_name)****VALUES****(21, 'Microsoft')*

***INTO****suppliers (supplier\_id, supplier\_name)****VALUES****(22, 'Apple')*

***SELECT****\*****FROM****dual;*

**ORACLE UPDATE STATEMENT**  
In Oracle, UPDATE statement is used to update the existing records in a table.

**Syntax:**

***UPDATE table  SET column1 = expression1,   column2 = expression2,     ...   column\_n = expression\_n  WHERE conditions*;**

***UPDATE suppliers  SET supplier\_name = 'Kingfisher'  WHERE supplier\_id = 2;***

**ORACLE TRUNCATE TABLE**

In Oracle, TRUNCATE TABLE statement is used to remove all records from a table. It works same as DELETE statement but without specifying a WHERE clause. It is generally used when you don’t have to worry about rolling back

Once a table is truncated, it can’t be rolled back.

**Syntax**

***TRUNCATE******TABLE****tablename;*

***TRUNCATE******TABLE****customers;*

**ORACLE DELETE**

In Oracle, DELETE statement is used to remove all or specific records from a table.

**Syntax**

***DELETE******from****tablename where condition;*

***DELETE******from****customers;*

***DELETE******from****customers where cid=1;*

**EXPERIMENT-4**

**OBJECTIVE:**

**To exercise and implements SQL Constraints.**

**CONCEPT:  
SQL - Constraints**  
Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

**FOLLOWING ARE SOME OF THE MOST COMMONLY USED CONSTRAINTS AVAILABLE IN SQL:  
NOT NULL Constraint −** Ensures that a column cannot have NULL value.

**DEFAULT Constraint −** Provides a default value for a column when none is specified.

**UNIQUE Constraint −** Ensures that all values in a column are different.

**PRIMARY Key** − Uniquely identifies each row/record in a database table.

**FOREIGN Key −** Uniquely identifies a row/record in any of the given database table**.**

**CHECK Constraint −** The CHECK constraint ensures that all the values in a column satisfies certain conditions.

**INDEX −** Used to create and retrieve data from the database very quickly.

**SQL NOT NULL Constraint**By default, a column can hold NULL values. The NOT NULL constraint enforces a column to NOT accept NULL values.

CREATE TABLE Persons (

ID number(10) NOT NULL,

LastName varchar(20) NOT NULL,

FirstName varchar(20) NOT NULL,

Age number(10)

);

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values:  
  
**SQL UNIQUE Constraint**  
The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a **column or set of columns.**

**A PRIMARY KEY constraint automatically has a UNIQUE constraint.**

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

**CREATE TABLE Persons (**

**ID int NOT NULL UNIQUE,**

**LastName varchar(20) NOT NULL,**

**FirstName varchar(20),**

**Age int**

**);**

**CREATE TABLE Persons (**

**ID int NOT NULL,**

**LastName varchar(20) NOT NULL,**

**FirstName varchar(20),**

**Age int,**

**CONSTRAINT UC\_Person UNIQUE (ID,LastName)**

**);  
ADDING/ DELETING UNIQUE CONSTRAINT IN AN EXISTING TABLE**ALTER TABLE Persons ADD UNIQUE (ID);

ALTER TABLE Persons ADD CONSTRAINT UC\_Person UNIQUE (ID,LastName);

ALTER TABLE PersonsDROP CONSTRAINT UC\_Person;  
 **SQL PRIMARY KEY Constraint**The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only one primary key, which may consist of single or multiple fields.

**Adding Primary Key in a New Table**

CREATE TABLE Persons (

ID number(10) PRIMARY KEY,

LastName varchar(20) NOT NULL,

FirstName varchar(20),

Age number (10)

);

**To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:**

CREATE TABLE Persons (

ID number(10),

LastName varchar(20) NOT NULL,

FirstName varchar(20),

Age number(10),

CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName)

);

Note: In the example above there is only ONE PRIMARY KEY (PK\_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).  
SQL PRIMARY KEY on ALTER TABLE

*ALTER TABLE Persons ADD PRIMARY KEY (ID);*

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

*ALTER TABLE Persons ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);*  
 **DROP a PRIMARY KEY Constraint**  
  
