

DATABASE MANAGEMENT SYSTEMS LAB MANUAL

(Version 2.0, A.Y 2022-23)

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING & APPLICATION

Approved By Teacher Council of Department of CSE&A on 24/05/2022

Prepared By:

Mrs. Sushma Rath

Content Contribution:

Mr. Pradyumna. Kumar Ratha

Mrs. Sushma Rath

~O~

SAMBALPUR UNIVERSITY INSTITUTE OF INFORMATION TECHNOLOGY

an autonomous constituent institute of Sambalpur University in the line of IIT's



Leveraging Technology

Inspiring Innovation

Flourishing Mankind

VISION AND MISSION OF INSTITUTE

Vision:

To enlighten the young minds by giving patronage to the unsatisfied thirst for knowledge, to contribute to mankind through excellence in scientific and technological education and research, to prove to be a valuable resource for industry and society alike and to remain a symbol of pride for all Indians.

Mission:

To inspire young minds to lead the world in the right direction by quality research in Information Technology and allied areas and to create a technically sound workforce that puts the nation in safer hands for a bright future.

OBJECTIVES & OUTCOMES

Objectives:

- Understand basic concepts of how a database stores information via tables. Understanding of SQL syntax used with MySQL.
- Learn how to retrieve and manipulate data from one or more tables. Know how to filter data based upon multiple conditions.
- Updating and inserting data into existing tables.
- Learning how the relationships between tables will effect the SQL.

Outcomes:

- Remember and understand the basic Concepts/Principles of database management systems lab.
- Analyze the various concepts to understand them through case studies.
- Apply the knowledge in understanding practical problems.
- Execute/Create the project or field assignment as per the knowledge gained in the course.

HARDWARE & SOFTWARE REQUIREMENTS

Hardware Requirements:

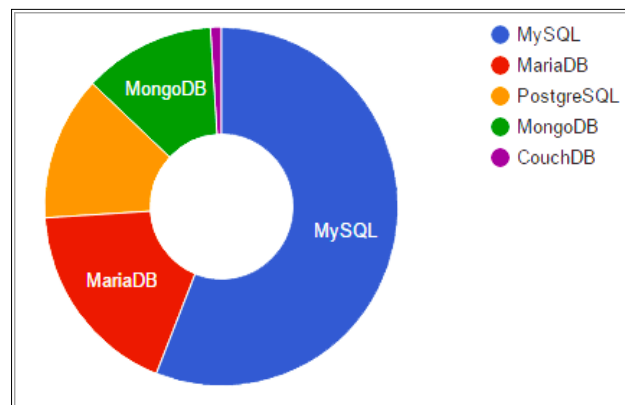
	MINIMUM	RECOMMENDED
CPU	64 bit x 86 CPU	Multi Core (64 bit x 86 CPU)
RAM	8 GB RAM	8 GB or Higher
DISPLAY	1024 x 768	1920 x 1200 or higher

Software Requirements:

Enlisted below are the most popular Free Database Software.

1. MySQL
2. Oracle
3. SQL Server
4. Firebird
5. PostgreSQL
6. MongoDB
7. Cubrid
8. MariaDB
9. DynamoDB
10. CouchDB
11. Altibase

The Below graph will show you more details of this research and the developer's choice of databases according to the requirement analysis.



INTERNAL ASSESMENT MARK SPUT UP

Attendance :	10 Marks
Lab Record :	30 Marks
Day to Day Performance :	30 Marks
Viva :	10 Marks
Semester Exam :	20 Marks
TOTAL MARKS :	100 Marks

***May change as per the academic decision.**

ACKNOWLEDGEMENT

We wish to acknowledge our deep gratitude to Ms. Niharika Panda for her tired-less cooperation.

INDEX

S. No	Title	Page no.
1	Introduction to DBMS	8-12
2	Relational Algebra & Relational Calculus	13-16
3	Mapping from ER to Relational Mapping	17-18
4	SQL	19-22
5	Various Data Types	23-25
6	Practice DDL Commands	26-28
7	Practice DML Commands	29-33
8	Practice DRL/DQL Commands	34-43
9	Practice Key Constraints	44-49
10	Practice Functions	50-61
11	Practice JOIN & VIEW	62-67
12	Practice Sub-Query	68-71
13	Practice SQL Clauses	72-76
14	Practice TCL Commands	77-81
15	Practice DCL Commands	82
16	Practice PL/SQL	83-86
17	Practice Cursors, Exception Handling &Triggers	87-98
18	Assignment 1	99-100
19	Assignment 2	101
20	Assignment 3	102-104
21	Assignment 4	105-107
22	Assignment 5	108
23	Assignment 6	109
24	Assignment 7	110-111

INTRODUCTION TO DBMS

Database Management System:

This model is like a hierarchical tree structure, used to construct a hierarchy of records in the form of nodes and branches. The data elements present in the structure have Parent-Child relationship.

Closely related information in the parent-child structure is stored together as a logical unit. A parent unit may have many child units, but a child is restricted to have only one parent.

The drawbacks of this model are:

- The hierarchical structure is not flexible to represent all the relationship proportions, which occur in the real world.
- It cannot demonstrate the overall data model for the enterprise because of the non-availability of actual data at the time of designing the data model.
- It cannot represent the Many-to-Many relationship.

Network Model:

It supports the One-To-One and One-To-Many types only. The basic objects in this model are Data Items, Data Aggregates, Records and Sets.

It is an improvement on the Hierarchical Model. Here multiple parent-child relationships are used.

Rapid and easy access to data is possible in this model due to multiple access paths to the data elements.

Relational Model:

- Does not maintain physical connection between relations
- Data is organized in terms of rows and columns in a table
- The position of a row and/or column in a table is of no importance
- The intersection of a row and column must give a single value

Features of an RDBMS:

- The ability to create multiple relations and enter data into them
- An attractive query language
- Retrieval of information stored in more than one table

- An RDBMS product has to satisfy at least seven of the 12 rules of Codd to be accepted as a full-fledged RDBMS.

Relational Database Management System:

RDBMS is acronym for Relation Database Management System. Dr. E. F. Codd first introduced the Relational Database Model in 1970. The Relational model allows data to be represented in a simple row-column. Each data field is considered as a column and each record is considered as a row. Relational Database is more or less similar to Database Management System. In relational model there is relation between their data elements. Data is stored in tables. Tables have columns, rows and names. Tables can be related to each other if each has a column with a common type of information.

The most famous RDBMS packages are Oracle, Sybase and Informix.

Degree of Relationship:

- One to One (1:1)
- One to Many or Many to One (1:M / M: 1)
- Many to Many (M: M)

The Degree of Relationship indicates the link between two entities for a specified occurrence of each.

One to One Relationship (1: 1)

Student Has Roll No. one student has only one Roll no. For one occurrence of the first entity, there can be, at the most one related occurrence of the second entity, and vice-versa.

One to Many or Many to One Relationship: (1: M /M: 1)

Course Contains Students as per the Institutions Norm, One student can enroll in one course at a time however, in one course; there can be more than one student.

For one occurrence of the first entity there can exist many related occurrences of the second entity and for every occurrence of the second entity there exists only one associated occurrence of the first.

Many to Many Relationships: (M: M)

Students Appears Tests the major disadvantage of the relational model is that a clear-cut interface cannot be determined. Reusability of a structure is not possible. The Relational Database now accepted model on which major database system are built.

Some basic rules have to be followed for a DBMS to be relational. They are known as Codd's rules, designed in such a way that when the database is ready for use it encapsulates the relational theory to its full potential. These twelve rules are as follows.

E. F. Codd Rules:

1. The Information Rule: All information must be store in table as data values.
2. The Rule of Guaranteed Access: Every item in a table must be logically addressable with the help of a table name.
3. The Systematic Treatment of Null Values: The RDBMS must be taken care of null values to represent missing or inapplicable information.
4. The Database Description Rule: A description of database is maintained using the same logical structures with which data was defined by the RDBMS.
5. Comprehensive Data Sub Language: According to the rule the system must support data definition, view definition, data manipulation, integrity constraints, and authorization and transaction management operations.
6. The View Updating Rule: All views that are theoretically updateable are also updateable by the system.
7. The Insert and Update Rule: This rule indicates that all the data manipulation commands must be operational on sets of rows having a relation rather than on a single row.
8. The Physical Independence Rule: Application programs must remain unimpaired when any changes are made in storage representation or access methods.
9. The Logical Data Independence Rule: The changes that are made should not affect the user's ability to work with the data. The change can be splitting table into many more tables.
10. The Integrity Independence Rule: The integrity constraints should store in the system catalog or in the database.
11. The Distribution Rule: The system must be access or manipulate the data that is distributed in other systems.
12. The Non-subversion Rule: If a RDBMS supports a lower level language then it should not bypass any integrity constraints defined in the higher level.

Object Relational Database Management System:

Oracle8 and later versions are supported object-oriented concepts. A structure once created can be reused is the fundamental of the OOP's concept. So we can say Oracle8 is supported Object Relational model, Object – oriented model both. Oracle products are based on a concept known as a client-server technology. This concept involves segregating the processing of an application between two systems.

MySQL Database:

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons –

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table.
- The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

ORACLE:

ORACLE is a powerful RDBMS product that provides efficient and effective solutions for major database features. This includes:

- Large databases and space management control
- Many concurrent database users
- High transaction processing performance
- High availability
- Controlled availability

- Industry accepted standards
- Manageable security
- Database enforced integrity
- Client/Server environment
- Distributed database systems
- Portability
- Compatibility
- Connectivity

An ORACLE database system can easily take advantage of distributed processing by using its Client/ Server architecture. In this architecture, the database system is divided into two parts:

A front-end or a client portion:

The client executes the database application that accesses database information and interacts with the user.

A back-end or a server portion:

The server executes the ORACLE software and handles the functions required for concurrent, shared data access to ORACLE database.

RELATIONAL ALGEBRA & RELATIONAL CALCULUS

Relational algebra and calculus are the theoretical concepts used on relational model.

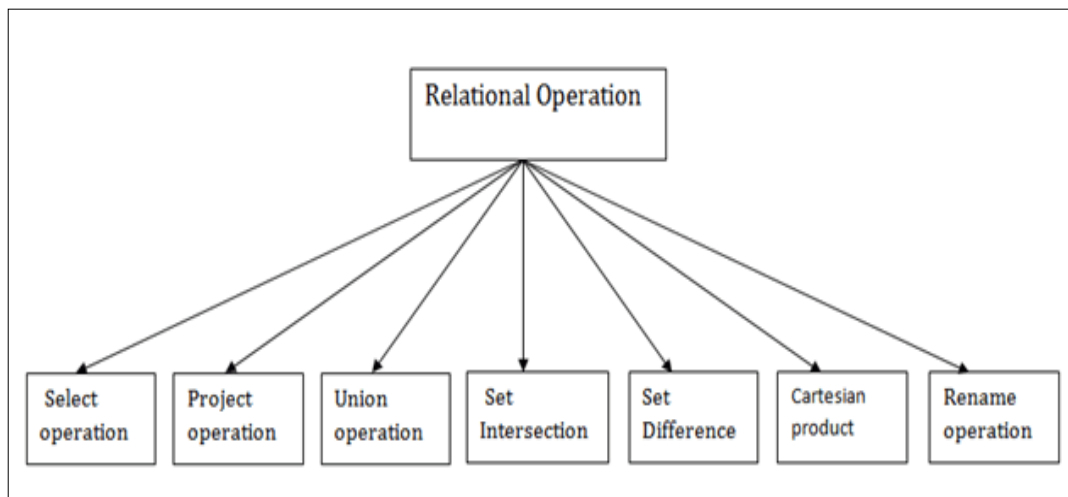
RDBMS is a practical implementation of relational model.

SQL is a practical implementation of relational algebra and calculus.

RELATIONAL ALGEBRA

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

Types of Relational operation



Select Operation:

- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).

Notation: $\sigma p(r)$

Where:

σ is used for selection prediction

r is used for relation

p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like $=, \neq, \geq, <, >, \leq$.

Project Operation:

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by Π .

Notation: $\Pi A_1, A_2, A_n (r)$

Where:

A_1, A_2, A_3 is used as an attribute name of relation r .

Union Operation:

- Suppose there are two tuples R and S . The union operation contains all the tuples that are either in R or S or both in $R \& S$.
- It

Notation: $R \cup S$

eliminates the duplicate tuples. It is denoted by \cup .