ALTER TABLE PersonsDROP PRIMARY KEY;

ALTER TABLE Persons DROP CONSTRAINT PK\_Person;  
 **SQL FOREIGN KEY Constraint**  
  
A FOREIGN KEY is a key used to link two tables together.A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

**Creating Foreign Key in a Table**

CREATE TABLE Orders (  
    OrderID number(10) NOT NULL,  
    OrderNumber number(10)NOT NULL,  
    PersonID number(10),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);  
**Adding FOREIGN KEY in an Existing Table**

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

ALTER TABLE Orders ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);  
  
  
  
**DROP a FOREIGN KEY Constraint**  
ALTER TABLE Orders DROP CONSTRAINT FK\_PersonOrder;

**SQL CHECK Constraint**  
  
The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column. If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

CREATE TABLE Persons (  
    ID number(10) NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age number(10)  CHECK (Age>=18)  
);  
**Adding/ Deleting CHECK Constraint from an Existing table**

ALTER TABLE Persons ADD CHECK (Age>=18);

**Deleting  a CHECK Constraint**

ALTER TABLE Persons DROP CONSTRAINT CHK\_PersonAge;

**SQL DEFAULT Constraint**  
The DEFAULT constraint is used to provide a default value for a column.

The default value will be added to all new records IF no other value is specified.

CREATE TABLE Persons (  
    ID number(10) NOT NULL,  
    LastName varchar(20) NOT NULL,  
    FirstName varchar(20),  
    Age number(10) ,  
    City varchar(20) DEFAULT 'Indore'  
);  
 **Adding /Deleting DEFAULT Constraint from an Existing Table**

ALTER TABLE Persons MODIFY City DEFAULT 'Indore';

**DROP a DEFAULT Constraint**

ALTER TABLE Persons ALTER COLUMN City DROP DEFAULT;

**SQL CREATE INDEX Statement**

The CREATE INDEX statement is used to create indexes in tables. Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

CREATE INDEX index\_name

ON table\_name (column1, column2, ...);

CREATE INDEX idx\_pname

ON Persons (LastName, FirstName);

DROP INDEX index\_name ON table\_name;

**EXPERIMENT-5**

**OBJECTIVE:**

**To exercise and implement SQL Functions.**

**CONCEPT:  
SQL Functions**

SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc. SQL functions are divided into two categories,

Aggregate Functions

Scalar Functions **AGGREGATE FUNCTIONS**These functions **return a single value** after performing calculations on a group of values. Following are some of the frequently used Aggregate functions:

* **AVG() Function**
* **COUNT() Function**
* **FIRST() Function**
* **LAST() Function**
* **MAX() Function**
* **MIN() Function**
* **SUM() Function**

**SCALAR FUNCTIONS**Scalar functions return a single value from an input value. Following are some frequently used Scalar Functions in SQL.

* **UCASE() Function**
* **LCASE() Function**
* **MID() Function**
* **ROUND() Function**

**EXPERIMENT-6**

**OBJECTIVE:**

**To exercise and implement Wildcard Characters in SQL.**

**CONCEPT:**

## SQL Wildcard Characters

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the [SQL LIKE](https://www.w3schools.com/sql/sql_like.asp) operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| \* | Represents zero or more characters | bl\* finds bl, black, blue, and blob |
| ? | Represents a single character | h?t finds hot, hat, and hit |
| [] | Represents any single character within the brackets | h[oa]t finds hot and hat, but not hit |
| ! | Represents any character not in the brackets | h[!oa]t finds hit, but not hot and hat |
| - | Represents a range of characters | c[a-b]t finds cat and cbt |
| # | Represents any single numeric character | 2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295 |

## Using the % Wildcard

The following SQL statement selects all customers with a City starting with "ber":

SELECT \* FROM Customers WHERE City LIKE 'ber%';

## Using the \_ Wildcard

The following SQL statement selects all customers with a City starting with any character, followed by "ondon":

### Example

SELECT \* FROM Customers WHERE City LIKE '\_ondon';

**EXPERIMENT-7**

**OBJECTIVE:**

**To exercise and implement Order By command.**

## CONCEPT:

## The SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

### ORDER BY Syntax

SELECT *column1*,*column2, ...*  
FROM *table\_name*  
ORDER BY *column1, column2, ...*ASC|DESC;

## ORDER BY Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:

### Example

SELECT \* FROM Customers  
ORDER BY Country;

## ORDER BY DESC Example

The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

### Example

SELECT \* FROM Customers  
ORDER BY Country DESC;

## ORDER BY Several Columns Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

### Example

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

**EXPERIMENT-8**

**OBJECTIVE:**

**To exercise and implement Group By command.**

**CONCEPT:**

## The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

### GROUP BY Syntax

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *condition*  
GROUP BY *column\_name(s)*ORDER BY *column\_name(s);*

## SQL GROUP BY Examples

The following SQL statement lists the number of customers in each country:

### Example

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country;

The following SQL statement lists the number of customers in each country, sorted high to low:

### Example

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
ORDER BY COUNT(CustomerID) DESC;

## The SQL HAVING Clause

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

### HAVING Syntax

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *condition*  
GROUP BY *column\_name(s)*HAVING *condition*ORDER BY *column\_name(s);*

## SQL HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

### Example

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;

The following SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

### Example

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5  
ORDER BY COUNT(CustomerID) DESC;

**EXPERIMENT-9**

**OBJECTIVE:**

**To exercise and implement Data SQL Joins.**

**CONCEPT:**

**What is SQL Join?**

* JOIN clause combines rows from two or more tables.
* creates a set of rows in a temporary table.

**Types of SQL JOIN**

**• EQUI JOIN**

– EQUI JOIN is a simple SQL join.

– Uses the equal sign(=) as the comparison operator for the condition

**• NON EQUI JOIN**

– NON EQUI JOIN uses comparison operator other than the equal sign.

– The operators uses like >, <, >=, <= with the condition.

**Types of SQL EQUI JOIN**

**• INNER JOIN**

– Returns only matched rows from the participating tables.

– Match happened only at the key record of participating tables.

**• OUTER JOIN**

– Returns all rows from one table and

– Matching rows from the secondary table and

– Comparison columns should be equal in both the tables.

**List of SQL JOINS**

**INNER JOIN**

• The INNER JOIN selects all rows from both participating tables as long as there is a match between the columns.

An SQL INNER JOIN is same as JOIN clause, combining rows from two or more tables.

**• Example: INNER JOIN**

• SELECT \* FROM table\_A INNER JOIN table\_B ON table\_A.A=table\_B.A;

**LEFT JOIN or LEFT OUTER JOIN**

The SQL LEFT JOIN, joins two tables and fetches rows based on a condition, which are matching in both the tables.

The unmatched rows will also be available from the table before the JOIN clause.

**• Example: LEFT JOIN or LEFT OUTER JOIN**

SELECT \* FROM table\_A LEFT JOIN table\_B ON table\_A.A=table\_B.A;

**RIGHT JOIN or RIGHT OUTER JOIN**

The SQL RIGHT JOIN, joins two tables and fetches rows based on a condition, which are matching in both the tables.

The unmatched rows will also be available from the table written after the JOIN clause.

**• Example : RIGHT JOIN or RIGHT OUTER JOIN**

• SELECT \* FROM table\_A RIGHT JOIN table\_B ON table\_A.A=table\_B.A;

**FULL OUTER JOIN**

Combines the results of both left and right outer joins.

Returns all matched or unmatched rows. Includes tables on both sides of the join clause.

**Example: FULL OUTER JOIN**

SELECT \* FROM table\_A FULL OUTER JOIN table\_B ONtable\_A.A=table\_B.A;

**NATURAL JOIN**

The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with same name of associate tables will appear once only.The associated tables have one or more pairs of identically named columns. The columns must be the same data type. Don’t use ON clause in a natural join.