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Set Intersection:

- Suppose there are two tuples R and S . The set intersection operation contains all tuples that are in both $R \& S$.
- It is denoted by intersection \cap .

Notation: $R \cap S$

Set Difference:

- Suppose there are two tuples R and S . The set intersection operation contains all tuples that are in R but not in S .
- It is denoted by intersection minus $(-)$.

Notation: $R \bowtie S$

Cartesian product:

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by \times .

Notation: $E \times D$

Rename Operation:

The rename operation is used to rename the output relation. It is denoted by **ρ** (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

$\rho(\text{STUDENT1}, \text{STUDENT})$

RELATIONAL CALCULUS

There is an alternate way of formulating queries known as Relational Calculus. Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results. The relational calculus tells what to do but never explains how to do. Most commercial relational languages are based on aspects of relational calculus including SQL-QBE and QUEL.

Why it is called Relational Calculus?

It is based on Predicate calculus, a name derived from branch of symbolic language. A predicate is a truth-valued function with arguments. On substituting values for the arguments, the function result in an expression called a proposition. It can be either true or false. It is a tailored version of a subset of the Predicate Calculus to communicate with the relational database.

Many of the calculus expressions involves the use of Quantifiers. There are two types of quantifiers:

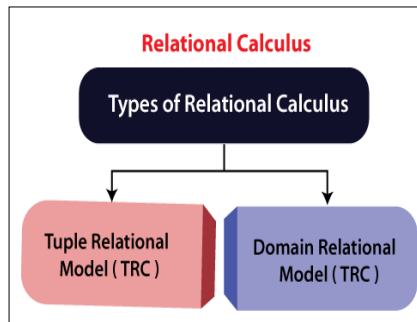
- **Universal Quantifiers:** The universal quantifier denoted by \forall is read as for all which means that in a given set of tuples exactly all tuples satisfy a given condition.
- **Existential Quantifiers:** The existential quantifier denoted by \exists is read as for all which means that in a given set of tuples there is at least one occurrences whose value satisfy a given condition.

Before using the concept of quantifiers in formulas, we need to know the concept of Free and Bound Variables.

A tuple variable t is bound if it is quantified which means that if it appears in any occurrences a variable that is not bound is said to be free.

Free and bound variables may be compared with global and local variable of programming languages.

Types of Relational calculus:



Tuple Relational Calculus (TRC)

It is a non-procedural query language which is based on finding a number of tuple variables also known as range variable for which predicate holds true. It describes the desired information without giving a specific procedure for obtaining that information. The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation. The result of the relation can have one or more tuples.

Notation:

A Query in the tuple relational calculus is expressed as following notation

$$\{T \mid P(T)\} \text{ or } \{T \mid \text{Condition}(T)\}$$

Where

T is the resulting tuples

P(T) is the condition used to fetch T.

Domain Relational Calculus (DRC)

The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes. Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives \wedge (and), \vee (or) and \neg (not). It uses Existential (\exists) and Universal Quantifiers (\forall) to bind the variable. The QBE or Query by example is a query language related to domain relational calculus.

$$\{a_1, a_2, a_3, \dots, a_n \mid P(a_1, a_2, a_3, \dots, a_n)\}$$

a_1, a_2 are attributes

P stands for formula built by inner attributes

MAPPING FROM ER TO RELATIONAL MAPPING

Step1: Mapping of regular entity types

Step2: Mapping of weak entity types

Step3: Mapping of binary 1:1 relationship types

Step4: Mapping of binary 1:N relationship types

Step5: Mapping of binary M:N relationship

Step6: Mapping of multivalued attributes

Step7: Mapping of n-ary relationship types

Step1: Mapping of Regular Entity Types

- For each regular(strong) entity type E in the ER schema,create a relation R that includes all the simple attributes of E.
- Choose one of the key attributes of E as the primary key of R.
- If the chosen key of E is composite the set of simple attributes together form the primary key of R.

Step2: Mapping of Weak Entity Types

- For each weak entity type w in the ER schema with owner entity type E,create a relation R and include all simple attributes (or simple components of composite attributes)of W as attributes of R.
- The primary key attribute of the relation that correspond to the owner entity type,include as foreign key attributes of R.
- The primary key of R is the combination of the primary key of the owner and the partial key of the weak entity type w ,if any.

Step 3: Mapping of binary 1:1 Relationship Types

Possible approaches :

- Foreign key approach
- Merged relation approach

Foreign key approach:

Choose one of the relation S say and include as a foreign key in s the primary key of T, it is better to choose entity type with total participation in R in the role of s, include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type R as attributes of S.

Merged relation approach:

- An alternative mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation.
- This may be appropriate when both participations are total.

Step 4: Mapping of Binary 1:N Relationship Types

For each regular binary 1:N relationship type R, identify the relation S that represents the participating entity type at the N-side of the relationship type. Use this as the base and create a relation including the primary key of the other entity type as a foreign key.

Include any simple attributes of the 1:N relationship type as attributes of S.

Step 5: Mapping of Binary M:N Relationship Types

- For each regular binary m:n relationship type R, create a new relation S to represent R.
- Include as foreign key attributes in S.
- The primary keys of the relations that represent the participating entity types, their combination will form the primary key of S.
- Also include any simple attributes of the M:N relationship type as attributes of S.

Step 6: Mapping of Multivalued Attributes

- For each multivalued attribute A, create a new relation R.
- The attribute locations represent the multivalued attribute locations of department while number as foreign key represents the primary key of the department relation.

Step 7: Mapping of n-ary Relationship Types

- For each n-ary relationship type R, where $n > 2$, create a new relation s to represent R.
- Include as foreign key attributes in s the primary keys of the relations that represent the participating entity types.
- Also include any simple attributes of the n-ary relationship type as attributes of S.

SQL

SQL is a database computer language designed for the retrieval and management of data in a relational database. SQL stands for Structured Query Language.

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Applications of SQL:

As mentioned before, SQL is one of the most widely used query language over the databases.

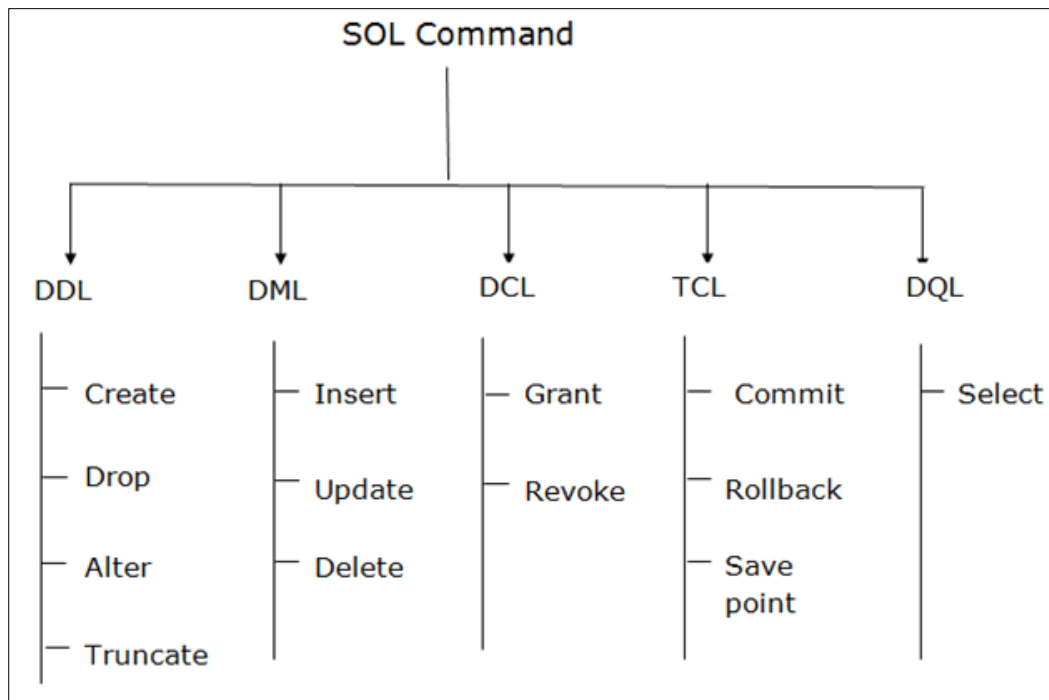
- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

CREATE It is used to create a new table in the database.

DROP: It is used to delete both the structure and record stored in the table.

ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

UPDATE: This command is used to update or modify the value of a column in the table.

DELETE: It is used to remove one or more row from a table.

Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

Grant: It is used to give user access privileges to a database.

Revoke: It is used to take back permissions from the user.

Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

Commit: Commit command is used to save all the transactions to the database.

Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Data Query/Retrieval Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

VARIOUS DATA TYPES

The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

String Data Types:

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LOBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Numeric Data Types:

Data type	Description
BIT(<i>size</i>)	A bit-value type. The number of bits per value is specified in <i>size</i> . The <i>size</i> parameter can hold a value from 1 to 64. The default value for <i>size</i> is 1.
TINYINT(<i>size</i>)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The <i>size</i> parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(<i>size</i>)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The <i>size</i> parameter specifies the maximum display width (which is 255)
MEDIUMINT(<i>size</i>)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The <i>size</i> parameter specifies the maximum display width (which is 255)
INT(<i>size</i>)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The <i>size</i> parameter specifies the maximum display width (which is 255)
INTEGER(<i>size</i>)	Equal to INT(<i>size</i>)
BIGINT(<i>size</i>)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The <i>size</i> parameter specifies the maximum display width (which is 255)
FLOAT(<i>size</i> , <i>d</i>)	A floating point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(<i>p</i>)	A floating point number. MySQL uses the <i>p</i> value to determine whether to use FLOAT or DOUBLE for the resulting data type. If <i>p</i> is from 0 to 24, the data type becomes FLOAT(). If <i>p</i> is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(<i>size</i> , <i>d</i>)	A normal-size floating point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter
DOUBLE PRECISION(<i>size</i> , <i>d</i>)	
DECIMAL(<i>size</i> , <i>d</i>)	An exact fixed-point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter. The maximum number for <i>size</i> is 65. The maximum number for <i>d</i> is 30. The default value for <i>size</i> is 10. The default value for <i>d</i> is 0.
DEC(<i>size</i> , <i>d</i>)	Equal to DECIMAL(<i>size</i> , <i>d</i>)

Date and Time Data Types:

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(<i>fsp</i>)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(<i>fsp</i>)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(<i>fsp</i>)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

PRACTICE DDL COMMANDS

Create Table Syntax :

```
CREATE TABLE <table_name>
(
column_name1 data_type(width),
column_name2 data_type(width),
column_name3 data_type(width),
....
);
```

Create Table Example:

Create table employee (empno number(5), Ename varchar(20), job char(10), hiredate date , sal number(10), deptno number(5));

Output: table created

Note: To view the structure of a table we have another command in SQL

```
DESCRIBE table_name;
```

Or

```
DESC table_name;
```

Example: DESC employee;

Output:

Object Type TABLE Object EMPLOYEE1									
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPLOYEE1	EMPNO	Number	-	5	0	-	✓	-	-
	ENAME	Varchar2	20	-	-	-	✓	-	-
	JOB	Char	10	-	-	-	✓	-	-
	HIREDATE	Date	7	-	-	-	✓	-	-
	SAL	Number	-	10	0	-	✓	-	-
	DEPTNO	Number	-	5	0	-	✓	-	-
1 - 6									

ALTER Command:

Syntax 1:

To add a column :

```
ALTER TABLE <table_name>  
ADD (column_name1 datatype(width),column_name2 datatype(width).....);
```

Example: Alter table employee

```
ADD(hobbyvarchar(10));
```

MONTHS_BETWEEN('05-JAN-2012','05-MAR-2012')	MONTHS_BETWEEN('05-MAR-2012','05-JAN-2012')
-2	2

Output:Table altered

Syntax 2:

To modify the width of a column:

```
ALTER TABLE <table_name>  
MODIFY (column_name1 datatype(width),column_name2 datatype(width).....);
```

Example: Alter table employee

```
MODIFY (hobby varchar(12));
```

Output: Table altered

Syntax 3:

To delete a column:

```
ALTER TABLE <table_name>  
DROP COLUMN column_name;
```

Example: Alter table employee

```
drop column hobby;
```

Output: Table dropped

DROP COMMAND:

drop or remove permanently the structure and records of the table.