**Example: NATURAL JOIN**

SELECT \* FROM table\_A NATURAL JOIN table\_B;

**CROSS JOIN**

The SQL CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table, if no WHERE clause is used along with CROSS JOIN.This kind of result is called as Cartesian Product. If, WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN.

**Example : CROSS JOIN**

SELECT \* FROM table\_A CROSS JOIN table\_B;

**SELF JOIN**

A self join is a join in which a table is joined with itself (Unary relationships), specially when the table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself means that each row of the table is combined with itself and with every other row of the table. The self join can be viewed as a join of two copies of the same table.

**Example : SELF JOIN**

SELECT \* FROM table\_A X, table\_A Y WHERE X.A=Y.A; Example : INNER JOIN

**EXPERIMENT-10**

**OBJECTIVE:**

**To exercise and implement SQL Subqueries.**

**CONCEPT:**

## Introduction to the Oracle subquery

A subquery is a [SELECT](https://www.oracletutorial.com/oracle-basics/oracle-select/) statement nested inside another statement such as [SELECT](https://www.oracletutorial.com/oracle-basics/oracle-select/), [INSERT](https://www.oracletutorial.com/oracle-basics/oracle-insert/), [UPDATE](https://www.oracletutorial.com/oracle-basics/oracle-update/), or [DELETE](https://www.oracletutorial.com/oracle-basics/oracle-delete/).

Consider this following subquery example that uses the products table from the [sample database](https://www.oracletutorial.com/getting-started/oracle-sample-database/).

The following query uses the [MAX()](https://www.oracletutorial.com/oracle-aggregate-functions/oracle-max/) function to return the highest list price from the products table:

|  |  |
| --- | --- |
|  | **SELECT MAX( list\_price ) FROM products;** |

To select the detailed information of the most expensive products, you use the list price above (8867.99) in the following query:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **SELECT**  **product\_id,**  **product\_name,**  **list\_price**  **FROM**  **products**  **WHERE**  **list\_price = 8867.99;** |

As you can see, we need to execute two queries separately to get the most expensive product information. By using a subquery, we can nest the first query inside the second one as shown in the following statement:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13** | **SELECT**  **product\_id,**  **product\_name,**  **list\_price**  **FROM**  **products**  **WHERE**  **list\_price = (**  **SELECT**  **MAX( list\_price )**  **FROM**  **products**  **);** |

In this example, the query that retrieves the max price is called the subquery and the query that selects the detailed product data is called the outer query. We say that the subquery is nested within the outer query. Note that a subquery must appear within parentheses ().

Oracle evaluates the whole query above in two steps:

* First, execute the subquery.
* Second, use the result of the subquery in the outer query.

## Advantages of Oracle subqueries

These are the main advantages of subqueries:

* Provide an alternative way to query data that would require complex [joins](https://www.oracletutorial.com/oracle-basics/oracle-inner-join/) and [unions](https://www.oracletutorial.com/oracle-basics/oracle-union/).
* Make the complex queries more readable.
* Allow a complex query to be structured in a way that it is possible to isolate each part.
* Second, use the result of the subquery in the outer query.

### Oracle subquery with comparison operators example

The subqueries that use comparison operators e..g, >, >=, <, <=, <>, = often include [aggregate functions](https://www.oracletutorial.com/oracle-aggregate-functions/), because an aggregate function returns a single value that can be used for comparison in the [WHERE](https://www.oracletutorial.com/oracle-basics/oracle-where/) clause of the outer query.

For example, the following query finds products whose list price is greater than the average list price.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15** | **SELECT**  **product\_id,**  **product\_name,**  **list\_price**  **FROM**  **products**  **WHERE**  **list\_price > (**  **SELECT**  **AVG( list\_price )**  **FROM**  **products**  **)**  **ORDER BY**  **product\_name;** |

### Oracle subquery with IN and NOT IN operators

The subquery that uses the [IN](https://www.oracletutorial.com/oracle-basics/oracle-in/) operator often returns a list of zero or more values. After the subquery returns the result set, the outer query makes uses of them**.**

**NESTED QUERIES IN SQL**

In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query. We will use STUDENT, COURSE, STUDENT\_COURSE tables for understanding nested queries.

**STUDENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S\_ID** | **S\_NAME** | **S\_ADDRESS** | **S\_PHONE** | **S\_AGE** |
| **S1** | **RAM** | **DELHI** | **9455123451** | **18** |
| **S2** | **RAMESH** | **GURGAON** | **9652431543** | **18** |
| **S3** | **SUJIT** | **ROHTAK** | **9156253131** | **20** |
| **S4** | **SURESH** | **DELHI** | **9156768971** | **18** |

**COURSE**

|  |  |
| --- | --- |
| **C\_ID** | **C\_NAME** |
| **C1** | **DSA** |
| **C2** | **Programming** |
| **C3** | **DBMS** |

**STUDENT\_COURSE**

|  |  |
| --- | --- |
| **S\_ID** | **C\_ID** |
| **S1** | **C1** |
| **S1** | **C3** |
| **S2** | **C1** |
| **S3** | **C2** |
| **S4** | **C2** |
| **S4** | **C3** |

**There are mainly two types of nested queries:**

● Independent Nested Queries: In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.

IN: If we want to find out S\_ID who are enrolled in C\_NAME ‘DSA’ or ‘DBMS’, we can write it with the help of independent nested query and IN operator. From COURSE table, we can find out C\_ID for C\_NAME ‘DSA’ or DBMS’ and we can use these C\_IDs for finding S\_IDs from STUDENT\_COURSE TABLE.

STEP 1: Finding C\_ID for C\_NAME =’DSA’ or ‘DBMS’

Select C\_ID from COURSE where C\_NAME = ‘DSA’ or C\_NAME = ‘DBMS’

STEP 2: Using C\_ID of step 1 for finding S\_ID

Select S\_ID from STUDENT\_COURSE where C\_ID IN

(SELECT C\_ID from COURSE where C\_NAME = ‘DSA’ or C\_NAME=’DBMS’);

The inner query will return a set with members C1 and C3 and outer query will return those S\_IDs for which C\_ID is equal to any member of set (C1 and C3 in this case). So, it will return S1, S2 and S4.

Note: If we want to find out names of STUDENTs who have either enrolled in ‘DSA’ or ‘DBMS’, it can be done as:

Select S\_NAME from STUDENT where S\_ID IN

(Select S\_ID from STUDENT\_COURSE where C\_ID IN

(SELECT C\_ID from COURSE where C\_NAME=’DSA’ or C\_NAME=’DBMS’));

NOT IN: If we want to find out S\_IDs of STUDENTs who have neither enrolled in ‘DSA’ nor in ‘DBMS’, it can be done as:

Select S\_ID from STUDENT where S\_ID NOT IN

(Select S\_ID from STUDENT\_COURSE where C\_ID IN

(SELECT C\_ID from COURSE where C\_NAME=’DSA’ or C\_NAME=’DBMS’));

The innermost query will return a set with members C1 and C3. Second inner query will return those S\_IDs for which C\_ID is equal to any member of set (C1 and C3 in this case) which are S1, S2 and S4. The outermost query will return those S\_IDs where S\_ID is not a member of set (S1, S2 and S4). So it will return S3.

## SQL Subqueries Example :

In this section, you will learn the requirements of using subqueries. We have the following two tables 'student' and 'marks' with common field 'StudentID'.

Now we want to write a query to identify all students who get better marks than that of the student who's StudentID is 'V002', but we do not know the marks of 'V002'.

Select \* from student where studentid in (Select studentid from marks where total\_marks>(Select total\_marks from marks where studentid=’v002’));

**Name Salary**

**---------------**

**abc 100000**

**bcd 1000000**

**efg 40000**

**ghi 500000**

**SELECT name, MAX (salary) AS salary**

**FROM employee**

**WHERE salary < (SELECT MAX (salary)**

**FROM employee);**

**How to find the third largest salary?**

Simple, we can do one more nesting.