Syntax:

```
DROP table <table_name>;
```

Example: DROP table employee;

Output: Table dropped

Note: You are requested not to drop the table employee, since it will remain the acting table for all the examples. So create another table of your choice and then try to drop the table.

TRUNCATE COMMAND:

Syntax:

```
TRUNCATE TABLE <table_name>;
```

Example : TRUNCATE TABLE employee;

Output: Table Truncated

RENAME COMMAND:

Syntax:

```
RENAME <existing table> to <new table>;
```

Example: RENAME employee to employee1;

Output : statement processed.

PRACTICE DML COMMANDS

INSERT COMMAND

Syntax 1 :

```
INSERT INTO <table_name>(columns to insert) VALUES(values to insert);
```

OR

```
INSERT INTO <tablename>values(value1,value2,value3.....);
```

Example1:

```
Insert into employee1 (empno , ename ,job,hiredate,sal,deptno)
values(10,'kabita','doctor','12-aug-2012',20000,101);
```

Output: 1 row(s) inserted

Example2:

```
Insert into employee1 (empno , ename ,job,hiredate,sal,deptno)
values(20,'sonam','manager','01-aug-2012',40000,201);
```

Output:1 row(s) inserted

Example3:

```
Insert into employee1 (empno , ename ,job,hiredate,sal,deptno)
values(30,'pranav','clerk','01-jan-2012',30000,301);
```

Output:1 row(s) inserted

How can you view these rows ?

Example:

Select * from employee1;

Output :

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
10	kabita	doctor	12-AUG-12	20000	101
20	sonam	engineer	01-AUG-12	40000	201
30	pranav	clerk	01-JAN-12	15000	301

Syntax 2:

Using substitution variable using CUI(Command User Interface)

```
SQL> create table info(name varchar2(10),rollno varchar2(10),place varchar2(10))  
;  
Table created.
```

```
SQL> insert into info values('&name','&rollno','&place');
```

```
Enter value for name: kabita  
Enter value for rollno: 83  
Enter value for place: sbp  
old 1: insert into info values('&name','&rollno','&place')  
new 1: insert into info values('kabita','83','sbp')  
  
1 row created.  
  
SQL> /  
Enter value for name: sipra  
Enter value for rollno: 89  
Enter value for place: Dkl  
old 1: insert into info values('&name','&rollno','&place')  
new 1: insert into info values('sipra','89','Dkl')  
  
1 row created.  
  
SQL> /  
Enter value for name: amita  
Enter value for rollno: 32  
Enter value for place: banglore  
old 1: insert into info values('&name','&rollno','&place')  
new 1: insert into info values('amita','32','banglore')  
  
1 row created.  
  
SQL> select * from info;  
  
NAME          ROLLNO    PLACE  
-----  
kabita        83        sbp  
sipra         89        Dkl  
amita         32        banglore
```

UPDATE COMMAND

Syntax 1:

```
UPDATE <table_name> SET <field_name>=value;
```

Example:

```
UPDATE employee1 set sal=25000;
```

Output: 3 row(s) updated.

Before updation:

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
10	kabita	doctor	12-AUG-12	20000	101
20	sonam	engineer	01-AUG-12	40000	201
30	pranav	clerk	01-JAN-12	15000	301

After updation: all the values of SAL column updated.

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
10	kabita	doctor	12-AUG-12	25000	101
20	sonam	engineer	01-AUG-12	25000	201
30	pranav	clerk	01-JAN-12	25000	301

OR

Syntax 2:

```
UPDATE <table_name> SET <field_name> = value where <field name> = value;
```

Example:

```
UPDATE employee1 set sal=25000 where deptno=101;
```

Before updation:

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
10	kabita	doctor	12-AUG-12	20000	101
20	sonam	engineer	01-AUG-12	40000	201
30	pranav	clerk	01-JAN-12	15000	301

After updation:only that row gets updated whose condition satisfied.

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
10	kabita	doctor	12-AUG-12	25000	101
20	sonam	engineer	01-AUG-12	40000	201
30	pranav	clerk	01-JAN-12	15000	301

DELETE COMMAND:

Syntax 1:

```
DELETE FROM <table_name>;
```

This command will delete all rows from the table.

Example: DELETE FROM employee1;

Syntax 2:

```
DELETE FROM <table_name> WHERE <condition>;
```

Example:

DELETE from employee1 where empno=10;

Output: 1 row(s) deleted

Before deletion:

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
10	kabita	doctor	12-AUG-12	25000	101
20	sonam	engineer	01-AUG-12	40000	201
30	pranav	clerk	01-JAN-12	15000	301

After deletion:

Select * from employee1;

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
20	sonam	engineer	01-AUG-12	40000	201
30	pranav	clerk	01-JAN-12	15000	301

PRACTICE DRL/DQL COMMANDS

DRL/DQL statement is used to access or retrieve data from the database.

In order to perform select operation, insert some more rows into the table.

```
insert into employee1 values(40,'raghav','analyst','01-feb-2012',40000,401);
```

```
insert into employee1 values(50,'rajiv','clerk','01-mar-2012',15000,501);
```

```
insert into employee1 values(60,'rahul','doctor','01-may-2012',35000,601);
```

```
insert into employee1 values(70,'sabita','engineer','01-jun-2012',40000,701);
```

DRL/DQL command :

Syntax 1:

```
Select <column_name1,column_name2.....> from<table_name>;
```

Example: select empno,ename from employee1;

Output:

EMPNO	ENAME
40	raghav
70	sabita
60	rahul
20	sonam
30	pranav
50	rajiv

Syntax 2:

```
Select * FROM <table_name>;
```

Example: select * from employee1;

Output:

Syntax 3:

Select * from <table_name> WHERE <condition>;

Example: select * from employee1 where sal=15000;

Output:

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
30	pranav	clerk	01-JAN-12	15000	301
50	rajiv	clerk	01-MAR-12	15000	501

Selecting Distinct Rows:

To prevent the selection of duplicate rows, we include Distinct clause in the select command.

Example:

Select job from employee1;

Output:

JOB
analyst
engineer
doctor
engineer
clerk
clerk

After applying the distinct clause:

Example:

Select distinct job from employee1;

Output:

JOB
doctor
clerk
analyst
engineer

Select Command With Where Clause :

The where clause is used along with select statement to specify the condition, based on which the rows will be extracted from a table with select.

Operators are used to specify the conditions:

Relational operator	meaning
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>,!=,^=	Not equal to

Logical operator	Meaning
And	Logical AND
Or	Logical OR
not	Logical not

Special operator	meaning
IN	Checking a value in a set
BETWEEN	Checking a value within a range
LIKE	Matching a pattern from a column

Example 1: To list the employees belonging to the job clerk.

Statement:

```
select * from employee1 where job='clerk';
```

Output:

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
30	pranav	clerk	01-JAN-12	15000	301
50	rajiv	clerk	01-MAR-12	15000	501

Example 2: To list the name and salary of the employees whose salary is more than 15000.

Statement:

```
Select * from employee1 where sal>15000;
```

Output:

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
40	raghav	analyst	01-FEB-12	40000	401
70	sabita	engineer	01-JUN-12	40000	701
60	rahul	doctor	01-MAY-12	35000	601
20	sonam	engineer	01-AUG-12	40000	201

Example 3:

```
Select * from employee1 where empno<=40;
```

Output:

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
40	raghav	analyst	01-FEB-12	40000	401
20	sonam	engineer	01-AUG-12	40000	201
30	pranav	clerk	01-JAN-12	15000	301

Example 4:

Select empno,ename,hiredate from employee1 where deptno>=500;

Output:

EMPNO	ENAME	HIREDATE
70	sabita	01-JUN-12
60	rahul	01-MAY-12
50	rajiv	01-MAR-12

Example 5:

Select ename,deptno from employee1 where deptno != 301;

Output:

ENAME	DEPTNO
raghav	401
sabita	701
rahul	601
sonam	201
rajiv	501

Example 6:

Select ename from employee1 where job = 'clerk' and deptno = 301;

Output:

ENAME
pranav

Example 7:

Select empno,ename,job from employee1 where sal <= 45000 or job = 'engineer';

Output:

EMPNO	ENAME	JOB
40	raghav	analyst
70	sabita	engineer
60	rahul	doctor
20	sonam	engineer
30	pranav	clerk
50	rajiv	clerk

Example 8:

Select ename,sal from employee1 where deptno in(301,401);

Output:

ENAME	SAL
raghav	40000
pranav	15000

Example 9:

Select ename ,sal,hiredate from employee1 where deptno between 300 and 700;

Output:

ENAME	SAL	HIREDATE
raghav	40000	01-FEB-12
rahul	35000	01-MAY-12
pranav	15000	01-JAN-12
rajiv	15000	01-MAR-12

Matching a pattern with a column from a table:

The LIKE operator is need only with CHAR , VARCHAR,VARCHAR2 to match the pattern.

'%' represents any number of unknown characters.

'_' denotes one unknown character.

Both '%' and '_' used with the LIKE operator to specify a pattern.

Example: '%'

select ename from employee1 where ename like 'r%';

Output:

ENAME
raghav
rahul
rajiv

Example: _ (underscore)

Note: use 5 no of underscore for the example below.

select hiredate,ename,sal from employee1 where ename like '_____';

HIREDATE	ENAME	SAL
01-MAY-12	rahul	35000
01-AUG-12	sonam	40000
01-MAR-12	rajiv	15000

Example:

select ename,deptno,empno from employee1 where ename like '_o%';

Output:

ENAME	DEPTNO	EMPNO
sonam	201	20

ALIAS TO THE COLUMN NAME:

It means giving duplicate name to the column name.

Example:

```
select empno as "employee no",ename as "employee name", sal as "salary" from employee1;
```

Output:

Employee No	Employee Name	Salary
40	raghav	40000
70	sabita	40000
60	rahul	35000
20	sonam	40000
30	pranav	15000
50	rajiv	15000

Ordering the results of a query:

ORDER BY clause is used with SELECT statement to impose an order on the result of a query.

Syntax:

```
SELECT[DISTINCT]<column list> from <table> [where condition][ORDER BY<COLUMN>][ASC/DESC];
```

NOTE: ascending is the default.order by clause must always be the last clause in the SELECT statement.

Example1:

```
select * from employee1 order by empno,deptno;
```

Output:

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
20	sonam	engineer	01-AUG-12	40000	201
30	pranav	clerk	01-JAN-12	15000	301
40	raghav	analyst	01-FEB-12	40000	401
50	rajiv	clerk	01-MAR-12	15000	501
60	rahul	doctor	01-MAY-12	35000	601
70	sabita	engineer	01-JUN-12	40000	701

Example 2:

select ename,hiredate as "date of joining" from employee1 order by "date of joining" desc;

Output:

ENAME	Date Of Joining
sonam	01-AUG-12
sabita	01-JUN-12
rahul	01-MAY-12
rajiv	01-MAR-12
raghav	01-FEB-12
pranav	01-JAN-12

Ordering output by column number:

In place of column names,we can use numbers to indicate the fields used to order the output.

Syntax:

Select <column_lists> from <table_name> order by column _number;

Example:

select empno,ename,job from employee1 order by 1;

Output:

EMPNO	ENAME	JOB
20	sonam	engineer
30	pranav	clerk
40	raghav	analyst
50	rajiv	clerk
60	rahul	doctor
70	sabita	engineer

Note: null values come at the end of the table in case of ORDER BY clause.

PRACTICE KEY CONSTRAINTS

CONSTRAINTS:

Constraints are the rules to control the data in a column can be applied at the column level or table level.

Column constraints apply only to individual columns.

Where as table constraints apply to group or more columns.

Note: The constraints are declared at the time of creating a table with the create table command or you can use alter table statement to add constraints to your existing table;

The list of constraints:

Constraints	Meaning
NOT NULL	Prevent a column accepting null values.
UNIQUE	Ensures uniqueness of the values in a column.
PRIMARY KEY	Same as unique, but only one column per table. It combines both the constraints unique and Not Null.
CHECK	Controls the value of a column(s) being inserted.
DEFAULT	Assigns a default value for the column(s), at the time of insertion when no value is given for that column.
REFERENCES	Assigns foreign key constraint to maintain "referential integrity".