**SELECT name, MAX (salary) AS salary**

**FROM employee**

**WHERE salary < (SELECT MAX (salary)**

**FROM employee**

**WHERE salary < (SELECT MAX (salary)**

**FROM employee)**

**);**

**EXPERIMENT-11**

**OBJECTIVE:**

**To exercise and implement PL/ SQL.**

**CONCEPT:**

# PL/SQL

PL/SQL stands for Procedural Language/SQL.

PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL.

The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other. Typically, each block performs a logical action in the program

# Features of PL/SQL

• PL/SQL has the following features − • PL/SQL is tightly integrated with SQL.

• It offers extensive error checking.

• It offers numerous data types.

• It offers a variety of programming structures.

• It supports structured programming through functions and procedures.

• It supports object-oriented programming.

• It supports the development of web applications and server pages.

# PL/SQL - Environment Setup

• Running large programs from the command prompt may land you in inadvertently losing some of the work.

It is always recommended to use the command files. To use the command files −

• Type your code in a text editor, like Notepad, Notepad+, or EditPlus, etc.

• Save the file with the .sql extension in the home directory.

• Launch the SQL\*Plus command prompt from the directory where you created your PL/SQL file.

• Type @file\_name at the SQL\*Plus command prompt to execute your program

# PL/SQL - Basic Syntax

# PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts −

**• 1. Declarations**

This section starts with the keyword DECLARE. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.

**• 2. Executable Commands**

This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.

**• 3. Exception Handling**

This section starts with the keyword EXCEPTION. This optional section contains exception(s) that handle errors in the program.

**PL/SQL - Basic Syntax**

**The 'Hello World' Example**

DECLARE

message varchar2(20):= 'Hello, World!'; BEGIN

dbms\_output.put\_line(message); END;

**/**

***NOTE: We need to execute "set serveroutput on" if we need to see the output of the code.***

# The PL/SQL Identifiers

• PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. The identifiers consist of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters.

• By default, identifiers are not case-sensitive. So you can use integer or INTEGER to represent a numeric value. You cannot use a reserved keyword as an identifier.

# The PL/SQL Comments

• Program comments are explanatory statements that can be included in the PL/SQL code that you write and helps anyone reading its source code. All programming languages allow some form of comments.

• The PL/SQL supports single-line and multi-line comments. All characters available inside any comment are ignored by the PL/SQL compiler.

• The PL/SQL single-line comments start with the delimiter -- (double hyphen) and multi-line comments are enclosed by /\* and \*/.

DECLARE

-- variable declaration

message varchar2(20):= 'Hello, World!'; BEGIN

/\* \* PL/SQL executable statement(s) \*/ dbms\_output.put\_line(message);

END**;**

**/**

# PL/SQL - Variables

The syntax for declaring a variable is −

*<variable\_name> <datatype> := <default\_value>;*

For example − • sales number(10, 2);

• name varchar2(25);

• address varchar2(100);

• name VARCHAR2(50) := ‘ZEBA';

**PL/SQL Variable Constraints**

**NOT NULL**

Can not be empty

**CONSTANT**

Can not be changed

**PL/SQL Variables Examples with and without Constraints**

Age number;

Last char ( 10 );

DVal Date := Sysdate;

SID number not null;

Adjust constant number := 1;

CanLoop boolean := true

# Program to Add 2 Numbers

Declare

Var1 integer; Var2 integer;

Var3 integer;

Begin

Var1:=&var1; Var2:=&var2;

Var3:=var1+var2;

Dbms\_output.put\_line('the sum of numbers is '|| var3); End;

**/**

**IF THEN**

**declare**  
var1 integer;  
**begin**  
var1:=&var1;  
if var1>20 then  
dbms\_output.put\_line('Number is Greater Than 20');  
end if;  
**end**;  
/

**IF THEN ELSE**

declare

var1 integer;

begin

var1:=&var1;

if var1>10 and var1<20 then

dbms\_output.put\_line('Number is between 10 and 20');

else

dbms\_output.put\_line('Number is out of range');

end if;

end;

/

**LOOP**

declare  
var1 integer;  
begin  
var1:=&var1;  
loop  
dbms\_output.put\_line(var1);  
var1:=var1+1;  
exit when var1>5;  
end loop;  
end;  
/