CREATING TABLES WITH CONSTRAINTS

NOT NULL

Example:

```
Create table stud(rollno number(6) not null, name varchar2(10), branch varchar2(6));
```

```
Desc stud;
```

Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
Number	-	6	0	-	-	-	-
Varchar2	10	-	-	-	✓	-	-
Varchar2	6	-	-	-	✓	-	-

insert into stud values(1,'puja','IT');

Output:

1 row(s)inserted

Select * from stud;

Output:

ROLLNO	NAME	BRANCH
1	puja	IT

UNIQUE

Create table stud1(rollno number(6) unique, name varhcar2(10),branch varchar2(6));

insert into stud1 values(1,'mona','mech');

Output:

1 row(s)inserted

select * from stud1;

Output:

ROLLNO	NAME	BRANCH
1	mona	mech

insert into stud1 values(1,'siya','civil');

Output:

ORA-00001: unique constraint (DIBYA42.SYS_C003900) violated

From this we get the conclusion that unique constraint allows unique values only.

Note: It also allows null value.

PRIMARY KEY:

Create table stud2(rollno number(6) primary key, name varchar2(10),branch varchar2(6));

Desc stud2;

Output:

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STUD2	ROLLNO	Number	-	6	0	1	-	-	-
	NAME	Varchar2	10	-	-	-	✓	-	-
	BRANCH	Varchar2	6	-	-	-	✓	-	-
1 - 3									

insert into stud2 values(1,'pankaj','etc');

Output:

1 row(s)inserted

Insert into stud2 values(1,'madhu','IT');

Output:

```
ORA-00001: unique constraint (DIBYA42.SYS_C003901) violated
```

Insert into stud2 values(null,'priya','cse');

Output:

```
ORA-01400: cannot insert NULL into ("DIBYA42"."STUD2"."ROLLNO")
```

Conclusion: primary key doesnot allow null and duplicate values.

CHECK

Requirement condition can be given through the check constraint.

Create table employee (empno number(6),ename varchar2(10),sal number(10) check(sal between 10000 and 50000));

Output:

Table created.

insert into employee values(11,'rupak',20000);

Output:

1 row(s)inserted

insert into employee values(11,'yash',5000);

Output:

ORA-02290: check constraint (DIBYA42.SYS_C003902) violated

Conclusion: The check constraint consist of the keyword CHECK followed by parenthesized condition.any attempt to update or insert column values that will make the condition false,will be rejected.

DEFAULT

while inserting a row into a table without having values for every column, SQL must insert a default value to fill in the excluded column.

Example:

create table stud3(name varchar2(10),section char(2) default 'A',place varchar2(10));

Output:

Table created.

desc stud3;

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STUD3	NAME	Varchar2	10	-	-	-	✓	-	-
	SECTION	Char	2	-	-	-	✓	'A'	-
	PLACE	Varchar2	10	-	-	-	✓	-	-
1 - 3									

insert into stud3(name,place) values('ravi','bbsr');

Output:

1 row(s) inserted

Select * from stud3;

Output:

NAME	SECTION	PLACE
ravi	A	bbsr

Conclusion:

When a record is added into a table, and the default column is left empty, then the column will be automatically loaded with default value.

REFERENCES :

Referential integrity is a system of rules that a DBMS uses to ensure that relationships between record in related tables are valid, and that users don't accidentally delete or change related data. referential integrity is ensured through foreign key constraint.

Example:

```
create table admission(stuid number(6)primary key,sname varchar2(15),per number(5));
```

```
insert into admission values(1,'abhisek',80);
```

```
insert into admission values(2,'rohit',89);
```

```
insert into admission values(3,'sachin',89);
```

```
insert into admission values(4,'naveen',89);
```

```
select * from admission;
```

STUID	SNAME	PER
1	abhisek	80
2	rohit	89
3	sachin	99
4	naveen	70

```
create table course(stuid number(6) references admission(stuid),branch varchar2(5),sec varchar2(10));
```



```
insert into course values(1,'cse','a');
```

1 row(s)inserted

```
insert into course values(5,'cse','b');
```

Output:

```
ORA-02291: integrity constraint (DIBYA42.SYS_C003904) violated - parent key not found
```

```
delete from admission where stuid=1;
```

Output:

```
ORA-02292: integrity constraint (DIBYA42.SYS_C003904) violated - child record found
```

```
delete from course where stuid=1;
```

1 row(s)deleted

```
delete from admission where stuid=1;
```

1 row(s)deleted

```
select * from admission;
```

Output:

STUID	SNAME	PER
2	rohit	89
3	sachin	99
4	naveen	70

Conclusion:

The constraint establishes a relationship between primary key table to foreign key table.

PRACTICE FUNCTIONS

FUNCTIONS:

Note: The examples which are not based on the previous tables use the system table dual. Dual is a table which is automatically created by oracle along with the data dictionary. Dual table has one column and only one row with value 'x' which can be used to test many function.

Functions can be used to perform complex calculations on data.

Types of functions

Scalar functions(single row functions)

Group functions(Aggregate functions)

Scalar functions:

Number functions:

accepts numeric input and returns numeric values as the output.

1.Absolute function: It returns the absolute value of data.

Syntax: ABS(data)

Example: select ABS(-200) from Dual;

Output:

ABS(-200)
200

2.CEIL function: It returns the smallest integer greater than or equal to data

Syntax: CEIL(data)

Example: select CEIL(74.5)from Dual;

Output:

CEIL(74.55)
75

3.FLOOR function: It returns the largest integer less than or equal to data.

Syntax: FLOOR(data)

Example: select FLOOR(74.5)from Dual;

Output:

FLOOR(74.5)
74

4.MOD function: It returns the modulus or remainder.

Syntax: MOD(data,y)

Example: select MOD(11,2)from Dual;

Output:

MOD(11,2)
1

5.POWER function: It returns the data raised to the power of y..

Syntax: POWER(data,y)

Example: select POWER(3,2)from Dual;

Output:

POWER(3,2)
9

6.ROUND function: It rounds up the data to the specified number of decimal places.

Syntax: ROUND(data,n)

Example: select ROUND(123.5,0)from Dual;

Output:

ROUND(123.5,0)
124

Example: select ROUND(123.55,1)from Dual;

Output:

ROUND(123.55,1)
123.6

7.SQRT function:It returns the square root of data.

Syntax: SQRT(data)

Example: select SQRT(81)from Dual;

Output:

SQRT(81)
9

8.TRUNC function:It truncates the data to be specified number of decimal places.

Syntax:TRUNC(data,n)

Example: select TRUNC(123.56,1)from Dual;

Output:

TRUNC(123.56,1)
123.5

Similarly we have some more mathematical functions like cos(x), COSH(x), SIN(x), SINH(x), TAN(x), TANH(x), LN(data), LOG(b,data).

Character Functions: Accept character input and return either character or number values.

1.INITCAP: It will convert the first letter of the string to the Uppercase.

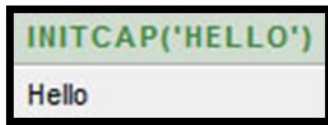
Syntax:

Initcap(char)

Example:

Select INITCAP('hello')from dual;

Output:

A screenshot of a SQL query and its result. The query is `INITCAP('HELLO')` and the result is `Hello`.

<code>INITCAP('HELLO')</code>
<code>Hello</code>

2.LENGTH: It return the length of the string.

Syntax:

<code>LENGTH(char)</code>

Example:

Select `length('hello')` from dual;

Output:

A screenshot of a SQL query and its result. The query is `LENGTH('HELLO')` and the result is `5`.

<code>LENGTH('HELLO')</code>
<code>5</code>

3.SUBSTR:It will extract the number of characters starting from the start position.

Syntax:

<code>SUBSTR(char,start_position,no_of_characters)</code>

Example:

Select `SUBSTR('UNIVERSITY',5,3)`;

Output:

A screenshot of a SQL query and its result. The query is `SUBSTR('UNIVERSITY',5,3)` and the result is `ers`.

<code>SUBSTR('UNIVERSITY',5,3)</code>
<code>ers</code>

4.GREATEST:It displays the greatest value from a group of expressions.

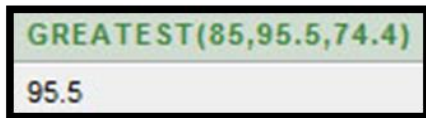
Syntax:

<code>GREATEST(epr1,expr2,.....exprn)</code>
--

Example:

Select `greatest(85,95.5,74.4)` from dual;

Output:



A screenshot of a SQL query result. The top row shows the query `GREATEST(85,95.5,74.4)` in green text. The bottom row shows the result `95.5` in black text.

5. LEAST: It displays the least value from a group of expressions.

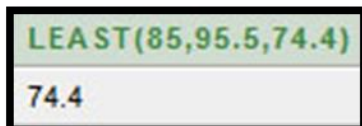
Syntax:

LEAST(expr1,expr2,.....exprn)

Example:

Select least(85,95.5,74.4) from dual;

Output:



A screenshot of a SQL query result. The top row shows the query `LEAST(85,95.5,74.4)` in green text. The bottom row shows the result `74.4` in black text.

6.LPAD:

Returns string1 padded on the left to length x with the characters in string2.

Syntax: lpad ('string1' , x , 'string2')

Example:

Select lpad('function',15,'*') from dual;

Output:



A screenshot of a SQL query result. The top row shows the query `LPAD('FUNCTION',15,'*')` in green text. The bottom row shows the result `*****function` in black text.

7.RPAD:

Returns string1 padded on the right to length x with the characters in string2.

Syntax: rpad('string1', 15 , 'string2')

Example:

Select rpad('function',15,'*') from dual;

Output:A screenshot of a SQL query and its output. The query is `RPAD('FUNCTION',15,'*')` and the output is `function*****`.

<code>RPAD('FUNCTION',15,'*')</code>
<code>function*****</code>

8.Lower

It convert string to lowercase.

Syntax:

<code>Lower('STRING')</code>

Example:

select lower('FUN')from dual;

Output:A screenshot of a SQL query and its output. The query is `LOWER('FUN')` and the output is `fun`.

<code>LOWER('FUN')</code>
<code>fun</code>

9.Upper

It convert string to uppercase.

Syntax:

<code>upper('STRING')</code>

Example:

select upper('fun')from dual;

Output:A screenshot of a SQL query and its output. The query is `UPPER('FUN')` and the output is `FUN`.

<code>UPPER('FUN')</code>
<code>FUN</code>

10.Translate: It translate one string to another string.

Syntax:

`Translate('string','from_str','to_str')`

Example:

```
select translate('abcdefgh','abcd','123')from dual;
```

Output:

TRANSLATE('ABCDEFGH','ABCD','123')
123efgh

11.Replace

It replace the part or whole of the string to a new string.

Syntax:

```
Replace('string','searchstring','replacestring')
```

Example:

```
select replace('supram','ram','ra')from dual;
```

Output:

REPLACE('SUPRAM','RAM','RA')
supra

DATE FUNCTIONS:**1.sysdate**

It returns the current date of the system.It takes no argument

Syntax:

Sysdate

Example:

```
select sysdate from dual;
```

Output:

SYSDATE
10-AUG-12

2.ADD_months: It add count 'months' to 'date'

Syntax: Add_months(date,count)

Example:

```
select add_months('4-mar-2012',2) from dual;
```

Output:

ADD_MONTHS('4-MAR-2012',2)
04-MAY-12

3.LAST_DAY

It gives date of the last date of the specified month in the date.

Syntax:

LAST_DAY(date)

Example:

```
select last_day('4-mar-2012')from dual;
```

Output:

LAST_DAY('4-MAR-2012')
31-MAR-12

4.months_between

It returns the number of months between two dates.

Syntax:

Months_between(d1,d2)

Note:If d1 is later than d2 the result is positive.if d1 is earlier than d2 the result is negative.

Example:

```
select months_between('05-jan-2012','05-mar-2012'), months_between('05-mar-2012','05-jan-2012')
from dual;
```

Output:

MONTHS_BETWEEN('05-JAN-2012','05-MAR-2012')	MONTHS_BETWEEN('05-MAR-2012','05-JAN-2012')
-2	2

5.next_day

It gives date of the next specified day of the week after the 'date',

Syntax:

Next_day(date,day)

Example:

select next_day('10-aug-2012','fri')from dual;

Output:

NEXT_DAY('10-AUG-2012','FRI')
17-AUG-12

6.TO_CHAR

This function converts the date 'd' to character format 'f'

Syntax:

To_char(sysdate,'day')

Example:

select sysdate,to_char(sysdate,'day')from dual;

Output:

SYSDATE	TO_CHAR(SYSDATE,'DAY')
10-AUG-12	friday

7.TO_DATE: This function converts the date into roman form

Syntax:

TO_DATE(date,'RM')

Example:

Output:

Group or Aggregate functions:

Aggregate functions perform a calculation on a set of values and return a single value. Except for COUNT, aggregate functions ignore null values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement. All aggregate functions are deterministic. This means aggregate functions return the same value any time that they are called by using a specific set of input values.

Note : Use the employee1 table for the group functions.

ENAME	JOB	HIREDATE	SAL	DEPTNO
raghav	analyst	01-FEB-12	40000	401
sabita	engineer	01-JUN-12	40000	701
rahul	doctor	01-MAY-12	35000	601
sonam	engineer	01-AUG-12	40000	201
pranav	clerk	01-JAN-12	15000	301
rajiv	clerk	01-MAR-12	15000	501

1.AVG

Syntax:

AVG(column_name)

Example:

```
select avg(sal) as "average salary" from employee1;
```

Output:

2.Count

It returns the number of rows in the result . It doesn't count the null values.

Syntax:

Count(* [distinct] column name)

Example1:

```
select count(*) from employee1;
```

Output:

COUNT(*)
6

Example2:

```
select count(distinct job) from employee1;
```

Output:

COUNT(DISTINCTJOB)
4

3.MAX

It is used to find the maximum value of the selected list of item.

Syntax:

Max(column_ name)

Example:

```
select max(sal)from employee1;
```

Output:

MAX(SAL)
40000

4.MIN

It is used to find the minimum value of the selected list of item.

Syntax:

MIN(column_name)

Example:

```
select min(sal)from employee1;
```

Output:

MIN(SAL)
15000

5.SUM

The sum function returns the sum of values for the select list of columns.

Syntax:

Sum([Distinct|All]column name)

Example:

```
select sum(sal)from employee1;
```

Output:

SUM(SAL)
185000

PRACTICE JOIN & VIEW

A **JOIN** operation is used when we want to retrieve data from two or more tables.

```
create table emp(empno number(5),ename varchar2(10),job varchar2(10),sal number(10),deptno number(5));
```

```
insert into emp values(8340,'bharti','salesman',5000,10);
```

```
insert into emp values(8341,'ajay','manager',10000,20);
```

```
insert into emp values(8344,'vijay','clerk',2500,30);
```

```
insert into emp values(8342,'sonia','president',15000,40);
```

```
select * from emp;
```

EMPNO	ENAME	JOB	SAL	DEPTNO
8344	vijay	clerk	2500	30
8340	bharti	salesman	5000	10
8341	ajay	manager	10000	20
8342	sonia	president	15000	40

```
create table dept(deptno number(5),dname varchar2(10),loc varchar2(10));
```

```
insert into dept values(20,'manager','mumbai');
```

```
insert into dept values(30,'clerk','bbsr');
```

```
insert into dept values(40,'president','new delhi');
```

```
insert into dept values(50,'sales','banglore'); insert into dept values(10,'sales','banglore');
```

```
select * from dept;
```

DEPTNO	DNAME	LOC
10	sales	banglore
40	president	new delhi
20	manager	mumbai
50	sales	banglore
30	clerk	bbsr

```
Create table salarygrade(grade number(10),losal number(10),hisal number(10));
```

```

insert into salarygrade values(1,700,1200);
insert into salarygrade values(2,1201,1400);
insert into salarygrade values(3,1401,2000);
insert into salarygrade values(4,2001,1000);
insert into salarygrade values(5,3001,9999);
select * from salarygrade;

```

GRADE	LOSAL	HISAL
2	1201	1400
4	2001	1000
1	700	1200
3	1401	2000
5	3001	9999

Syntax for the select statement where we join two tables:

Select<select_list> from <table1>,<table2>.....<table n>

Where <table1.column1>=<table2.column2>

And.....

<table2.column3>=<tablen.columnn>

.....additional conditions;

Perform equi joins, Cartesian joins, outer joins, self joins on the above two created tables.

Equi Join:

The join, in which columns are compared for equality .

Syntax:

Select *

From <table_name1>join <table_name2>

On(tablename1.column_name=tablename2.column_name);

Note: column_name should be common in both the table.

Example:

```
select * from emp join dept on (emp.deptno=dept.deptno);
```

Output:

EMPNO	ENAME	JOB	SAL	DEPTNO	DEPTNO	DNAME	LOC
8340	bharti	salesman	5000	10	10	sales	banglore
8342	sonia	president	15000	40	40	president	new delhi
8341	ajay	manager	10000	20	20	manager	mumbai
8344	vijay	clerk	2500	30	30	clerk	bbsr

Note: All the columns from joining table appear in the output even if they are identical.

Comparison operator (“=”) used in equi joins.

Non-Equi-Join:

A non equi join is a query that specifies some relationship other than equality between the column.

Example:

```
select ename,dname,job,sal,empno ,grade from emp e,dept d,salarygrade s where e.deptno=d.deptno and  
sal between losal and hisal order by e.deptno,empno;
```

Output:

ENAME	DNAME	JOB	SAL	EMPNO	GRADE
bharti	sales	salesman	5000	8340	5

Natural join:

The result of an equijoin contain two identical columns.one of the identical columns can be eliminated by restating the query.This result is called a natural join(equi join minus one of the two identical columns).

Syntax:

```
Select * from <table1> natural join <table2>;
```

If you restate the query of equijoin as:

```
select * from emp natural join dept;
```


Output:

DEPTNO	EMPNO	ENAME	JOB	SAL	DNAME	LOC
10	8340	bharti	salesman	5000	sales	banglore
40	8342	sonia	president	15000	president	new delhi
20	8341	ajay	manager	10000	manager	mumbai
30	8344	vijay	clerk	2500	clerk	bbsr

Left , Right Join:

When we display an equi-join or natural join ,it shows only the matched rows. what is we want to know which all rows from a table didnt match with other. In such a case left or right join can be very helpful and it refers to the order in which the tables are put together and the results are displayed.

Left join syntax:

Select <select-list>

From <table1> left join <table2>

On <joining-condition>;

Example:

```
select * from emp left join dept on emp.deptno=dept.deptno;
```

EMPNO	ENAME	JOB	SAL	DEPTNO	DEPTNO	DNAME	LOC
8340	bharti	salesman	5000	10	10	sales	banglore
8342	sonia	president	15000	40	40	president	new delhi
8341	ajay	manager	10000	20	20	manager	mumbai
8344	vijay	clerk	2500	30	30	clerk	bbsr

Right join syntax:

Select <select-list>

From <table1> right join <table2>

On <joining-condition>;

Example:

```
select * from emp right join dept on emp.deptno=dept.deptno;
```

Output:

EMPNO	ENAME	JOB	SAL	DEPTNO	DEPTNO	DNAME	LOC
8344	vijay	clerk	2500	30	30	clerk	bbsr
8340	bharti	salesman	5000	10	10	sales	banglore
8341	ajay	manager	10000	20	20	manager	mumbai
8342	sonia	president	15000	40	40	president	new delhi
-	-	-	-	-	50	sales	banglore

Note: Unmatched rows will be filled with null(-).

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 statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

CREATE VIEW SYNTAX

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS  
SELECT name, age  
FROM CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

This would produce the following result.

name	age
Ramesh	32
Khilan	25
kaushik	23
Chaitali	25
Hardik	27
Komal	22
Muffy	24

PRACTICE SUB-QUERY

SQL Sub Query

A Subquery is a query within another SQL query and embedded within the WHERE clause.

Important Rules:

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Subqueries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.
- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

Subqueries with the Select Statement

SQL subqueries are most frequently used with the Select statement.

Syntax

```
SELECT column_name  
FROM table_name  
WHERE column_name expression operator  
( SELECT column_name from table_name WHERE ... );
```

Example

Consider the EMPLOYEE table have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
6	Harry	42	China	4500.00
7	Jackson	25	Mizoram	10000.00

The subquery with a SELECT statement will be:

```
SELECT *  
FROM EMPLOYEE  
WHERE ID IN (SELECT ID  
FROM EMPLOYEE  
WHERE SALARY > 4500);
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
7	Jackson	25	Mizoram	10000.00

Subqueries with the INSERT Statement

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

Syntax:

```
INSERT INTO table_name (column1, column2, column3,...)  
SELECT *  
FROM table_name  
WHERE VALUE OPERATOR
```

Example

Consider a table EMPLOYEE_BKP with similar as EMPLOYEE.

Now use the following syntax to copy the complete EMPLOYEE table into the EMPLOYEE_BKP table.

```
INSERT INTO EMPLOYEE_BKP  
SELECT * FROM EMPLOYEE  
WHERE ID IN (SELECT ID  
FROM EMPLOYEE);
```

Subqueries with the UPDATE Statement

The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

Syntax

```
UPDATE table
SET column_name = new_value
WHERE VALUE OPERATOR
(SELECT COLUMN_NAME
FROM TABLE_NAME
WHERE condition);
```

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example updates the SALARY by .25 times in the EMPLOYEE table for all employee whose AGE is greater than or equal to 29.

```
UPDATE EMPLOYEE
SET SALARY = SALARY * 0.25
WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
WHERE AGE >= 29);
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	1625.00
5	Kathrin	34	Bangalore	2125.00
6	Harry	42	China	1125.00
7	Jackson	25	Mizoram	10000.00

Subqueries with the DELETE Statement

The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

Syntax

```
DELETE FROM TABLE_NAME  
WHERE VALUE OPERATOR  
(SELECT COLUMN_NAME  
FROM TABLE_NAME  
WHERE condition);
```

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example deletes the records from the EMPLOYEE table for all EMPLOYEE whose AGE is greater than or equal to 29.

```
DELETE FROM EMPLOYEE  
WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP  
WHERE AGE >= 29 );
```

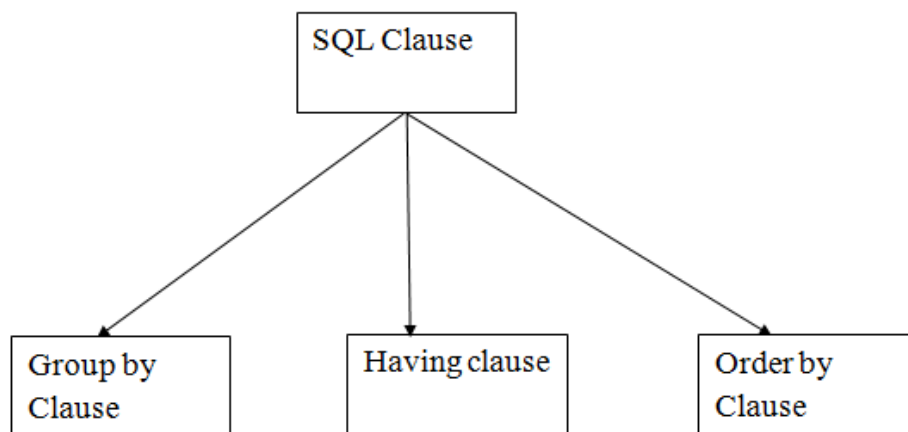
This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
7	Jackson	25	Mizoram	10000.00

PRACTICE SQL CLAUSES

SQL Clauses

The following are the various SQL clauses:



GROUP BY

- SQL GROUP BY statement is used to arrange identical data into groups. The GROUP BY statement is used with the SQL SELECT statement.
- The GROUP BY statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- The GROUP BY statement is used with aggregation function.

```
SELECT column  
FROM table_name  
WHERE conditions  
GROUP BY column  
ORDER BY column
```


Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example:

```
SELECT COMPANY, COUNT(*)  
FROM PRODUCT_MAST  
GROUP BY COMPANY;
```

Output:

```
Com1  5  
Com2  3  
Com3  2
```

HAVING

- HAVING clause is used to specify a search condition for a group or an aggregate.
- Having is used in a GROUP BY clause. If you are not using GROUP BY clause then you can use HAVING function like a WHERE clause.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE conditions
GROUP BY column1, column2
HAVING conditions
ORDER BY column1, column2;
```

Example:

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING COUNT(*)>2;
```

Output:

```
Com1  5
Com2  3
```

ORDER BY

- The ORDER BY clause sorts the result-set in ascending or descending order.
- It sorts the records in ascending order by default. DESC keyword is used to sort the records in descending order.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE condition
ORDER BY column1, column2... ASC|DESC;
```

Where

ASC: It is used to sort the result set in ascending order by expression.

DESC: It sorts the result set in descending order by expression.

Example: Sorting Results in Ascending Order

Table: CUSTOMER

CUSTOMER_ID	NAME	ADDRESS
12	Kathrin	US
23	David	Bangkok
34	Alina	Dubai
45	John	UK
56	Harry	US

Enter the following SQL statement:

```
SELECT *  
FROM CUSTOMER  
ORDER BY NAME;
```

Output:

CUSTOMER_ID	NAME	ADDRESS
34	Alina	Dubai
23	David	Bangkok
56	Harry	US
45	John	UK
12	Kathrin	US

Example: Sorting Results in Descending Order

Using the above CUSTOMER table

```
SELECT *  
FROM CUSTOMER  
ORDER BY NAME DESC;
```

Output:

CUSTOMER_ID	NAME	ADDRESS
12	Kathrin	US
45	John	UK
56	Harry	US
23	David	Bangkok
34	Alina	Dubai

PRACTICE TCL COMMANDS

A transaction is a logical unit of work that must succeed or fail in its entirety. It is an atomic operation which may not be divided into smaller operations.

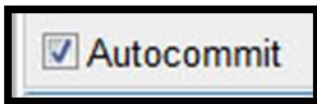
Each transaction involves one or more (DML) Data manipulation language statements (such as INSERT, DELETE OR UPDATE) and ends with either a commit to make the changes permanent or ROLLBACK to undo the changes.

TCL commands: Commit, Rollback

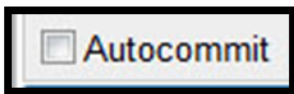
COMMIT - save work done

ROLLBACK - restore database to original since the last COMMIT

Commit:



By default autocommit is on. In order to understand the concept of commit command uncheck it.



```
create table item1(item_no number(10),item_name varchar2(10),price number(10));
```

Output:

table created.

```
insert into item1 values(10,'chocobar',30);
```

Output:

1 row(s) inserted.

```
select * from item1;
```

Output:

ITEM_NO	ITEM_NAME	PRICE
10	chocobar	30

Logout from the database



You are now logged out.

[Login](#)

Login again by using your own username and password.

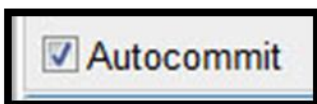
Select * from item1;

Output:

No data found.

Conclusion: if we uncheck the autocommit then no data saved.

Check it



insert into item1 values(10,'chocobar',30);

output:

1 row(s) created.

Select * from item1;

Output:

ITEM_NO	ITEM_NAME	PRICE
10	chocobar	30

Again logout from your database and login again to check whether your work has been saved or not.

Select * from item1;

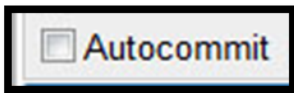
Output:

ITEM_NO	ITEM_NAME	PRICE
10	chocobar	30

Conclusion: commit command saved work

Rollback:

Uncheck the autocommit checkbox



insert into item1 values(30,'chips',20);

Output:

1 row(s)inserted.

Select * from item1;

ITEM_NO	ITEM_NAME	PRICE
20	biscuits	40
30	chips	20
10	chocobar	30

Now i don't want the row which i have already inserted.

Use the command

Rollback;**Output:**

Statement processed.

select * from item1;

Output:

ITEM_NO	ITEM_NAME	PRICE
20	biscuits	40
10	chocobar	30

Conclusion: changes made to the database are undone by rollback.

Savepoints in (CUI)

Savepoints are special operations that allow you to divide the work of a transaction into different segments.

```
SQL> create table info(name varchar2(10),rollno varchar2(10),place varchar2(10))  
;  
Table created.
```



```

SQL> insert into info values('kabita','42','sbp');
1 row created.
SQL> select * from info;
NAME          ROLLNO        PLACE
-----
kabita        42            sbp
SQL> insert into info values('sipra','83','dkl');
1 row created.
SQL> select * from info;
NAME          ROLLNO        PLACE
-----
kabita        42            sbp
sipra         83            dkl
SQL> savepoint s1;
Savepoint created.
SQL> insert into info values('amita','12','banglore');
1 row created.
SQL> select * from info;
NAME          ROLLNO        PLACE
-----
kabita        42            sbp
sipra         83            dkl
amita         12            banglore
SQL> rollback to s1;
Rollback complete.
SQL> select * from info;
NAME          ROLLNO        PLACE
-----
kabita        42            sbp
sipra         83            dkl

```

PRACTICE DCL COMMANDS

DCL Commands: Grant, Revoke

GRANT - gives user's access privileges to database

REVOKE - withdraw access privileges given with the GRANT command

EXAMPLE

CREATING A USER

SQL>CONNECT SYSTEM;

ENTER PASSWORD:KISTCSE

SQL>CREATE USER MANISHA IDENTIFIED BY IT;

SQL>GRANT DBA TO MANISHA;

SQL>CONNECT MANISHA/IT;

SQL>REVOKE DBA FROM MANISHA;

```
SQL> connect system
Enter password:
Connected.
SQL> create user manisha identified by IT;

User created.

SQL> grant dba to manisha;

Grant succeeded.

SQL> connect manisha/it;
Connected.
SQL> revoke dba from manisha;

Revoke succeeded.

SQL>
```

PRACTICE PL/SQL

PL/SQL Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

- Header: The header contains the name of the procedure and the parameters or variables passed to the procedure.
- Body: The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

How to pass parameters in procedure:

When you want to create a procedure or function, you have to define parameters. There are three ways to pass parameters in procedure:

1. IN parameters: The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
2. OUT parameters: The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. INOUT parameters: The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

PL/SQL Create Procedure

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[ (parameter [,parameter]) ]
IS
[declaration_section]
BEGIN
executable_section
[EXCEPTION
exception_section]
END [procedure_name];
```

Create procedure example

In this example, we are going to insert record in user table. So you need to create user table first.

Table creation:

```
create table user(id number(10) primary key,name varchar2(100));
```

Now write the procedure code to insert record in user table.

Procedure Code:

```
create or replace procedure "INSERTUSER"  
(id IN NUMBER,  
 name IN VARCHAR2)  
is  
begin  
insert into user values(id,name);  
end;  
/
```

Output:

```
Procedure created.
```

PL/SQL program to call procedure

Let's see the code to call above created procedure.

```
BEGIN  
insertuser(101,'Rahul');  
dbms_output.put_line('record inserted successfully');  
END;  
/
```

Now, see the "USER" table, you will see one record is inserted.

ID	Name
101	Rahul

PL/SQL Drop Procedure

Syntax for drop procedure

```
DROP PROCEDURE procedure_name;
```

Example of Drop procedure

```
DROP PROCEDURE pro1;
```

PL/SQL Function

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

Syntax to create a function:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
    < function_body >
END [function_name];
```

- Function_name: specifies the name of the function.
- [OR REPLACE] option allows modifying an existing function.
- The optional parameter list contains name, mode and types of the parameters.
- IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

The function must contain a return statement.

- RETURN clause specifies that data type you are going to return from the function.
- Function_body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

PL/SQL Function Example

Let's see a simple example to create a function.

```
create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
/
```

Now write another program to call the function.

```
DECLARE
  n3 number(2);
BEGIN
  n3 := adder(11,22);
  dbms_output.put_line('Addition is: ' || n3);
END;
/
```

Output:

```
Addition is: 33
Statement processed.
0.05 seconds
```

PRACTICE CURSORS

PL/SQL Cursor

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

- Implicit Cursors
- Explicit Cursors

PL/SQL Implicit Cursors

The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement.

These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.

Oracle provides some attributes known as Implicit cursor's attributes to check the status of DML operations. Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.

For example: When you execute the SQL statements like INSERT, UPDATE, DELETE then the cursor attributes tell whether any rows are affected and how many have been affected. If you run a SELECT INTO statement in PL/SQL block, the implicit cursor attribute can be used to find out whether any row has been returned by the SELECT statement. It will return an error if there no data is selected.

The following table soecifies the status of the cursor with each of its attribute.

Attribute	Description
%FOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE.
%NOTFOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise it returns FALSE. It is a just opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements.
%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement.

PL/SQL Implicit Cursor Example

Create customers table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Let's execute the following program to update the table and increase salary of each customer by 5000. Here, SQL%ROWCOUNT attribute is used to determine the number of rows affected:

Create procedure:


```

DECLARE
    total_rows number(2);
BEGIN
    UPDATE customers
    SET salary = salary + 5000;
    IF sql%notfound THEN
        dbms_output.put_line('no customers updated');
    ELSIF sql%found THEN
        total_rows := sql%rowcount;
        dbms_output.put_line( total_rows || ' customers updated ');
    END IF;
END;
/

```

Output:

```

6 customers updated
PL/SQL procedure successfully completed.

```

PL/SQL procedure successfully completed.

Now, if you check the records in customer table, you will find that the rows are updated.

```
select * from customers;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	25000
2	Suresh	22	Kanpur	27000
3	Mahesh	24	Ghaziabad	29000
4	Chandan	25	Noida	31000
5	Alex	21	Paris	33000
6	Sunita	20	Delhi	35000

PL/SQL Explicit Cursors

The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Following is the syntax to create an explicit cursor:

Syntax of explicit cursor

Following is the syntax to create an explicit cursor:

```
CURSOR cursor_name IS select_statement;;
```

Steps:

You must follow these steps while working with an explicit cursor.

1. Declare the cursor to initialize in the memory.
2. Open the cursor to allocate memory.
3. Fetch the cursor to retrieve data.
4. Close the cursor to release allocated memory.

1) Declare the cursor:

It defines the cursor with a name and the associated SELECT statement.

Syntax for explicit cursor declaration

```
CURSOR name IS  
SELECT statement;
```

2) Open the cursor:

It is used to allocate memory for the cursor and make it easy to fetch the rows returned by the SQL statements into it.

Syntax for cursor open:

```
OPEN cursor_name;
```

3) Fetch the cursor:

It is used to access one row at a time. You can fetch rows from the above-opened cursor as follows:

Syntax for cursor fetch:

```
FETCH cursor_name INTO variable_list;
```

4) Close the cursor:

It is used to release the allocated memory. The following syntax is used to close the above-opened cursors.

Syntax for cursor close:

```
Close cursor_name;
```

PL/SQL Explicit Cursor Example

Explicit cursors are defined by programmers to gain more control over the context area. It is defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Let's take an example to demonstrate the use of explicit cursor. In this example, we are using the already created CUSTOMERS table.

Create customers table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create procedure:

Execute the following program to retrieve the customer name and address.

```
DECLARE
  c_id customers.id%type;
  c_name customers.name%type;
  c_addr customers.address%type;
  CURSOR c_customers IS
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
    FETCH c_customers INTO c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
```

Output:

```
1 Ramesh Allahabad
2 Suresh Kanpur
3 Mahesh Ghaziabad
4 Chandan Noida
5 Alex Paris
6 Sunita Delhi
PL/SQL procedure successfully completed.
```

PRACTICE EXCEPTION HANDLING

PL/SQL Exception Handling

What is Exception

An error occurs during the program execution is called Exception in PL/SQL.

PL/SQL facilitates programmers to catch such conditions using exception block in the program and an appropriate action is taken against the error condition.

There are two type of exceptions:

- System-defined Exceptions
- User-defined Exceptions

Syntax for Exception handling:

Following is a general syntax for Exception handling:

```
DECLARE
    <declarations section>
BEGIN
    <executable command(s)>
EXCEPTION
    <exception handling goes here >
    WHEN exception1 THEN
        exception1-handling-statements
    WHEN exception2 THEN
        exception2-handling-statements
    WHEN exception3 THEN
        exception3-handling-statements
    .....
    WHEN others THEN
        exception3-handling-statements
END;
```

Example of Exception handling

Let's take a simple example to demonstrate the concept of exception handling. Here we are using the already created CUSTOMERS table.

```
SELECT* FROM COUSTOMERS;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

```

DECLARE
c_id customers.id%type := 8;
c_name customers.name%type;
c_addr customers.address%type;
BEGIN
    SELECT name, address INTO c_name, c_addr
    FROM customers
    WHERE id = c_id;
    DBMS_OUTPUT.PUT_LINE ('Name: ' || c_name);
    DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
EXCEPTION
    WHEN no_data_found THEN
        dbms_output.put_line('No such customer!');
    WHEN others THEN
        dbms_output.put_line('Error!');
END;
/

```

After the execution of above code at SQL Prompt, it produces the following result:

```

No such customer!
PL/SQL procedure successfully completed.

```

The above program should show the name and address of a customer as result whose ID is given. But there is no customer with ID value 8 in our database, so the program raises the run-time exception NO_DATA_FOUND, which is captured in EXCEPTION block.

If you use the id defined in the above table (i.e. 1 to 6), you will get a certain result. For a demo example: here, we are using the id 5.

```

DECLARE
c_id customers.id%type := 5;
c_name customers.name%type;
c_addr customers.address%type;
BEGIN
    SELECT name, address INTO c_name, c_addr
    FROM customers
    WHERE id = c_id;
    DBMS_OUTPUT.PUT_LINE ('Name: ' || c_name);
    DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
EXCEPTION
    WHEN no_data_found THEN
        dbms_output.put_line('No such customer!');
    WHEN others THEN
        dbms_output.put_line('Error!');
END;
/

```

After the execution of above code at SQL prompt, you will get the following result:

```
Name: alex  
Address: paris  
PL/SQL procedure successfully completed.
```

Raising Exceptions

In the case of any internal database error, exceptions are raised by the database server automatically. But it can also be raised explicitly by programmer by using command RAISE.

Syntax for raising an exception:

```
DECLARE  
    exception_name EXCEPTION;  
BEGIN  
    IF condition THEN  
        RAISE exception_name;  
    END IF;  
EXCEPTION  
    WHEN exception_name THEN  
        statement;  
END;
```

PL/SQL User-defined Exceptions

PL/SQL facilitates their users to define their own exceptions according to the need of the program. A user-defined exception can be raised explicitly, using either a RAISE statement or the procedure DBMS_STANDARD.RAISE_APPLICATION_ERROR.

Syntax for user define exceptions

```
DECLARE  
my-exception EXCEPTION;
```

PRACTICE TRIGGER

Trigger is invoked by Oracle engine automatically whenever a specified event occurs. Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Syntax for creating trigger:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
    Declaration-statements
BEGIN
    Executable-statements
EXCEPTION
    Exception-handling-statements
END;
```

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Trigger Example

Let's take a simple example to demonstrate the trigger. In this example, we are using the following CUSTOMERS table:

Create table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create trigger:

Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
    sal_diff number;
BEGIN
    sal_diff := :NEW.salary - :OLD.salary;
    dbms_output.put_line('Old salary: ' || :OLD.salary);
    dbms_output.put_line('New salary: ' || :NEW.salary);
    dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

After the execution of the above code at SQL Prompt, it produces the following result.

```
Trigger created.
```

ASSIGNMENT- 1

Q.1 create the tables described below:

Table Name1: Client_Master			Table Name2: Product_master		
COLUMN NAME	DATA TYPE	SIZE	Column name	Data type	Size
Client_no	Varchar2	6	Product_no	Varchar2	6
Name	Varchar2	20	Description	Varchar2	15
Address1	Varchar2	30	Profit_percent	Number	4,2
Address2	Varchar2	30	Unit_measure	Varchar2	10
City	Varchar2	15	Qty_on_Hand	Number	8
Pincode	Number	8	Re_order_lvl	Number	8
State	Varchar2	15	Sell_price	Number	8,2
Baldue	Number	10,2	Cost_price	Number	8,2

Table Name3: Salesman_master

Column name	Data type	Size
Sales_man_no	Varchar2	6
Sales_man_name	Varchar2	20
Address1	Varchar2	30
Address2	Varchar2	30
City	Varchar2	20
Pincode	Number	8
State	Varchar2	20
Sal_amt	Number	8,2
TGTTGET	Number	6,2
Ytd_sales	Number	6,2
Remarks	Varchar2	60

Q.2 Insert the following data into their respective tables:

A) Data for CLIENT_MASTER table:

Client_no	Name	City	Pincode	State	Baldue
C00001	Rashmi Dash	Sundargarh	700054	Odisha	15000
C00002	Surya Nanda	Sambalpur	780001	Odisha	0
C00003	Golu Acharya	Rourkela	400057	Odisha	5000
C00004	Mukesh Sharma	Bhubaneswar	560001	Odisha	0

C00005	Bhawani Bansal	Cuttack	400060	Odisha	2000
C00006	Kamal Thaty	Bolangir	560050	Odisha	0

B) Data for **PRODUCT_MASTER** table:

Product_no	Description	Profit Percent	Unit Measure	QtyOn Hand	Reorder LVL	Sell Price	Cost Price
P00001	T-Shirts	5	Piece	200	50	350	250
P0345	Shirts	6	Piece	150	50	500	350
P06754	Cotton Jeans	5	Piece	100	20	600	450
P07865	Jeans	5	Piece	100	20	750	500
P07868	Trousers	2	Piece	150	50	850	550
P07885	Pull Overs	2.5	Piece	80	30	700	450
P07965	Denim Shirts	4	Piece	100	40	350	250
P07975	Lycra Tops	5	Piece	70	30	300	175
P08865	Skirts	5	Piece	75	30	450	300

C) Data for **SALESMAN_MASTER** table:

Salesman_no	Name	Address1	Address2	City	Pincode	State	Sal_amt	Tgt To Get	Ytd Sales	Remarks
S00001	Aman	A/14	Sahid nagar	BBSR	400002	Odisha	3000	100	50	Good
S00002	Omkar	65	Burla	SBP	400001	Odisha	3000	200	100	Good
S00003	Raj	P-7	Ainthapali	SBP	400032	Odisha	3000	200	100	Good
S00004	Ashish	A/5	Sect. -2	RKL	400044	Odisha	3500	200	150	Good

Q.3 Exercise on retrieving record from table:

- Find out the names of all the clients.
- Retrieve the entire contents of the Client_Master table.
- Retrieve the list of names, city and the state of all the clients.
- List the various products available from the Product_Master table.
- List all the clients who are located in SBP.
- Find the names of salesmen who have a salary equals or more than Rs.3000.

ASSIGNMENT- 2

Write the following queries (1-14) based on schema as follows:

Product (maker, model, type)

Pc (model, speed, ram, hd, rd, price)

Laptop (model, speed, ram, hd, screen, price)

Priter (model , colour, type, price)

1. Find the model number, speed and hard disk size for all pc's whose price is under 12000.
2. Find the products cost more than 2000.
3. Find the menu feature of printers
4. Find the model number , memory size and screen size for laptops Costing more then 2000.
5. Find the model number speed and hard-disk size for those pc's that have either a 12x or 16x DVD and price less than 2000, you may required the rd attribute as having a string type.
6. Give the manufacture and speed of laptops with a hard disk of least thirty gigabytes.
7. Find the model number and price of all products made by manufactures.
8. Find those manufactures that sell laptops out not PC's.
9. Find the average speed of PC's.
10. Find the average speed of laptops costing over 2000.
11. Find the model number, speed and hard disk size for all PC's whose Price is under 1200.
12. Find the printer with highest price.
13. Find the laptops whose speed is slower than that of any PC.
14. Find the maker of the color printer with the lowest price.

ASSIGNMENT- 3

Given bellow Employee Database which has 5 relation schemas namely EMPLOYEE, PROJECT, EMPLOYEE_PROJECT, DEPARTMENT, DEPENDENT as given bellow.

EMPLOYEE							
<u>Emp ID</u>	EName	BossID	DeptID	Basic	Grade Pay	Specialization	PhoneNo
E001	R. K Pati	E004	D01	27000	2000	Sr. Manager	9338250001
E002	D. K. Mallick	E001	D01	12000	1000	Ex. Officer	9338250002
E003	P.K. Patra	E005	D02	12000	1000	Ex. Officer	9338250003
E004	A. Nanda	---	D01	50000	2000	General Mgr.	9338250004
E005	A. Bag	E004	D02	26000	2000	Sr. Manager	9338250005
E006	B.P. Nag	E004	D03	28000	2000	Sr. Manager	9338250006
E007	S.K. Souri	E001	D01	16000	1200	Sr. Ex. Officer	9338250007
E008	D.J. Das	E005	D02	18000	1200	Sr. Ex. Officer	9338250008
E009	R.K. Jadon	E006	D03	20000	1200	Jr. Manager	9338250009
E010	R.R. Suman	E006	D04	25000	2000	Jr. Manager	9338250010
E011	S.J. Kumar	E010	D04	17000	1200	Ex. Officer	9338250011
E012	K.K Till	E006	D03	11000	800	Finance Offcr.	9338250012

PROJECT							
<u>Proj ID</u>	Proj Name	Funding Agency	PrinInvst ID	Dept ID	CostOf Proj	DateOf Start	Status
P001	Project 1	AM Steels	E001	D01	6,50,000	13-JUN-2009	Running
P002	Project 2	KP Production	E005	D02	10,40,000	15-JAN-2009	Running
P003	Project 3	RP Traders	E006	D03	8,20,000	20-AUG-2010	Complete
P004	Project 4	AM Steels	E006	D03	15,10,000	21-DEC-2011	Running
P005	Project 5	KP Traders	E010	D04	20,20,000	10-OCT-2012	Running
P006	Project 6	RP Traders	E005	D02	18,40,000		Proposed

EMPLOYEE_PROJECT		
<u>EmpID</u>	<u>ProjID</u>	<u>DateOfJoin</u>
E001	P001	13-JUN-2009
E002	P001	13-JUN-2009
E007	P001	10-AUG-2009
E006	P001	10-AUG-2009
E005	P002	15-JAN-2009
E008	P002	15-JAN-2009
E003	P002	15-MAR-2009
E007	P002	15-MAR-2009
E001	P002	01-APR-2009

E006	P003	20-AUG-2010
E002	P003	20-SEP-2010
E007	P003	20-SEP-2010
E008	P003	20-SEP-2010
E006	P004	21-DEC-2011
E005	P004	25-DEC-2011
E001	P004	25-DEC-2011
E010	P005	10-AUG-2012
E011	P005	20-AUG-2012
E009	P005	20-AUG-2012
E008	P005	10-SEP-2012

DEPARTMENT			
<u>DeptID</u>	<u>DeptName</u>	<u>Location</u>	<u>HOD</u>
D01	Production-I	East Block	E001
D02	Design	West Block	E005
D03	Production-II	SE Block	E006
D04	Production-III	NE-Block	E010

DEPENDENT				
<u>DepID</u>	<u>EmpID</u>	<u>DepName</u>	<u>Relation</u>	<u>PhoneNo</u>
DEP01	E001	S. Saha	WIFE	9861303628
DEP02	E001	R. K Pati	FATHER	9861313628
DEP03	E001	G. Pati	MOTHER	9896312620
DEP01	E003	S. Devi	WIFE	9437303628
DEP01	E004	S. Nanda	FATHER	9737363628
DEP01	E006	K. Nag	WIFE	9777363629
DEP02	E006	G. Nag	SON	9077003649
DEP01	E008	B.K. Mohanty	WIFE	
DEP02	E008	G. Das	FATHER	8877883669
DEP01	E010	V. Susheela	MOTHER	9090003669
DEP01	E011	S. Naik	WIFE	9977883669
DEP02	E011	P. Naik	MOTHER	
DEP03	E011	R.Naik	SON	

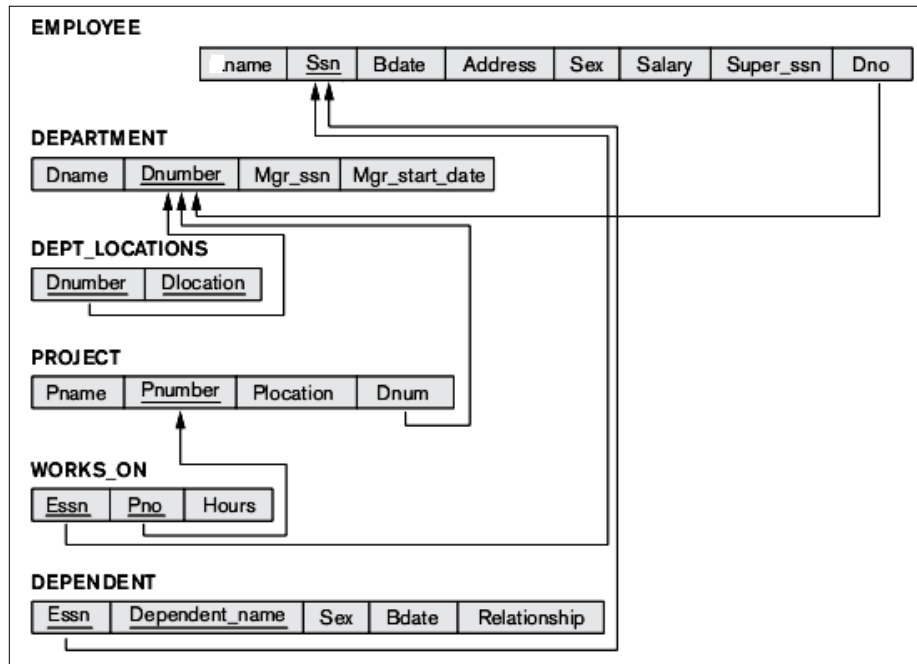
(Answer the queries using above relational database.)

1. List all employees' name, their salary, Phone numbers who are working in Production-I order by their basic salary.
2. Display the name of employee who is the boss of the entire system.
3. Display Project name and no. of employees working under it.
4. Display the Employee name and no. of employees working under them.
5. List employees name who are principal investigator of at least two Projects.
6. Display the name of all employees who doesn't have any dependent.
7. List the department names and sum of cost of all projects running under it.
8. List all department names, their location and their HOD's name.

9. List the name of all employees who are HOD of different department, their dependents and relationship with the dependent.
10. List the names of all the employees who are not associated with any project.
11. List the names of all the employees, their boss name and department name where they work.
12. List the names of all the employees who are the principal investigator of different projects, their project name, Funding agency of the project, Cost of project and status of the project.
13. List the name of all the employees who are working in Project-1 but joined later to the project.
14. List the name of all the employees who are working in Project-2 but joined on same date when the project starts.
15. List the name of all the employees who are working in Project-3 but don't belong to the same department.
16. Display the name of Employee and dependent's name whose surname is 'Pati'.
17. Display the name of employee and their dependent with phone number in which dependent's phone number starts with 98.
18. Display the list of all employees and their Father's name or Mother's name (If records are available).
19. Update/Change the location of department to 'South Block' where 'P.K. Patra' works.
20. Add 200 bucks extra to all 'Jr. Manager' who works in a department whose HOD is 'B.P. Nag'.

ASSIGNMENT- 4

Create the following schema :



EMPLOYEE		
Attribute	Type	Remarks
<u>SSN</u>	Number(8)	Primary Key
Name	Varchar2(25)	Not Null
Bdate	Date	Not Null
Address	Varchar2(25)	
Sex	Char(1)	Is like M/F
Salary	Number(10,2)	Default 0.0
Superssn	Varchar2(8)	
Dno	Varchar2(6)	Foreign key referred to Dnumber of Department

DEPARTMENT		
Attribute	Type	Remarks
<u>Dnumber</u>	Number(6)	Primary Key
Dname	Varchar2(15)	Not Null
Mgrssn	Varchar2(8)	Not Null
Mgr_start_date	Date	Default System date (Sysdate)

DEPENDENT		
Attribute	Type	Remarks
<u>Essn</u>	Number(8)	Composite Primary Key. Essn referred to ssn of Employee table.
<u>Dependent Name</u>	Varchar2(20)	
Sex	Char(1)	Is like M/F
Bdate	Date	
Relationship	Varchar2(15)	Not Null

PROJECT		
Attribute	Type	Remarks
Pname	Varchar2(20)	Not Null
<u>Pnumber</u>	Number(10)	Primary Key
Plocation	Varchar2(20)	Not Null
<u>Dnum</u>	Number(6)	Foreign key referred to Dnumber of Department

WORKS_ON		
Attribute	Type	Remarks
<u>Essn</u>	Number(8)	Composite Primary key. Foreign key Essn referred to ssn of Employee and Pno referred to Pnumber of Project
<u>Pno</u>	Number(10)	
Hour	Number(4,2)	Not Null

DEPARTMENT_LOCATION		
Attribute	Type	Remarks
<u>Dnumber</u>	Number(6)	Composite Primary, Dnumber is the foreign key referred to Dnumber of Department
<u>Dlocation</u>	Varchar2(20)	

Write SQL statements for followings and note the output of the query. In some case error message will be displayed, note down the error message and analyze it.

1. Create the above database schema.
2. Remove the attributes Sex and Bdate from Dependent Relation.
3. Suppose each department has a fixed/single location. Modify the schema of database accordingly.
(Hint: Add the uncommon attributes of DEPARTMENT_LOCATION to DEPARTMENT TABLE)
4. Add a new attribute Mobile_number (numeric 10 digit) to employee table and enforce **NOT NULL** constraint on it.
5. Create another table DEPARTMENT_MANAGER which will borrow the following properties of both the relation –

Dnumber, Dname, Start_date: From DEPARTMENT table

Mgr_Name: from *Name* field EMPLOYEE table

Constraints on table:

DEPARTMENT_MANAGER(Dnumber) references DEPARTMENT(Dnumber) *Mgr_Name* and *Start_Date* are NOT NULL.

(Hints: **CREATE TABLE AS SELECT....** Statement. Then rename *Name* attribute to *Mgr_Name*. Finally add constraints. (Multiple SQL statements))

6. Change the size of **SSN** 8 to 6 in Employee Relation
7. Change the size of **Dnumber** 6 to 7
8. Remove the constraint **Foreign Key** on *Dnumber* of DEPARTMENT_MANAGER.
9. Remove the primary key of DEPARTMENT table.
10. Insert any record into EMPLOYEE, DEPARTMENT table.
11. Drop the schema.

ASSIGNMENT- 5

(DML USE OF INSERT, UPDATE, AND DELETE STATEMENTS) (For the schema and data refer Assignment-III)

Write SQL DML statement(s) for following scenario and note the output of the query. In some case error message will be displayed, note down the error message and analyze it.

1. Change the boss of “R. K. Pati” from “A. Nanda” to “D.K. Mahallik”.
2. Update the basic salary of all employee of “Design” department by raising it to 10% of of basic salary.
3. Update the basic salary of the boss of the company to 15%.
4. Update the basic salary of all employee raised by 4% who works in “Project 1”.
5. Update the following information of “Project 6”-
 - Change the Funding Agency to “B.K.Traders”, Project Status to “Running” , set start date “10/10/14”.
 - Add the following employee to the project -
 1. “A. Bag”, Date of joining is same as date of project started
 2. “K.K.Till” date of joining is “12/12/2014”
6. Update the following dependent information of “K.K. Till” in schema -
 - “J. K. Till”, Relationship “Brother”, and Contact No. is 9861020380
 - “N.M. Till”, Relationship “Mother”, and no contact number.
7. Remove the list of all related people from the project whose status is “Completed”.
8. The organization open a new department/division *Sales*, for that update the following information in the schema -
 - Add one more department “Sales” locate at “North Block” and shift “B.P.Nag” as head of department
 - Add following two personnel to new department Sales –
 - a. P. Shyam as “Sales executive”, basic salary 20,000/-, Grade Pay 11000, contact no. 93398250013. Dependent (P. Kamala, Wife, 9861088766)
 - b. S. Reddy as “Executive officer”, basic salary 15,000/-, Grade Pay 10,000, contact no. 93398250014. Dependent (S. Reddy, Father, 9431055666) & (S. Reddy, Mother, 9431055777)

N.B. Default boss of new employee is HOD.
9. Add extra 10% to the basic salary and 30% to the Grade pay of Finance of officer.
10. Update the basic salary of following category employee –
 - Increase the basic salary of Employees with Manager specialization to extra 5%.
 - Decrease the basic salary of Employees with Officer specialization to extra 2%.

ASSIGNMENT- 6

ORACLE VIEW : USE OF CREATE INSERT, UPDATE, AND DELETE STATEMENTS)(For the schema and data refer Assignment-III)

Write SQL statement(s) for followings and note the output of the query. In some case error message will be displayed, note down the error message and analyze it. Ensure that integrity constraints enforced in original database schema.

DDL/DML Statements

1. Create a view *Employee_Personal* that will keep following information –
Employee Id, Employee Name, Department name, Specialization, Phone Number
2. Create a view *Employee_Salary* that will keep following information –
Employee Id, Employee Name, Basic Salary, Grade Pay
3. Create a view *Employee_Project_Details* that will keep following information –
Project Id, Project Name, Funding Agency, Employee name, Date of Join
4. Create a view *Project_Details* that will keep following information –
Project Id, Project Name, Department Id, Department Name, Name of Principal Investigator, Location (Same as Department location)
5. Create a view *Department_Details* that will keep following information –
Department Id, Department name, Location, Name of HOD
6. Create a view *Employee_Dependent* that will keep followings –
Employee Id, employee Name, Dependent Name, Relationship, Phone number of employee, Phone number of Dependent.
7. Using Select statement display the contents of all views.
8. Insert following row into *Employee_Salary* Views-
(‘E013’, ‘P.K. Rath’, 20000, 12500)
9. Insert following row into *Employee_Personal* Views-
(‘E013’, ‘P.K. Rath’, ‘A. Nanda’, ‘Production-I’, ‘Sr. Engineer’, NULL)
10. Update view *Employee_Project_Details* set Project Id = ‘P001’, Project Name = ‘Project 1’, Date of joining is Today where Employee Name is ‘P.K. Rath’
11. Delete a record from *Department_Details* where employee id is ‘D01’.
12. Delete a record from *Employee_Personal* whose employee id is ‘E013’.
13. Delete record from *Employee_Dependent* where employee id is ‘E001’.
14. Drop all views.

ASSIGNMENT- 7

(PL/SQL: Named Block (Procedure, Function, Trigger))
(For the schema and data refer Assignment-III)

Write PL/SQL statement(s) for followings and note the output of the query

(Database Update and o/p of block).

1. Write a **PL/SQL unnamed block** that will list the names of all the employees, their boss name, group by their department.
2. Write **PL/SQL Procedure** that will read data from employee and department table and display –
<Employee Name1>, Designation <Specialization> Working in <Department Name> getting Net Salary <Net Salary>
<Employee Name2>, Designation <Specialization> Working in <Department Name> getting Net Salary <Net Salary>
.... For all employee
Net Salary of employee = Basic Salary + Grade pay
3. Write a **PL/SQL procedure** that will accept the employee id and generate a report that will display the employee id, name of employee, his phone no., boss name, basic salary, grade pay, specialization, department name, name of project(s) with which he is working , name of his all dependent(s) if any.
4. Create another **table EmployeeLeave** having following attributes: EmployeeID, Month, Year, TotalDays, WorkingDays, LeaveWithPay, LeaveWithoutPay, TotalBillingDays

EmployeeID	Month	Year	WorkingDays	LeaveWithPay	LeaveWithoutPay	TotalBillingDays

TotalBillingDays is the derived attribute and its value is (WorkingDays - LeaveWithoutPay). Enter at least two months leave statements for all the employees. Create a **trigger** that will insert the value of TotalBillingDays after inserting each record into EmployeeLeave Table.

5. Write a **PL/SQL procedure** to generate Employee Salary Statement of a month. The procedure will accept Month and Year from user. After generating salary statement it will display salary slip of all employee in the following format –

$\text{Pay} = ((\text{Basic} + \text{GradePay}) / (\text{TotalDays})) * \text{TotalBillingDays}$

$\text{DA} = (\text{Pay}) * \text{DA\%}$ [Assume DA% is 53%]

$\text{HRA} = \text{Basic} * \text{HRA\%}$ [Assume HRA% is 10% of Basic]

$\text{GROSSPAY} = (\text{Basic} + \text{GradePay}) + \text{HRA} + \text{DA}$

$\text{PF} = \text{Basic} * \text{PF\%}$ [Assume PF% is 20%]

$\text{MatchingPF} = \text{Basic} * \text{PF\%}$ [Assume MatchingPF% is 10%]

$\text{ITAX} = \text{Basic} * \text{IT\%}$ [Assume IT% is 5%]

$\text{PTAX} = 250$ if Gross Pay is $\leq 10,000$

$= 500$ if Gross Pay is $\leq 10,000$

$\text{NETDEDUCTION} = \text{PF} + \text{ITAX} + \text{PTAX}$

$\text{NETPAY} = \text{GROSSPAY} - \text{NETDEDUCTION} + \text{MatchingPF}$

At the end it will insert all the rows into EmployeeSalaryStatement Table. The table contains following attributes : EMPID, YEAR, MONTH, BASIC, GRADEPAY, DA, HRA, GROSSPAY, PF, MATCHINGPF, ITAX, PTAX, NETDEDUCTION, NETPAY. Select the data type for the attributes accordingly.

End Notes